

SCALABLE VECTOR GRAPHICS 1.2 –
NEUERUNGEN UND ANWENDUNGEN

DIPLOMARBEIT

IM FACHBEREICH ELEKTROTECHNIK,
INFORMATIONSTECHNIK UND MEDIEN DER FACHHOCH-
SCHULE MERSEBURG

eingereicht am 02.12.2004 von
U. Menzel, 00 KTM 2, aus Merseburg

Erster Betreuer: Herr Dr. Meinike; Zweiter Betreuer: Herr Prof.
Hofmann

Vielen Dank

- ganz besonders an Herrn Dr. Meinike für die Möglichkeit zu dieser Arbeit und für die ausgezeichnete Betreuung;
- an Herrn Prof. Hofmann für die freundliche Bereitschaft, als zweiter Betreuer zu fungieren;
- natürlich auch an die Professorinnen des Studiengangs Kommunikation und Technische Dokumentation: Frau Prof. Dr. Dietzel, Frau Prof. Alexander, Frau Prof. Dr. Trundt;
- last but not least an meine Eltern für ihre stetige Unterstützung während des Studiums und an S. Tollas für die Hilfe zu Beginn dieser Arbeit.

Inhalt

Abbildungen	VII
Tabellen	VII
Listings	VII
1 Einführung	1
1.1 Das Grafikformat SVG	1
1.1.1 <u>Skalierbarkeit</u>	1
1.1.2 <u>Strukturiertheit</u>	1
1.1.3 <u>Textbasiertheit</u>	2
1.1.4 <u>Orientierung am World Wide Web</u>	2
1.2 Der W3C-Standard SVG	3
1.3 Bisherige Entwicklung	4
1.4 Working Drafts zu SVG 1.2	4
1.4.1 <u>Bedeutung für die vorliegende Arbeit</u>	4
1.4.2 <u>Beurteilung der Verbindlichkeit</u>	4
1.5 Software	5
1.6 Zielsetzung und Gliederung	8
2 SVG-Grundlagen	9
2.1 Die Extensible Markup Language	9
2.1.1 <u>XML als Meta-Sprache</u>	9
2.1.2 <u>Dokumentaufbau</u>	9
2.1.3 <u>Validierung</u>	12
2.1.4 <u>Schreibregeln</u>	13
2.1.5 <u>Namensräume</u>	14
2.2 SVG-Grundgerüst	15
2.3 Koordinatensystem und Maßeinheiten	16
2.4 Reihenfolge beim Rendern	18
2.5 Grundformen	18
2.5.1 <u>Rechtecke</u>	18
2.5.2 <u>Kreise</u>	19
2.5.3 <u>Ellipsen</u>	19
2.5.4 <u>Linien</u>	19
2.5.5 <u>Polylinien</u>	19
2.5.6 <u>Polygone</u>	19
2.6 Pfade	20
2.7 Symbole	21
2.8 Text	21
2.9 Textlinks	23

2.10	Formatierung	23
2.10.1	<u>Eigenschaften</u>	23
2.10.2	<u>Zuweisung im Tag</u>	24
2.10.3	<u>Zuweisung mit CSS-Notation</u>	25
2.10.4	<u>Ablage im Definitionsbereich</u>	27
2.10.5	<u>Ablage in externer CSS-Datei</u>	28
2.11	Externe Bilder	28
2.12	Effekte	29
2.13	Gruppen und Transformationen	30
2.14	Animationen	32
2.15	Scripting	33
2.15.1	<u>Ablage eines Scripts</u>	35
2.15.2	<u>Beispiel für ein Script</u>	35
2.15.3	<u>Eigenschaft pointer-events</u>	38
2.16	Einbindung in HTML	38
3	Neuerungen in SVG 1.2	40
3.1	Einleitung	40
3.2	Bereich Text	41
3.2.1	<u>Fließtext mit Bildern</u>	41
3.2.2	<u>Fließtext-Elemente und ihre Aufgaben</u>	42
3.2.3	<u>Fließtext-Elemente im Zusammenhang</u>	43
3.2.4	<u>Fließtext-Beispiel</u>	44
3.2.5	<u>Fließtext-Ausrichtung</u>	46
3.2.6	<u>Overflow</u>	47
3.2.7	<u>Editierbarer Text</u>	47
3.2.8	<u>Attribut requiredFonts</u>	48
3.2.9	<u>Zu erwartende Textlänge</u>	49
3.2.10	<u>Anwendungen im Bereich Text</u>	49
3.3	RCC und sXBL	49
3.3.1	<u>Vorbemerkung</u>	49
3.3.2	<u>Rendering Custom Content</u>	50
3.3.3	<u>Shadow Trees</u>	50
3.3.4	<u>RCC-Elemente und ihre Aufgaben</u>	51
3.3.5	<u>RCC-Elemente im Zusammenhang</u>	52
3.3.6	<u>RCC-Beispiel</u>	52
3.3.7	<u>XML Binding Language von SVG</u>	56
3.3.8	<u>Veränderungen in sXBL gegenüber RCC</u>	56
3.3.9	<u>XBL-Namensraum</u>	57
3.3.10	<u>sXBL-Elemente</u>	57
3.3.11	<u>Komponenten-Benennung mit QName</u>	59

3.3.12	<u>Übernehmen von Komponenten</u>	59
3.3.13	<u>Fokus-Behandlung bei Komponenten</u>	61
3.3.14	<u>sXBL-Beispiel</u>	61
3.3.15	<u>Anwendung von sXBL</u>	64
3.4	<u>Multiple Seiten</u>	64
3.4.1	<u>Konzept der multiplen Seiten</u>	64
3.4.2	<u>Trennung von Seiten</u>	65
3.4.3	<u>Elemente für Seiten</u>	65
3.4.4	<u>Beispiel für multiple Seiten</u>	66
3.4.5	<u>Anwendung der multiplen Seiten</u>	67
3.5	<u>Vektor-Effekte</u>	67
3.5.1	<u>Konzept der Vektor-Effekte</u>	67
3.5.2	<u>Element vectorEffect</u>	67
3.5.3	<u>Globale Attribute</u>	68
3.5.4	<u>Elemente für Vektor-Effekte</u>	68
3.5.5	<u>Eigenschaft vector-effect</u>	70
3.5.6	<u>Beispiel für einen Vektor-Effekt</u>	71
3.6	<u>Fixierte Transformationen</u>	72
3.7	<u>Rendering-Effekte</u>	74
3.7.1	<u>Konzept der Rendering-Effekte</u>	74
3.7.2	<u>Eigenschaft comp-op</u>	74
3.7.3	<u>Eigenschaft clip-to-self</u>	75
3.7.4	<u>Eigenschaft enable-background</u>	75
3.7.5	<u>Eigenschaft knock-out</u>	75
3.7.6	<u>Beispiel für Rendering-Effekte</u>	76
3.8	<u>Bereiche Darstellung und Farbe</u>	77
3.8.1	<u>Füllung des Hintergrunds</u>	77
3.8.2	<u>Element solidColor</u>	77
3.8.3	<u>Eigenschaft rendering-color-space</u>	78
3.8.4	<u>XLink-Attribut für style-Element</u>	78
3.8.5	<u>Akzentuierung</u>	79
3.8.6	<u>Attribut shadowInherit</u>	79
3.8.7	<u>Attribute für filter-Element</u>	80
3.8.8	<u>Eigenschaften overlay und overlay-host</u>	80
3.8.9	<u>Cursor mit SVG-Mitteln</u>	82
3.9	<u>Audio und Video</u>	82
3.10	<u>Bereich Animation</u>	83
3.10.1	<u>Element animation</u>	83
3.10.2	<u>Element transition</u>	84
3.10.3	<u>Attribute transIn und transOut</u>	85
3.10.4	<u>Beispiel für Übergangs-Effekte</u>	85

3.11	<u>Bereich Navigation</u>	86
3.11.1	<u>Erweiterte Links</u>	86
3.11.2	<u>Fokus und die Eigenschaft focusable</u>	87
3.11.3	<u>Navigations-Reihenfolge</u>	88
3.11.4	<u>Tooltipps</u>	88
3.11.5	<u>Beispiel für den Bereich Navigation</u>	88
3.12	<u>Streaming</u>	90
3.12.1	<u>Zielsetzung</u>	90
3.12.2	<u>Attribut streamedContents</u>	90
3.12.3	<u>Attribut timelineBegin</u>	91
3.13	<u>Schrittweises Rendern</u>	91
3.14	<u>Auflösungsabhängige Bild-Inhalte</u>	93
3.14.1	<u>Element multimage</u>	93
3.14.2	<u>Element subImageRef</u>	93
3.14.3	<u>Element subImage</u>	93
3.14.4	<u>Beispiel für auflösungsabhängige Bild-Inhalte</u>	93
3.15	<u>Technische Details und Optimierung</u>	94
3.15.1	<u>Element deviceColor</u>	94
3.15.2	<u>Element prefetch</u>	95
3.15.3	<u>Element switch als Kind-Element</u>	96
3.15.4	<u>Attribut requiredFormats</u>	97
3.15.5	<u>Häufigkeit von grafischen Elementen</u>	97
3.15.6	<u>Attribut snapshotTime</u>	97
3.16	<u>Nicht-grafische Erweiterungen</u>	97
3.16.1	<u>XLink-Attribut</u>	97
3.16.2	<u>Element metadata</u>	97
3.17	<u>Scripting und DOM</u>	98
3.17.1	<u>Elemente script und handler</u>	98
3.17.2	<u>Neuerungen bei script und handler</u>	99
3.17.3	<u>Beispiel für Element handler</u>	100
3.17.4	<u>DOM-Erweiterungen</u>	101
3.17.5	<u>Erweiterte Programmier-Schnittstellen</u>	102
4	<u>Zusammenfassung</u>	103
	<u>Print- und Online-Ressourcen</u>	CVI

Abbildungen

Abbildung 1: SVG Roadmap vom 24.11.2004.....	5
Abbildung 2: Amaya und Adobe SVG Viewer im Vergleich.....	6
Abbildung 3: Jasc WebDraw, ein Authoring Tool für SVG.....	7
Abbildung 4: SVG-Koordinatensystem – koordsys.svg.....	17
Abbildung 5: Grundformen in SVG – grndform.svg.....	20
Abbildung 6: Pfade als Kreissegmente – path.svg.....	20
Abbildung 7: Mehrzeiliger Text mit tspan – tspan.svg.....	23
Abbildung 8: Effekte in SVG – effects.svg.....	30
Abbildung 9: Transformationen in SVG – transfo.svg.....	32
Abbildung 10: DOM-Baum in Jasc WebDraw.....	34
Abbildung 11: Formatierung beeinflussen durch Scripting – script.svg.....	37
Abbildung 12: Fließtext – flow.svg.....	46
Abbildung 13: Komponenten mit RCC – rckreis.svg und rckreis.js.....	56
Abbildung 14: Vektor-Effekte – effvecto.svg.....	71
Abbildung 15: Fixierte Transformationen – transfix.svg.....	74

Tabellen

Tabelle 1: Entity-Referenzen in XML.....	14
Tabelle 2: Einheiten in SVG.....	18
Tabelle 3: Wichtige Eigenschaften zur Formatierung von SVG.....	24
Tabelle 4: Ausrichtung von Fließtext.....	47
Tabelle 5: sXBL- und RCC-Elemente.....	58

Listings

Listing 1: XML-Grundgerüst – grundxml.xml.....	12
Listing 2: SVG-Grundgerüst – grundsvg.svg.....	16
Listing 3: Mehrzeiliger Text mit tspan – tspan.svg.....	22
Listing 4: Script zur Änderung der Strichfarbe.....	36
Listing 5: Fließtext – flow.svg.....	45

Listing 6: Editierbarer Text mit editable.....	47
Listing 7: Attribut requiredFonts.....	48
Listing 8: Rendering Custom Content – rcckreis.svg.....	54
Listing 9: Shadow Tree-Beeinflussung – rcckreis.js.....	55
Listing 10: SVG's XML Binding Language – sxblkreis.svg.....	62
Listing 11: Shadow Tree-Beeinflussung – sxblkreis.js.....	63
Listing 12: Multiple Seiten – pages.svg.....	66
Listing 13: Vektor-Effekte – effvecto.svg.....	71
Listing 14: Fixierte Transformationen – transfix.svg.....	73
Listing 15: Rendering-Effekte – effrend.svg.....	76
Listing 16: Eigenschaften background-fill, background-fill-opacity.....	77
Listing 17: Element solidColor.....	78
Listing 18: Attribut shadowInherit.....	80
Listing 19: Eigenschaft overlay – paint12.svg.....	81
Listing 20: Elemente audio und video – audiovid.svg.....	82
Listing 21: Element animation.....	84
Listing 22: Übergangs-Effekte – anim12.svg.....	86
Listing 23: Erweiterte Links.....	87
Listing 24: Navigation in SVG 1.2 – navi12.svg.....	89
Listing 25: Schrittweises Rendern – renprog.svg.....	92
Listing 26: Auflösungsabhängige Bild-Inhalte.....	94
Listing 27: Element deviceColor.....	95
Listing 28: Element prefetch.....	96
Listing 29: Element metadata.....	98
Listing 30: Element handler – script12.svg.....	101

1 Einführung

1.1 Das Grafikformat SVG

SVG (Scalable Vector Graphics) bezeichnet ein Grafikformat zur Beschreibung von zweidimensionalen Vektorgrafiken, das auf der *Extensible Markup Language (XML)* basiert. SVG besitzt damit eine Reihe von Vorzügen.

1.1.1 Skalierbarkeit

In der Computergrafik herrschen traditionell zwei Konzepte vor: *Vektorgrafik* und *Rastergrafik*. Während Rastergrafiken aus einzelnen Bildpunkten (*Pixel*) zusammengesetzt sind, bestehen Vektorgrafiken aus Objekten (z. B. ein Kreis), die durch Koordinaten (z. B. Kreismittelpunkt) und bestimmte Parameter (z. B. Radius) definiert sind. Während bei Rastergrafik-Formaten (*.gif*, *.png*, *.tiff*, ...) ab einer bestimmten Auflösung die Pixel sichtbar werden, können Vektorgrafik-Formate (*.dxf*, *.dwg*, *.cdr*, *.ai*, ...) und damit auch Grafiken im SVG-Format (*.svg*) in

- hoher Qualität und
- hoher Auflösung

angezeigt werden. Da die Objekte in SVG-Grafiken nicht aus (unzähligen) Pixeln zusammengesetzt, sondern durch Koordinaten und Parameter beschrieben werden, besitzen SVG-Grafiken zudem eine

- kleinere Dateigröße als Rastergrafiken.

1.1.2 Strukturiertheit

SVG basiert auf XML. SVG-Grafiken sind XML-Dokumente, in denen die Inhalte in einer hierarchischen Struktur abgelegt sind. Die Struktur von SVG-Dokumenten ermöglicht

- gezielten Zugriff auf Grafikobjekte,
- Veränderung von Bestandteilen der Struktur.

1.1.3 Textbasiertheit

Als XML-Dokumente werden SVG-Dokumente im *ASCII-Format* (*ASCII = American Standard Code for Information Interchange*) gespeichert. Dieses Text-Format ist:

- plattform- und anwendungsunabhängig,
- weit verbreitet.

SVG-Dokumente sind somit

- mit einfachen Text-Editoren erstellbar.

Weitere Vorteile, die sich aus der Textbasiertheit ergeben, sind:

- Zugänglichkeit für Suchmaschinen,
- Möglichkeit zum Einsehen des Quelltextes.

Unzertrennbar verbunden mit dem letztgenannten Vorteil ist leider auch ein größerer Nachteil: Der Quelltext eines SVG-Dokuments liegt jeder Person offen. Alle enthaltenen Daten sind somit nicht geschützt, was missbraucht werden kann. Jedoch sollen zukünftig Schutzmechanismen in SVG integriert werden.

1.1.4 Orientierung am World Wide Web

Basierend auf XML, einem offiziellen Standard des World Wide Web (im Folgenden »Web« genannt), ist SVG selbst ein solcher Standard. Offizielle Web-Standards können

- lizenzfrei

eingesetzt werden. Zudem eignet sich SVG hervorragend für den Einsatz zusammen mit anderen Techniken im Web:

- Einsatz von Script-Sprachen,
- Server-seitige Generierung von SVG,
- Einbindung von Daten aus Datenbanken.

1.2 Der W3C-Standard SVG

Hinter SVG steht das *World Wide Web Consortium (W3C)*, eine Vereinigung von Firmen, Organisationen und Institutionen, die sich zur Aufgabe gemacht hat, Standards für das Web zu entwickeln.¹ Bekannte Standards, die auf der Arbeit des W3C beruhen, sind u. a. die *Hypertext Markup Language (HTML)*, die *Extensible Markup Language (XML)* oder auch die *Cascading Style Sheets (CSS)*.

Das W3C hat seine Arbeit dabei in verschiedene Bereiche aufgeteilt. So gibt es z. B. einen Bereich, wo die grundlegenden Web-Technologien wie XML entwickelt werden (*Architecture Domain*). In einem anderen Bereich beschäftigt man sich mit Benutzer-Schnittstellen für das Web (*Interaction Domain*). Die Entwicklung von SVG ist im letztgenannten Bereich angesiedelt, innerhalb der Grafik-Aktivitäten (*Graphics Activity*) des W3C.²

Die eigentliche Arbeit des W3C besteht im Erstellen von Dokumenten, welche die Standards detailliert beschreiben (Spezifikationen). Dabei durchlaufen die Dokumente den sog. *Recommendation Process* – ein Verfahren, bei dem die Dokumente unter Einbeziehung der Öffentlichkeit immer weiter entwickelt werden, bis sie die gewünschte Reife erreicht haben und als offizielle Spezifikation verabschiedet werden. Während des Verfahrens werden die Dokumente ihrem Entwicklungsstand entsprechend benannt. Dabei sind möglich:³

- *Working Draft* (Arbeitsentwurf),
- *Candidate Recommendation* (Empfehlungskandidat),
- *Proposed Recommendation* (Empfehlungsvorschlag),
- *Recommendation* (Empfehlung; auch Spezifikation).

1 W3C-Homepage: <http://www.w3.org/>

2 SVG-Homepage beim W3C: <http://www.w3.org/Graphics/SVG/>

3 zitiert nach [SELF80]: <http://de.selfhtml.org/intro/hilfsmittel/dokus.htm#recommendations>

1.3 Bisherige Entwicklung

Die erste Spezifikation zu SVG (SVG 1.0) wurde im September 2001 verabschiedet. Im Januar 2003 folgte die Spezifikation zu SVG 1.1.⁴ Sie brachte geringfügige Änderungen und eine Modularisierung von SVG. Derzeit befindet sich die Version 1.2 als Working Draft in Bearbeitung.

1.4 Working Drafts zu SVG 1.2

1.4.1 Bedeutung für die vorliegende Arbeit

Eine wesentliche Grundlage der vorliegenden Arbeit war der siebente Working Draft zu SVG 1.2 vom 10.05.2004.⁵ Während des Schreibens der Arbeit wurde am 27.10.2004 der achte Working Draft veröffentlicht.⁶ Bei ihm handelt es sich um den letzten Working Draft (*Last Call*) vor der Veröffentlichung der Candidate Recommendation. Er aktualisiert den siebenten Working Draft bzw. löst ihn ab. Somit stellt der achte Working Draft auch die wichtigste Grundlage dieser Arbeit dar. Gleichzeitig repräsentiert sein Entwicklungsstand den Stand der SVG-Entwicklung, der in dieser Arbeit berücksichtigt werden kann.

1.4.2 Beurteilung der Verbindlichkeit

Als Arbeitsentwürfe besitzen die Working Drafts keinen verbindlichen Charakter. Jedoch werden die darin beschriebenen Features von der Arbeitsgruppe, die sich beim W3C mit SVG beschäftigt (*SVG Working Group*), als »annähernd stabil« angesehen.⁷ Auch der Zeitplan für die Entwicklung von SVG, die sog. *Roadmap*⁸, lässt erkennen, dass sich SVG 1.2 im letzten Drittel seiner Entwicklung befindet (siehe Abbildung 1). Beide Anzeichen ermutigen anzunehmen, dass die Features, insbesondere die des ach-

4 siehe [SVG11REC]

5 siehe [SVG12WD7]

6 siehe [SVG12WD8]

7 vgl. [SVG12WD7] Abschnitt Status of this Document

8 aktuelle Roadmap unter [SVGRoad]

ten Working Drafts, letztendlich auch in die offizielle Spezifikation gehen.

Scalable Vector Graphics Roadmap						
This page contains descriptions of drafts that the SVG WG (Scalable Vector Graphics Working Group) is working on or has completed. If you want to follow the development of SVG, this page should be the place to start. Note that the content of this page changes often, so you should not draw any conclusions from the data here.						
The following table lists the publication dates (past or <i>expected</i>) for all of the SVG documents. The predicted dates are intended to be indicative only.						
Document	FWD	Next WD	LC	CR	PR	REC
SVG 1.0	11 Feb 1999	-	03 Mar 2000	02 Aug 2000	19 July 2001	5 Sep 2001
SVG 1.1	30 Oct 2001	-	15 Feb 2002	30 Apr 2002	15 Nov 2002	14 Jan 2003
SVG Mobile Profiles	30 Oct 2001	-	15 Feb 2002	30 Apr 2002	15 Nov 2002	14 Jan 2003
SVG Mobile 1.2	9 Dec 2003	-	13 Aug 2004	[Nov 2004]	[Mar 2005]	[May 2005]
SVG 1.2	11 Nov 2002	-	27 Oct 2004	[Dec 2004]	[Mar 2005]	[May 2005]
DOM Level 3 Events	01 Sep 2000	-	31 Mar 2003	[Dec 2004]	[Mar 2005]	[May 2005]
DOM Level 3 XPath	18 Jun 2001	-	28 Mar 2002	31 Mar 2003	[Mar 2005]	[May 2005]
sXBL	01 Sep 2004	[Nov 2004]	[Feb 2005]	[Apr 2005]	[Sep 2005]	[Nov 2005]
SVG Print	15 July 2003	[Dec 2004]	[Mar 2005]	[Jun 2005]	[Sep 2005]	[Nov 2005]
Authoring Tool Guidelines	[Feb 2005]	-	-	-	-	-
Accessibility Techniques	[May 2005]	-	-	-	-	-

Legend: **FWD** = First working draft; **LC** = last call for comments (i.e., last WD); **CR** = Candidate Recommendation; **PR** = Proposed Recommendation; **REC** = W3C Recommendation. [Feb 2005] = expected date.

Note: DOM Level 3 Events is currently parked as a Note dated 07 Nov 2003. DOM Level 3 XPath is currently parked as a Note dated 07 Nov 2003. sXBL was previously RCC and was part of SVG 1.2.

Abbildung 1: SVG Roadmap vom 24.11.2004

1.5 Software

Zum Anzeigen von SVG ist derzeit ein Browser-PlugIn oder ein Viewer erforderlich. Es existieren zwar bereits Browser, die SVG nativ unterstützen, aber leider können diese Browser wichtige SVG-Features nicht umsetzen.

Zum Beispiel unterstützt der Browser *Amaya* vom W3C keine grafischen Effekte, Filter und Animationen.⁹ Auch CSS-Eigenschaften werden von ihm nur äußerst eingeschränkt interpretiert. So wird z. B. die CSS-Eigenschaft **text-anchor** aus SVG 1.0 nicht umgesetzt, wie links in Abbildung 2 ersichtlich. Im Vergleich dazu

⁹ zum Zeitpunkt des Tests lag Version 8.5 vor; mittlerweile ist Version 8.7 als Open Source verfügbar: <http://www.w3.org/Amaya/User/BinDist.html> (22.10.2004)

rechts die Darstellung vom *Adobe SVG Viewer*, bei dem die Implementierung von SVG am weitesten fortgeschritten ist.¹⁰

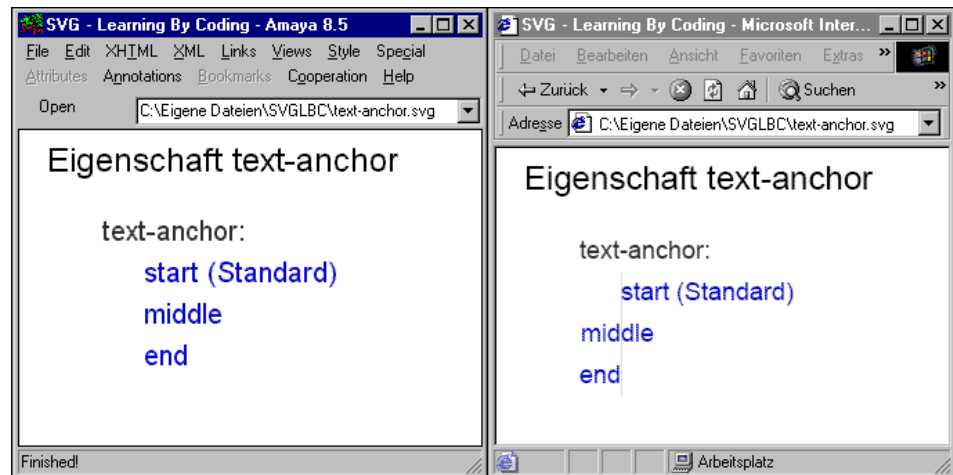


Abbildung 2: Amaya und Adobe SVG Viewer im Vergleich

Wie zuvor erwähnt, wird SVG derzeit am besten vom *Adobe SVG Viewer* unterstützt. Das Browser-PlugIn wird in zwei Versionen kostenfrei zum Download angeboten:

- Der *Adobe SVG Viewer 3.02* ist die offizielle, aktuelle Version.¹¹ Er unterstützt die SVG-Version 1.0 weitestgehend.
- Der *Adobe SVG Viewer 6.0 Development Release 1* wird Entwicklern für Testzwecke angeboten.¹² Er stellt teilweise schon Möglichkeiten aus SVG 1.2 zur Verfügung. Alle grafischen Ausgaben in dieser Arbeit sind mit ihm erzeugt. Wegen der Verständlichkeit soll er im Folgenden kurz mit »ASV 6.0 preview 1« bezeichnet werden.

Zum Erstellen von SVG gibt es eine ganze Reihe von Werkzeugen. SVG-Dokumente können erstellt werden mit professionellen Vektorgrafik-Programmen, die SVG-Export

¹⁰ Quelle SVG-Datei: Offline-Variante von [MEIN02]

¹¹ Download unter [ASV302]

¹² Download unter [ASV60p1]

ermöglichen (wie *Adobe Illustrator*¹³) oder mit Programmen, die auf SVG spezialisiert sind (wie *Jasc WebDraw*¹⁴, siehe auch Abbildung 3).

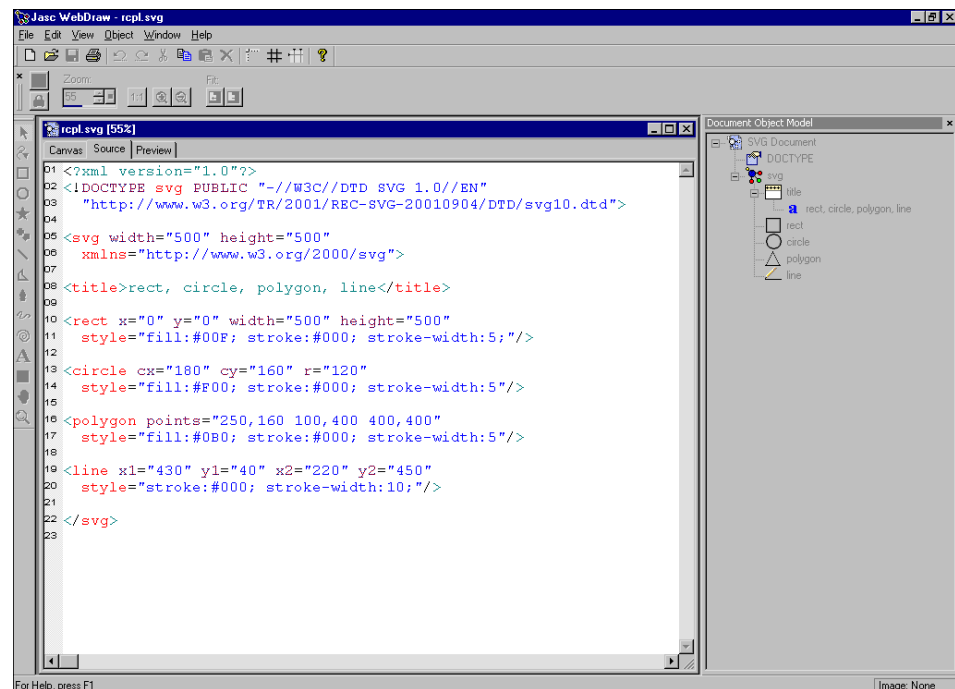


Abbildung 3: Jasc WebDraw, ein Authoring Tool für SVG

Aufgrund des XML-Formates können auch XML-Editoren (wie *Altova XMLSpy*¹⁵) verwendet werden. Prinzipiell reicht auch ein einfacher Texteditor aus.

An dieser Stelle soll nicht weiter auf SVG-Software eingegangen werden, da bereits eine Fülle davon existiert und diese zu beschreiben, nicht Schwerpunkt dieser Arbeit ist. Zudem hängt der Einsatz von Software sicherlich auch mit persönlichen Vorlieben zusammen.¹⁶

¹³ weitere Infos: <http://www.adobe.com/products/illustrator/>

¹⁴ weitere Infos: <http://www.jasc.com/products/webdraw/>

¹⁵ weitere Infos: http://www.altova.com/products_ide.html

¹⁶ eine vollständige Auflistung von SVG-unterstützender Software, die regelmäßig aktualisiert wird, findet sich unter: <http://www.w3.org/Graphics/SVG/SVG-Implementations.htm#>

1.6 Zielsetzung und Gliederung

Ziel dieser Arbeit ist es, die wesentlichen Neuerungen in SVG 1.2 aufzuzeigen und anhand von Beispielen ihre Anwendungsmöglichkeiten zu veranschaulichen.

Zunächst werden die SVG-Grundlagen in einem Abschnitt separat behandelt. Hier werden einerseits die XML-Grundlagen dargestellt sowie andererseits die grundlegenden Möglichkeiten aus den vorausgegangenen W3C-Spezifikationen beschrieben. Darauf folgt der Hauptteil, der sich mit den Neuerungen in SVG 1.2 befasst. Abschließend sollen die wichtigsten Aspekte noch einmal kurz zusammengefasst werden.

2 SVG-Grundlagen

2.1 Die Extensible Markup Language

2.1.1 XML als Meta-Sprache

Die Extensible Markup Language wurde 1998 vom W3C in der Version 1.0 als Spezifikation verabschiedet.¹⁷ XML bedeutet auf Deutsch »Erweiterbare Auszeichnungssprache«. Es bezeichnet damit eine Sprache, mit der weitere Sprachen für die verschiedensten Zwecke entworfen werden können. So ist SVG eine (Sprach-)Anwendung von XML. Andere Anwendungen sind z. B. *XHTML* oder *MathML*.

Grundidee von XML und aller XML-Anwendungen ist die Trennung von Inhalt, Struktur und Formatierung eines Dokuments. Daraus resultieren erhebliche Vorteile, wie z. B. die Möglichkeit, Inhalte medien- und anwendungsunabhängig zu veröffentlichen.

Realisiert wird diese Grundidee durch eine spezielle Syntax, bestehend aus *Tags* (sprich: Tägs). Die Tags bilden das *Markup*, d. h. sie zeichnen den Inhalt aus.

2.1.2 Dokumentaufbau

Inhalt wird immer von einem Start-Tag und einem Ende-Tag umschlossen. Die Gesamtheit dieses Konstrukts bildet die Grundeinheit eines XML-Dokumentes, das *Element*:

```
<tag>Inhalt</tag> .
```

Es gibt auch *leere Elemente*, die keinen Inhalt besitzen:

```
<tag/> .
```

Tags werden mit einem *Elementnamen* bezeichnet. Ein Start-Tag kann zudem *Attribute* besitzen, die aus einem *Attributnamen* und einem *Attributwert* bestehen, sofern Attribute für das bezeichnete Element definiert sind:

¹⁷ XML 1.0 liegt derzeit in der dritten Ausgabe vor: <http://www.w3.org/TR/2004/REC-xml-20040204/>

```
<elementname attributname="attributwert">
  Inhalt
</elementname> .
```

Elemente können *Unterelemente* enthalten (*Verschachtelung*). Dabei muss das Unterelement vom Start-Tag und Ende-Tag des Elementes umgeben sein. Das Element wird dann als *Eltern-Element*, das Unterelement als *Kind-Element* bezeichnet:

```
...
<elementname attributname="attributwert">
  <unterelementname attributname="attributwert">
    Inhalt
  </unterelementname>
</elementname>
... .
```

Die Verschachtelung muss konsequent eingehalten werden, bis hin zum sog. *Wurzelement*, das alle anderen Elemente in einem XML-Dokument umschließt:

```
...
<wurzelementname>
  <elementname attributname="attributwert">
    <unterelementname
      attributname="attributwert">
        Inhalt
      </unterelementname>
    </elementname>
  </wurzelementname> .
```

Die Definition aller in einem XML-Dokument erlaubten Elemente, ihrer Verschachtelung, ihrer Attribute, deren mögliche Attributwerte sowie weiterer Festlegungen erfolgt in der sog. *Document Type Definition (DTD)*. Dies ist ein Regelwerk, das genau eingehalten werden muss, wenn *gültige* XML-Dokumente erstellt werden sollen. Dann stellt die DTD überhaupt erst die Grundlage für solche XML-Dokumente dar, da sie deren Bestandteile und Struktur bestimmt. Eine solche DTD ist auch die DTD zu SVG 1.0, die vom W3C mit der SVG 1.0-Spezifikation verabschiedet wurde.

Mit der *DOCTYPE-Deklaration* wird ein XML-Dokument einer bestimmten DTD zugeordnet. Dabei sind *interne und externe DTDs* möglich.

Interne DTD

Bei einer internen DTD stehen die Definitionen direkt im XML-Dokument. Nach **DOCTYPE** folgen der Wurzelement-Name und in eckigen Klammern die Definitionen:

```
...
<!DOCTYPE wurzelementname [Definitionen] >
....
```

Externe DTD

Im Falle einer externen DTD wird auf eine DTD-Datei verwiesen: Wenn sich die Datei auf dem System befindet, folgen nach dem Wurzelement-Namen **SYSTEM** und der **Pfad** der Datei:

```
...
<!DOCTYPE wurzelementname SYSTEM "Pfad">
....
```

Handelt es sich um eine öffentlich bereitgestellte DTD, folgt auf den Wurzelement-Namen der sog. *öffentliche Bezeichner (Public Identifier)*. Er setzt sich zusammen aus **PUBLIC** und einer bestimmten Zeichenfolge. Danach wird der sog. *System-Bezeichner (System Identifier)* aufgeführt. Die *DOCTYPE-Deklaration* für ein SVG-Dokument, das auf SVG 1.0 beruht, lautet z. B. folgendermaßen:

```
...
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-
  20010904/DTD/svg10.dtd">
....
```

Für ein XML-Dokument, das einer externen, öffentlichen DTD zugeordnet ist, bedeutet das:

```
...
<!DOCTYPE wurzelementname PUBLIC "... "...">
```

```

<wurzelementname>
  <elementname attributname="attributwert">
    <unterelementname
      attributname="attributwert">
      Inhalt
    </unterelementname>
  </elementname>
</wurzelementname> .

```

Die *XML-Deklaration* in der ersten Zeile komplettiert das Grundgerüst eines XML-Dokuments. Sie enthält

- die *XML-Version*, gebräuchlich ist: **version="1.0"**,
- einen *Zeichensatz*, z. B.: Latin 1, Westeuropa: **encoding="ISO-8859-1"**,
- die *Angabe, ob eine interne oder externe DTD verwendet wird*: **standalone="yes"** oder **"no"** .

Listing 1: XML-Grundgerüst – grundxml.xml

```

<?xml version="1.0" encoding="ISO-8859-1"
standalone="no"?>
<!DOCTYPE wurzelementname PUBLIC "... " ">
<wurzelementname>
  <elementname attributname="attributwert">
    <unterelementname attributname="attributwert">
      Inhalt
    </unterelementname>
  </elementname>
</wurzelementname>

```

2.1.3 Validierung

Wenn XML-Dokumente verarbeitet werden sollen, werden sie zuvor von einer Software (*Parser, XML-Prozessor*¹⁸) überprüft. Überprüft werden die Korrektheit der Syntax und die Struktur des XML-Dokuments. Dabei gibt es zwei Abstufungen:

¹⁸ im Adobe SVG Viewer und aktuellen Browsern integriert

- *well-formed*:
Das Dokument ist *wohlgeformt*, d. h. die Syntax und die Verschachtelung sind korrekt.
- *valid*:
Das Dokument ist *gültig*: das Dokument ist *wohlgeformt*; zusätzlich entspricht es der verwendeten *DTD*.

Diesen Kriterien entsprechend existieren zwei Arten von Parsern. *Nicht validierende Parser* überprüfen die Wohlgeformtheit (*well-formedness*) eines Dokuments, während *validierende Parser* zusätzlich die Übereinstimmung eines Dokuments mit den in der DTD definierten Regeln überprüfen.

Werden von einem Parser Syntax-Fehler festgestellt, führt das zum sofortigen Abbrechen der Verarbeitung eines Dokuments: Syntax-Fehler machen XML-Dokumente unbrauchbar. Deswegen ist korrekte Syntax beim Erstellen von XML- (und SVG-) Dokumenten wichtig, woraus sich einige essentielle Schreibregeln ergeben.

2.1.4 Schreibregeln

(nach [FIBI02] S.42–49)

- alle Tags müssen mit `<` und `>` versehen sein, Ende-Tags bzw. leere Elemente mit `/` geschlossen werden.
- Attributwerte müssen in einfachen oder doppelten Anführungszeichen stehen:
`...='attributwert'` oder `...="attributwert"`.

Namen für Elemente und Attribute:

- dürfen nur mit Buchstaben, Unterstrich beginnen, nicht erlaubt am Anfang sind *xml* oder *XML* (vorbehalten für *Processing Instructions*).
- Groß- und Kleinschreibung wird unterschieden.
- Buchstaben, Ziffern, Unterstriche, Bindestriche, Punkte sind erlaubt.
- müssen (für gültige Dokumente) wie in der DTD geschrieben werden.

- alle Elemente müssen korrekt ineinander verschachtelt sein.
- *Kommentare* können innerhalb von `<!--` und `-->` eingefügt werden: `<!-- ein Kommentar -->`.
- Inhalte, die vom Parser nicht überprüft werden sollen, können in *CDATA-Abschnitten* abgelegt werden: `<![CDATA[Inhalte]]>`.
- einige Zeichen sind ausschließlich für XML-Code vorgesehen. Sie müssen, wenn sie in einem bestimmten Kontext auftreten, durch vordefinierte Zeichen-Kombinationen (*Entity-Referenzen*) ersetzt werden:

Zeichen	&	<	>	"	'
Ersatz	<code>&amp;</code>	<code>&lt;</code>	<code>&gt;</code>	<code>&quot;</code>	<code>&apos;</code>

Tabelle 1: Entity-Referenzen in XML

2.1.5 Namensräume

Ein Namensraum (*namespace*) ist hilfreich, wenn verschiedene, auf XML basierende Sprachen in einem Dokument kombiniert werden sollen. Dabei verhindert er Konflikte zwischen möglicherweise identischen Elementnamen oder Attributnamen.

Ein Namensraum wird im Wurzelement eines XML-Dokumentes mit dem Attribut `xmlns` festgelegt (*Standard-Namensraum*). Für weitere Sprachen können weitere Namensräume angegeben werden. Dabei folgt dem Attribut `xmlns` ein Doppelpunkt, ein frei wählbares *Präfix* sowie die Namensraum-Angabe, die zwar wie eine Web-Adresse aussieht, aber nicht wie eine solche fungiert. Die Namensraum-Angabe dient hier nur zum eindeutigen Bezeichnen des Namensraums:

```
...
<wurzelement xmlns="http://example.org/bsp"
  xmlns:praefix="http://example.net/bsp">
... .
```

Im Verlauf des Dokuments muss dann allen Elementnamen, die nicht zum Standard-Namensraum gehören, das gewählte Präfix vorangestellt werden:

```
...
<prae:elementname>Inhalt</prae:elementname>
... .
```

Als Ergebnis können dann verschiedene *XML-Dialekte* in einem Dokument verwendet werden.

2.2 SVG-Grundgerüst

Als Anwendung von XML besitzen SVG-Dokumente auch ein XML-typisches Grundgerüst (siehe [XML-Grundgerüst](#), Listing 1, S.12). In der ersten Zeile befindet sich also die XML-Deklaration. Als Zeichensatz soll hier *Latin 1, Westeuropa* angegeben werden, zusätzlich der Hinweis, dass eine externe DTD verwendet wird:

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="no"?>
... .
```

Für SVG spezifisch ist nun der Wurzelement-Name **svg** und die vom W3C zur Verfügung gestellte DTD. Beides wird in der DOCTYPE-Deklaration angegeben. **PUBLIC** signalisiert, dass es sich um eine öffentliche DTD handelt; danach wird die Zeichenfolge, die die DTD bezeichnet, und der Verweis auf die DTD-Datei angegeben:

```
...
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
"http://www.w3.org/TR/2001/REC-SVG-
20010904/DTD/svg10.dtd">
<svg ...>
...
</svg> .
```

Der für SVG vorgesehene *Namensraum* kann im Wurzelement als Attributwert von **xmlns** angegeben werden. Zum Betiteln und

Beschreiben des SVG-Dokuments können die Elemente **title** und **desc** verwendet werden. Zudem soll mit dem Element **defs** ein Bereich eingefügt werden, der für Definitionen reserviert ist (*Definitionsbereich*). Danach folgen die Grafik-Beschreibungen, um die es bei SVG geht:

```
...
<svg xmlns="http://www.w3.org/2000/svg">
  <title><!-- Titel --></title>
  <desc><!-- Beschreibung --></desc>
  <defs><!-- Definitionsbereich --></defs>
  <!-- Grafik-Beschreibungen -->
</svg> .
```

Werden alle beschriebenen Bestandteile zusammengefügt, ergibt sich das folgende SVG-Grundgerüst:

Listing 2: SVG-Grundgerüst – grundsvg.svg

```
<?xml version="1.0" encoding="ISO-8859-1"
standalone="no"?>

<!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">
  <title><!-- Titel--></title>
  <desc><!-- Beschreibung --></desc>
  <defs><!-- Definitionsbereich --></defs>

  <!-- Grafik-Beschreibungen -->

</svg>
```

2.3 Koordinatensystem und Maßeinheiten

Der Ursprung des Koordinatensystems in SVG befindet sich standardmäßig (*initial*) in der linken oberen Ecke der Zeichenfläche (*canvas*). Die X-Achse verläuft in positiver Richtung horizontal

nach rechts; die Y-Achse vertikal nach unten, ebenso in positiver Richtung (siehe Abbildung 4).

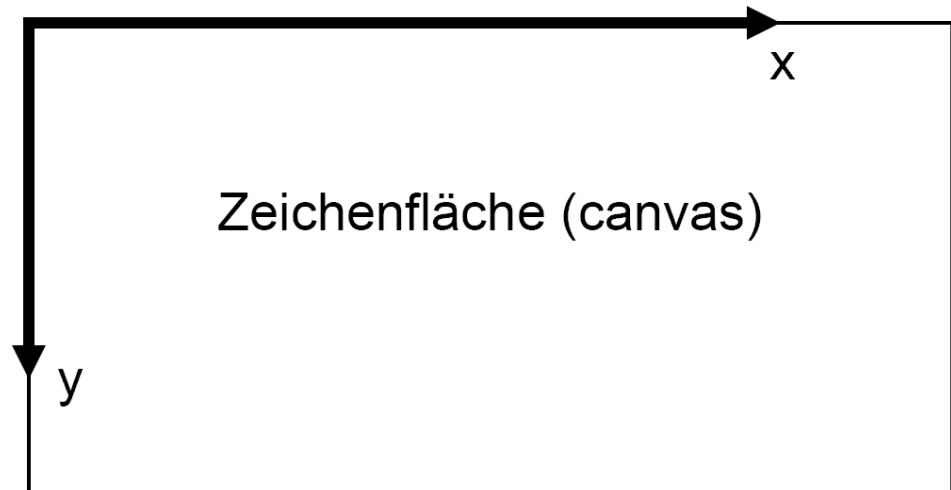


Abbildung 4: SVG-Koordinatensystem – koordsys.svg

Das Koordinatensystem kann vom SVG-Anwender bei Bedarf verändert werden. Mit dem Attribut **transform** sind u. a. Verschiebungen, Vergrößerungen oder Drehungen des Koordinatensystems für eine Reihe von Elementen möglich (siehe Transformationen in SVG, Abbildung 9, S.32). Das veränderte Koordinatensystem wird dann als *benutzerdefiniertes Koordinatensystem* bezeichnet.

In SVG stehen nach [FIBI02], S.94, die in Tabelle 2 aufgeführten Maßeinheiten zur Verfügung.

Kürzel	Bemerkung
px	Pixel: relativ zur Bildschirmauflösung und zum benutzerdefinierten Koordinatensystem; das Weglassen eines Einheiten-Kürzels entspricht ebenfalls px
em	relativ zur Größe eines verwendeten Fonts

Kürzel	Bemerkung
ex	relativ zur Höhe des Buchstabens »x« eines verwendeten Fonts
%	Prozentwert: relativ zum benutzerdefinierten Koordinatensystem
in	für inch
cm	für Zentimeter
mm	für Millimeter
pt	für Points (= 1/72 inch)
pc	für Picas (1 Pica = 12 Points)

Tabelle 2: Einheiten in SVG

2.4 Reihenfolge beim Rendern

Beim *Rendern*, dem Abbilden von Grafik-Objekten auf der Zeichenfläche, werden die einzelnen Objekte nach dem sog. *Painter's Model* abgebildet. Hiernach werden auf der Zeichenfläche die Objekte, die im Quelltext des SVG-Dokuments zuerst erscheinen, von den darauf folgenden Objekten im Quelltext übermalt.

2.5 Grundformen

SVG stellt Elemente für die Beschreibung von grundlegenden grafischen Objekten zur Verfügung (vgl. dazu Abbildung 5):

2.5.1 Rechtecke

Werden mit dem Element **rect** beschrieben. Attribute existieren für die Koordinate des linken oberen Eckpunktes, Breite und Höhe des Rechtecks sowie die Rundung der Eckpunkte:

```
<rect x="..." y="..." width="..." height="..."
rx="..." ry="..." /> .
```

2.5.2 Kreise

Werden mit dem Element **circle** beschrieben. Angaben für Mittelpunkt und Radius als Attribute:

```
<circle cx="..." cy="..." r="..."/> .
```

2.5.3 Ellipsen

Werden mit dem Element **ellipse** beschrieben. Angabe des Mittelpunkts sowie der zwei Radien:

```
<ellipse cx="..." cy="..." rx="..." ry="..."/> .
```

2.5.4 Linien

Werden mit dem Element **line** beschrieben, mit Attributen für die Koordinaten des Anfangs- und Endpunkts:

```
<line x1="..." y1="..." x2="..." y2="..."/> .
```

2.5.5 Polylinien

Sind nicht geschlossen. Werden mit dem Element **polyline** beschrieben. Als Attribut sind die durch Leerzeichen getrennten Wertepaare für die benötigten Koordinaten anzugeben:

```
<polyline points="x1,y1 x2,y2 x3,y3 ..."/> .
```

2.5.6 Polygone

Werden mit dem Element **polygon** beschrieben. Wie Polylinien, außer, dass Anfangs- und Endpunkt automatisch verbunden werden:

```
<polygon points="x1,y1 x2,y2 x3,y3 ..."/> .
```

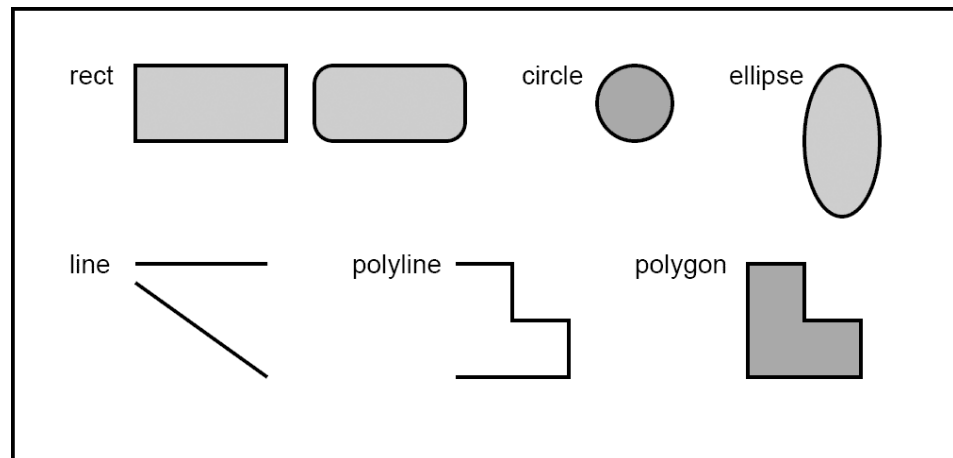


Abbildung 5: Grundformen in SVG – grndform.svg

2.6 Pfade

Pfade können mit dem Element **path** beschrieben werden, z. B. ein Viertel von einem Kreis:

```
<path d="M 0,0 L 50,0 L 50,50 A 50,50 0 0,1 0,0 Z"/> .
```

Das Attribut **d** beinhaltet die Daten für die Beschreibung. Großbuchstaben stehen für absolute, Kleinbuchstaben für relative Koordinaten. Nach **M** (*moveto*) wird der Startpunkt angegeben, **L** (*lineto*) zeichnet eine gerade Linie zum danach angegebenen Punkt. **A** gibt die Art des Bogens an, die folgenden Werte geben seine Radien sowie Richtung an. Das letzte Wertepaar legt den Endpunkt des Bogens fest. **Z** schließt den Pfad.

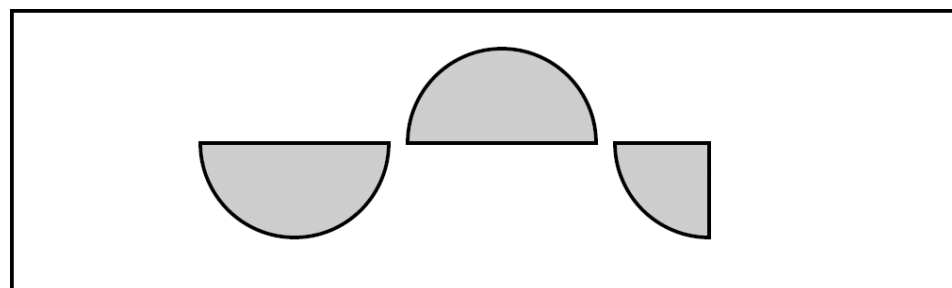


Abbildung 6: Pfade als Kreissegmente – path.svg

2.7 Symbole

Im Element `symbol` kann ein häufig benötigtes Symbol oder Ähnliches aufgenommen werden, ohne dass es auf der Zeichenfläche abgebildet wird. Um es an einer anderen Stelle im Dokument abbilden zu können, ist `symbol` ein ID-Name zuzuteilen:

```
...
<symbol id="symbol1">
  <rect x="0" y="0" width="15" height="15"
    fill="#000"/>
  <rect x="5" y="5" width="5" height="5"
    fill="#FFF"/>
</symbol>
....
```

Im weiteren Verlauf des Dokuments kann das Symbol mit dem Element `use` beliebig oft aufgerufen und gerendert werden:

```
...
<use xlink:href="#symbol1" x="50" y="50"/>
...
<use xlink:href="#symbol1" x="100" y="50"/>
....
```

Für Verweise wird in SVG der *XLink-Standard* verwendet. Dafür ist stets der *XLink-Namensraum* im Wurzelement `svg` anzugeben:

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

2.8 Text

Der Austausch über Texte und Textzeichen stellt ein Grundbedürfnis menschlicher Kommunikation dar. Daher ist auch die Integration von Text in Grafiken notwendig. Eine treffende Aussage dazu macht [WATT02], S.305:

It is essential that you be able to lay out text in SVG. For example, a map may need text to name towns or other features, a technical diagram may need text to label parts of a piece of machinery, and so on. In other settings, a logo may contain text whose positioning may in turn be animated. ...

Diesem Umstand wird Rechnung getragen mit einem Satz an Elementen, der die Integration von Text in Grafiken ermöglicht: Es stehen die Elemente **text**, **tspan**, **tref**, und **textPath** bereit.

Das Element **text** dient als Eltern-Element, das alle Texte sowie alle weiteren Kind-Elemente enthält.

Mit **tspan** können Teile von Texten hervorgehoben bzw. Texte über mehrere Zeilen hinweg dargestellt werden. Dazu wird zunächst der Startpunkt des Textes über **x**- und **y**-Koordinaten definiert. Dann wird der unterschiedliche Startpunkt der Zeilen über eine Verschiebung der **y**-Koordinate mittels **dy** verwirklicht. Im folgenden Beispiel (Listing 3) beträgt die Verschiebung 1.2 mal die gewählte Schriftgröße, durch Angabe der relativen Einheit **em**.

Listing 3: Mehrzeiliger Text mit tspan – tspan.svg

```
...
<text x="30" y="80">
  <tspan x="120" dy="1.2em">Zeile 1</tspan>
  <tspan x="120" dy="1.2em"
    style="font-style:italic; font-weight:bold">
    Zeile 2 hervorgehoben</tspan>
  <tspan x="120" dy="1.2em">Zeile 3</tspan>
  <tspan x="120" dy="1.2em">Zeile 4</tspan>
</text>
...
```

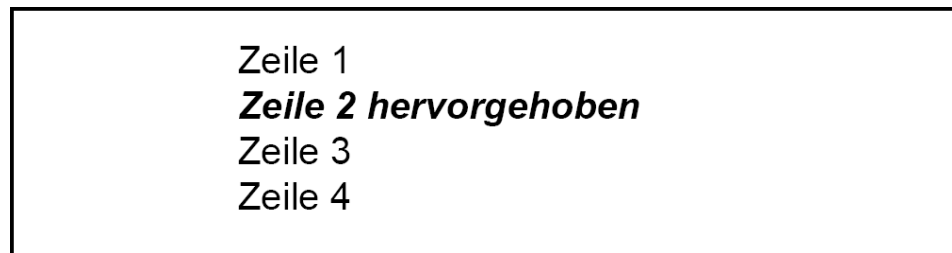


Abbildung 7: Mehrzeiliger Text mit tspan – tspan.svg

Außerdem dient das Element **tref** zum Referenzieren von Textbausteinen; während **textPath** zum Ausrichten von Text entlang eines Pfades eingesetzt werden kann.

2.9 Textlinks

Textlinks werden unter Verwendung des *XLink-Namensraums* an der gewünschten Stelle im Dokument platziert. Im Element **a** mit dem Verweisziel wird der entsprechende Linktext eingefügt:

```
...  
<a xlink:href="...">  
  <text x="..." y="...">Linktext</text>  
</a>  
....
```

2.10 Formatierung

2.10.1 Eigenschaften

Zum Formatieren (*styling*) von Elementen in SVG stehen CSS-Eigenschaften sowie SVG-spezifische Eigenschaften zur Verfügung.¹⁹ Einige wichtige davon sind die folgenden:

¹⁹ siehe auch [SVG11REC] Abschnitt 6.1 SVG's styling properties

Eigenschaft	Beschreibung	Mögliche Werte
stroke	Strichfarbe für Grundformen, Text	Werte in Hexadezimal-Notation, z. B.: #0000FF oder #00F (Kurzform); Werte als Farbname, z. B.: blue ; Werte in RGB-Notation, z. B.: rgb(0,0,255) rgb(0%,0%,100%)
stroke-width	Strichbreite für Grundformen, Text	Numerische Werte in einer Längeneinheit
fill	Füllfarbe für Grundformen, Text	siehe stroke
font-family	Schriftart	Name der Schriftart und Namen alternativer Schriftarten, z. B.: Arial, Helvetica, sans-serif
font-size	Schriftgröße	Numerische Werte in einer Längeneinheit, ...
font-weight	Schriftstärke	normal, bold, bolder, lighter ...

Tabelle 3: Wichtige Eigenschaften zur Formatierung von SVG

Es gibt verschiedene Möglichkeiten, Elementen die Eigenschaften zuzuweisen.

2.10.2 Zuweisung im Tag

Eine Möglichkeit ist die *Zuweisung als Attribut* mit entsprechendem Wert: Alle Eigenschaften existieren in SVG auch als Attribut, so dass die Formatierung als Attribut direkt in das Tag geschrieben werden kann, z. B. für ein Rechteck:

```
<rect ... stroke="#000" stroke-width="2"
fill="#CCC"/> .
```

Eine andere Möglichkeit ist die *Zuweisung über das style-Atribut*:

```
<rect ... style="stroke: #000; stroke-width: 2;
fill: #CCC"/> .
```

Diese Möglichkeiten sind jedoch nur gebräuchlich, wenn einige wenige Elemente formatiert werden sollen, da die Zuweisung für jedes Element einzeln erfolgt. Für umfangreichere SVG-Dokumente mit einer Vielzahl an Elementen ist diese Möglichkeit sehr aufwändig. Für solche Dokumente empfiehlt sich daher eine andere Zuweisung.

2.10.3 Zuweisung mit CSS-Notation

Für umfangreichere SVG-Dokumente mit einer Vielzahl an Elementen empfiehlt es sich, die Eigenschaften an einer zentralen Stelle abzulegen und die Formatierungen von dort aus vorzunehmen. Dafür existiert eine spezielle Notation, bei der den Elementen die Eigenschaften (*properties*) und Werte (*values*) durch einen *Selektor (selector)* zugewiesen werden:

Selektor

```
{Eigenschaft1: Wert; Eigenschaft2: Wert; ...} .
```

Als Selektor sind u. a. möglich:

- Element-Namen, z. B. **rect**,
- *Klassen-Selektoren*, z. B. **.schwarzgrau**,
- Element-Namen in Verbindung mit Klassen-Selektoren, z. B. **rect.schwarzgrau**,
- *ID-Selektoren*, z. B. **#schwarz1**,
- Element-Namen in Verbindung mit ID-Selektoren, z. B. **circle#schwarz2** .

Wird ein *Element-Name* als Selektor verwendet, so wirken sich die Eigenschaften auf alle entsprechenden Elemente in einem Dokument aus. Bei der Verwendung von

rect

```
{stroke: #000; stroke-width: 2; fill: #CCC}
```

würden z. B. alle Rechtecke in einem SVG-Dokument mit einem zwei Pixel breiten, schwarzen Rand sowie grau gefüllt formatiert werden.

Eine weitere Möglichkeit ist die Verwendung von *Klassen-Selektoren*. Damit können auch unterschiedliche Elemente mit einheitlicher Formatierung versehen werden. So würde

.schwarzgrau

```
{stroke: #000; stroke-width: 2; fill: #CCC}
```

Rechtecke sowie auch Kreise einheitlich formatieren, wenn sie im SVG-Dokument mit dem Klassen-Namen **schwarzgrau** bezeichnet sind:

...

```
<rect class="schwarzgrau" ... />
```

```
<circle class="schwarzgrau" ... />
```

....

Dagegen würde die *Kombination eines Element-Namens mit einem Klassen-Selektor* wie

rect.schwarzgrau

```
{stroke: #000; stroke-width: 2; fill: #CCC}
```

nur Rechtecke ansprechen, die zudem auch den entsprechenden Klassen-Namen tragen:

```
<rect class="schwarzgrau" ... /> .
```

Als Selektor möglich sind auch *ID-Selektoren*, allein stehend oder in Verbindung mit einem Element-Namen, wobei eine ID bzw. ihr Name in einem Dokument einzigartig bleiben muss. So würde

#schwarz1

```
{stroke: #000; stroke-width: 2; fill: #000}
```

das Element formatieren, das die ID **schwarz1** besitzt:

```
<rect id="schwarz1" ... /> .
```

Dagegen formatiert

```
circle#schwarz2
  {stroke: #000; stroke-width: 2; fill: #000}
```

exklusiv einen Kreis, der die ID **schwarz2** besitzt:

```
<circle id="schwarz2" ... /> .
```

Liegen die Formatierungen in der beschriebenen Notation vor, so können sie an einer zentralen Stelle abgelegt werden. Dafür bieten sich der Definitionsbereich eines SVG-Dokuments oder eine externe CSS-Datei an.

2.10.4 Ablage im Definitionsbereich

Die Formatierungen stehen hierbei in einem SVG-Dokument innerhalb der Elemente **defs** und **style** in einem *CDATA-Abschnitt*. Die Formatierungen wirken sich dabei (nur) auf das aktuelle SVG-Dokument aus:

```
...
<defs>
  <style type="text/css">
    <![CDATA[
      Selektor
      {
        Eigenschaft1: Wert;
        Eigenschaft2: Wert;
        ...
      }
    ]]>
  </style>
</defs>
....
```

Der CDATA-Abschnitt (CDATA = *character data*) stellt eine sichere Methode dar, um in einem XML-Dokument Zeichen zu verwenden, die nicht als XML-Syntax interpretiert werden sollen, wie es hier bei der CSS-Syntax der Fall ist. Er ist zwar nicht in jedem Fall unbedingt notwendig, da nur bestimmte Zeichen kritisch sind (siehe Entity-Referenzen, Tabelle 1, S.14). Es ist aber dennoch ratsam, ihn zu verwenden, um von vornherein Fehler zu vermeiden.

2.10.5 Ablage in externer CSS-Datei

Sollen in mehreren SVG-Dokumenten die gleichen Bestandteile einheitlich formatiert werden, so empfiehlt es sich, die Formatierungen in einer externen CSS-Datei abzulegen. Die CSS-Datei besteht dabei ausschließlich aus Formatierungen in CSS-Notation und ggf. Kommentaren:

```
Selektor1
  {Eigenschaft1: Wert; Eigenschaft2: Wert; ...}
  /* CSS-Kommentar */
Selektor2
  {Eigenschaft1: Wert; Eigenschaft2: Wert; ...}
....
```

Die CSS-Datei muss nun den SVG-Dokumenten zugeordnet werden, auf die sie sich auswirken soll. Dazu wird in den entsprechenden SVG-Dokumenten vor dem Wurzelement **svg** eine *Verarbeitungs-Anweisung (Processing Instruction)* eingefügt, die auf die externe CSS-Datei verweist:

```
...
<?xml-stylesheet type="text/css"
href="extern.css"?>
....
```

2.11 Externe Bilder

Mit dem Element **image** können externe (Raster-)Bilder eingebunden werden. Die Attribute **x** und **y** dienen zum Positionieren,

width und **height** bestimmen die Größe. Die Quelle wird nach **xlink:href** angegeben:

```
<image x="..." y="..." width="..." height="..."  
xlink:href="bild.jpg" /> .
```

2.12 Effekte

SVG unterstützt eine Reihe von Features, die Inhalten ein attraktives Erscheinungsbild ermöglichen:²⁰

- lineare und radiale Verläufe (Gradienten),
- Muster für das Belegen von Flächen,
- Masken,
- Filter.

Lineare und radiale Verläufe werden in SVG mit den Elementen **linearGradient** und **radialGradient** verwirklicht. Dabei notwendig ist das Kind-Element **stop** mit dem Attribut **offset**, damit die einzelnen Teilabschnitte der Verläufe beschrieben werden können. Zudem ermöglichen die Eigenschaften **stop-color** und **stop-opacity** die Angabe der gewünschten Farben und der gewünschten Transparenz.

Um Flächen mit einem Muster zu belegen, kann in SVG das Element **pattern** verwendet werden. Diesem werden eine Höhe und Breite zugewiesen, was den Abstand der einzelnen Grundbestandteile des Musters festlegt. Als Grundbestandteile kommen Grundformen und Pfade infrage. Sie können innerhalb von **pattern** abgelegt, aber auch von einer anderen Stelle her referenziert werden.

In SVG ist zudem auch das Verwenden von Masken möglich. Hierbei sind *Beschneidungs-Masken* und *Alpha-Masken* möglich. Beschneidungs-Masken dienen zum Ausschneiden von gewünschten Formen aus Grafiken oder Bildern. Für sie wird das Element **clipPath** verwendet. Für Alpha-Masken kann das Element **mask** eingesetzt werden.

²⁰ siehe auch [SVG11REC] Abschnitte 13–15

Auch Filter werden von SVG unterstützt. Sie werden durch das gleichnamige Element **filter** beschrieben. Innerhalb dieses Elements befinden sich weitere Elemente, die einzelne Filter-Effekte ermöglichen. Sie sind gekennzeichnet durch die Vorsilbe **fe**. Beispielsweise bewirkt das Element **feGaussianBlur** eine Weichzeichnung. Mehrere solcher **fe**-Elemente lassen sich so zu einem Filter zusammensetzen, der bei Bedarf auf Elemente oder Element-Gruppen angewendet werden kann.

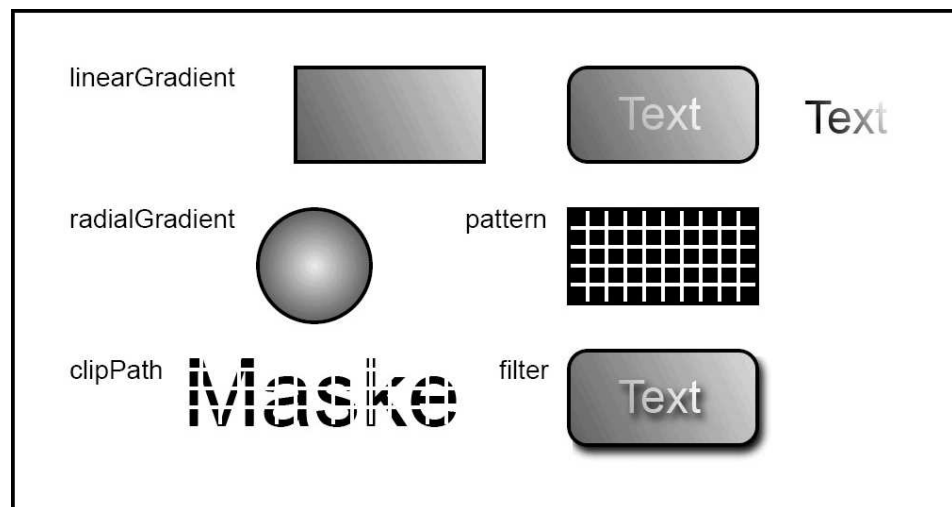


Abbildung 8: Effekte in SVG – effects.svg

2.13 Gruppen und Transformationen

Insbesondere bei komplexeren SVG-Grafiken ist die Möglichkeit, mehrere Elemente in einer Gruppe zusammenzufassen, hilfreich. SVG stellt dafür das Element **g** zur Verfügung. Alle Elemente innerhalb von **<g>** und **</g>** gehören einer Gruppe an.

Gruppen sind auch hinsichtlich von *Transformationen* interessant. Damit lassen sich mehrere Elemente mit einer Operation umformen. SVG stellt vier vordefinierte Transformationen für Elemente zur Verfügung (siehe auch Abbildung 9):

- Verschiebung (*Translation*),
- Vergrößerung und Verkleinerung (*Skalierung*),
- Drehung (*Rotation*),

- Neigen der Achsen.

Für Transformationen ist das Attribut **transform** vorgesehen, das je nach Art der Transformation unterschiedliche Attributwerte besitzt:

- **translate(... , ...)**,
mit Angabe der Werte für die Verschiebung in x- und y-Richtung,
- **scale(... , ...)**,
mit Angabe der Skalierungs-Faktoren für die **x**- und **y**-Werte,
- **rotate(... , ... , ...)**,
mit Angabe des Drehwinkels im Uhrzeigersinn und der Koordinaten für den Drehpunkt,
- **skewX(...)**,
Verzerren aller **x**-Werte, d. h. Neigen der Y-Achse; Werte für den Neigungs-Winkel der Y-Achse (bei positiven Werten Neigung gegen den Uhrzeigersinn) bzw.
skewY(...),
Verzerren aller **y**-Werte, d. h. Neigen der X-Achse; Werte für den Neigungs-Winkel der X-Achse (bei positiven Werten Neigung im Uhrzeigersinn).

Zudem existiert mit **matrix()** ein Attributwert, mit dem die zuvor beschriebenen Transformationen und weitere möglich sind.²¹

21 weitere Informationen siehe [SVG11REC] Abschnitt 7.6 The transform attribute

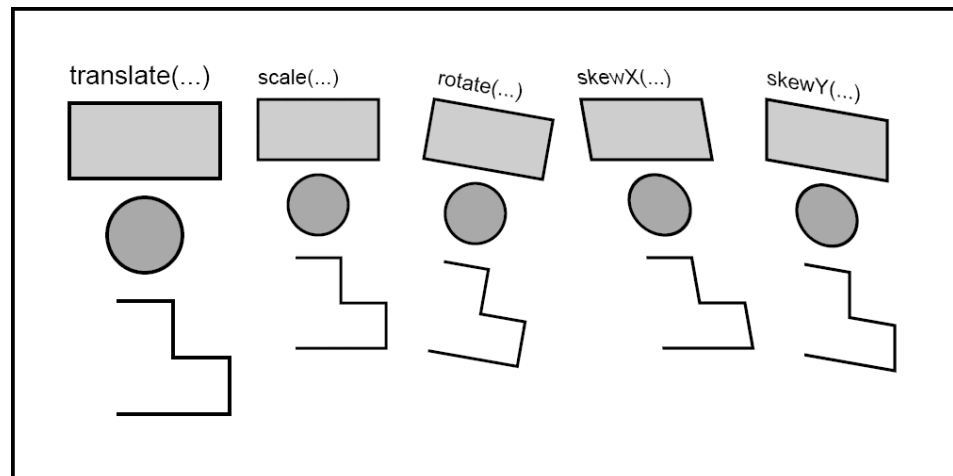


Abbildung 9: Transformationen in SVG – transfo.svg

2.14 Animationen

SVG stellt auch Elemente zur Verfügung, die Animationen erlauben. Diese stammen zum größten Teil aus der *Synchronized Multimedia Integration Language (SMIL)*, sprich: smeil), ebenfalls ein W3C-Standard.²² Die Elemente erfüllen nach [MEIN03], S.72, folgende Aufgaben:

- **animate:**
Animation von skalaren XML-Attributwerten und von CSS-Eigenschaften in Form der Zuweisung verschiedener Werte über eine bestimmte Zeitspanne,
- **set:**
einfache Schaltvorgänge wie das Ein- und Ausblenden von Objekten oder sofortiges Zuweisen von Farbwerten,
- **animateMotion:**
Steuerung von Bewegungsabläufen, insbesondere entlang von Pfaden,
- **animateColor:**
zeitliche Veränderung von Farbwerten,

²² siehe [SMIL20REC]

- **animateTransform:**

Beeinflussung von Transformationen von Objekten auf einer Zeitskala.

Die Grundidee ist, auf XML-Attribute oder CSS-Eigenschaften wie z. B. die Höhe **height** oder die Opazität **opacity** eines Rechtecks zuzugreifen und sie an bestimmten Zeitpunkten zu verändern. Dazu dienen die Attribute **attributeName**, das den Namen des Attributs oder der Eigenschaft zugewiesen bekommt und **attributeType**, das entsprechend **XML** oder **CSS** als Wert erhält.

Der Ablauf kann dabei genau gesteuert werden. Möglich ist das mit den Attributen **begin**, **dur** und **end**, die die Startzeit, die Zeitspanne und das Ende der Animation festlegen. Mit Angabe der Attribute **from** und **to** werden der Start- und Endwert des zu animierenden Attributs oder der zu animierenden Eigenschaft angegeben. Wenn diese Werte zur Beschreibung der Animation nicht ausreichen, ist auch die Angabe von mehreren Zeitpunkten mit Hilfe des Attributs **keyTimes** und den entsprechenden Werten zu den Zeitpunkten mit dem Attribut **values** möglich. Hinzu kommt das Attribut **fill** (bezeichnet hier keine CSS-Eigenschaft), mit dem der Zustand beim Erreichen des Endwerts beschrieben wird. Der Wert **freeze** erhält den Endwert des zu animierenden Attributs oder der zu animierenden Eigenschaft, während der Wert **remove** wieder den Startwert herstellt.

Die beschriebenen Elemente und Attribute stellen die grundlegenden Möglichkeiten dar. Weitere Details sind unter [SVG11REC] Abschnitt 19 Animation zu finden.

Siehe auch `anim.svg`.

2.15 Scripting

SVG-Dokumente unterstützen Dynamik und Interaktivität, d. h. sie können so gestaltet werden, dass sie auf Ereignisse wie Mausklicks oder Mausbewegungen reagieren. Dies kann mit den Script-Sprachen *JavaScript* bzw. *ECMAScript* verwirklicht werden.

Dabei wird über das SVG-eigene *Document Object Model (DOM)*, das die Bestandteile eines SVG-Dokuments in einer Baumstruktur abbildet, auf die Bestandteile zugegriffen, sodass diese beeinflusst werden können. Eine Verzweigung in der Baumstruktur wird als Knoten (*node*) bezeichnet. Entsprechend den Bestandteilen eines SVG-Dokuments gibt es verschiedene Knoten, über die auf Elemente, Attribute, Formatierung und Inhalte zugegriffen werden kann (siehe Abbildung 10).

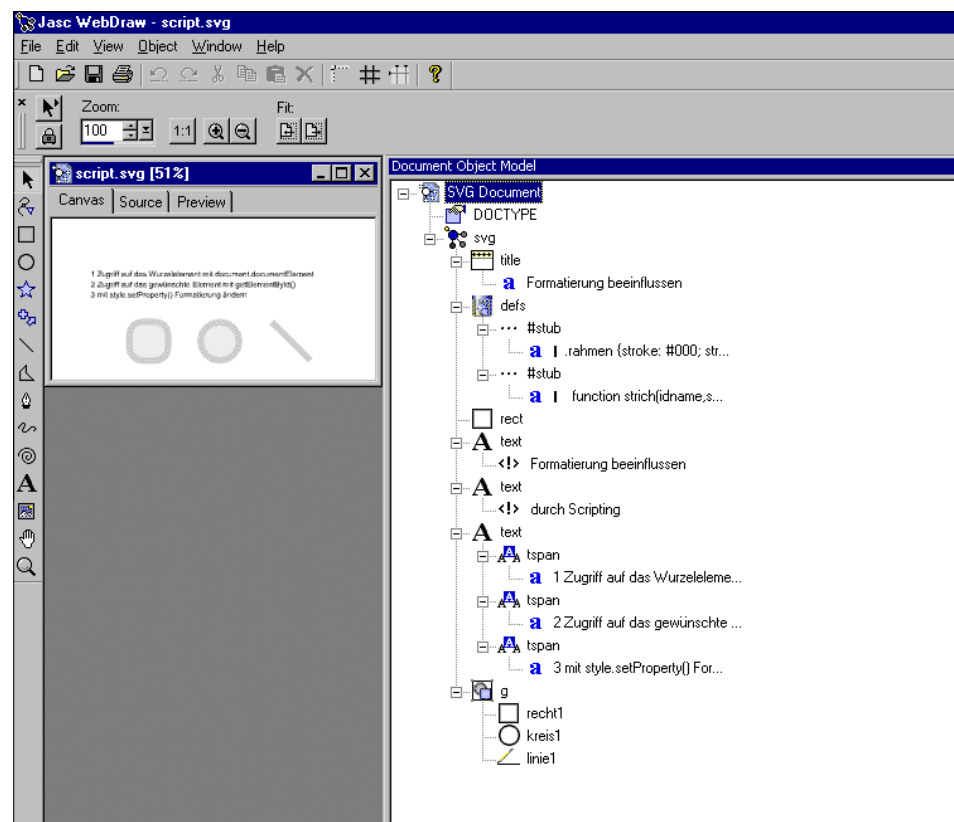


Abbildung 10: DOM-Baum in Jasc WebDraw

Zum Beeinflussen der Bestandteile sind ein *Script* und ein Ereignis (*event*) notwendig, das die Ausführung des Scripts startet. Hinzu kommt ein *Event-Handler*, über den das Event erfasst wird. Er ist das Glied, das die Verbindung zum Nutzer herstellt und die In-

teraktionskette *Nutzer – Event – Event-Handler – Script – DOM-Beeinflussung* schließt.²³

2.15.1 Ablage eines Scripts

Ein Script wird in einem SVG-Dokument im **defs**-Bereich im Element **script** in einem CDATA-Abschnitt abgelegt, wobei dem **script**-Tag noch der Typ der verwendeten Script-Sprache (JavaScript oder ECMAScript) hinzugefügt wird:

```
...
<defs>
  <script type="text/javascript">
    <![CDATA[
      /* Script */
    ]]>
  </script>
</defs>
... .
```

Möglich ist auch, das Script in einer externen Datei abzulegen und diese dann in ein SVG-Dokument einzubinden. Dazu wird im SVG-Dokument notiert:

```
<script xlink:href="extern.js"
type="text/javascript"/> .
```

2.15.2 Beispiel für ein Script

Scripts sind kleine Programme, die über die Baumstruktur eines Dokuments auf seine Bestandteile zugreifen sowie diese beeinflussen können. Anhand des folgenden Scripts soll erläutert werden, wie ein Script zustande kommt und wie es wirkt:

²³ eine Übersicht zu Events und Event-Handlern findet sich in [ADAM02], S.278f. oder in [SVG11REC] Abschnitt 16.2 Complete list of supported events

Listing 4: Script zur Änderung der Strichfarbe

```
1 ...
2 function strich(idname, strichfarbe)
3 {
4   var svgdoc, element;
5   svgdoc=document.documentElement;
6   element=svgdoc.getElementById(idname);
7   element.style.setProperty("stroke", strichfarbe);
8 }
9 ...
```

Listing 4 zeigt ein Script, das aus einer Funktion namens **strich** besteht (Zeile 2). Sie soll die Aufgabe haben, die Eigenschaft **stroke** (Strichfarbe) von Elementen zu ändern. Dazu muss gezielt auf den Dokument-Baum zugegriffen werden, was sich in drei Teilschritte aufgliedern lässt:

- Zugriff auf das Wurzelement des Dokument-Baums,
- Zugriff auf das gewünschte Element mit Hilfe seines ID-Namens,
- Zugriff auf die Formatierungs-Eigenschaften des Elements und Setzen eines neuen Wertes für die Eigenschaft **stroke**.

Zunächst werden in Zeile 4 für die ersten zwei Schritte die *Variablen* **svgdoc** und **element** definiert. In Zeile 5 wird der Variablen **svgdoc** mit **document.documentElement** der Zugriff auf das Wurzelement zuteilt. In Zeile 6 erhält die Variable **element** mit **svgdoc.getElementById(idname)** Zugriff auf das Wurzelement und weiter Zugriff auf das durch seinen ID-Namen gekennzeichnete Element. Damit kann das gewünschte Element im Dokument-Baum angesprochen werden. In Zeile 7 schließlich werden zu der Variable **element** das Objekt **style** und die Methode **setProperty()** hinzugefügt. Wobei **style** auf die Formatierungs-Eigenschaften des Elements zugreift und mit **setProperty("stroke", strichfarbe)** die Eigenschaft **stroke** ausgewählt sowie eine Strichfarbe gesetzt wird.

Die *Parameter-Namen* in Klammern von **getElementById(idname)** bzw. **setProperty("stroke", strichfarbe)**

werden nun in Zeile 2 in Klammern hinter dem Funktions-Namen notiert:

```
...
function strich(idname, strichfarbe)
....
```

Somit kann die Funktion (bzw. das Script) im SVG-Dokument auf alle Elemente angewendet werden, die einen ID-Namen besitzen, wie z. B. ein Rechteck:

```
...
<rect id="recht1" ... style="stroke:#CCC;
fill:#EEE; stroke-width:10;"
  onmouseover="strich('recht1', '#000')"
  onmouseout="strich('recht1', '#CCC')"/>
....
```

Im SVG-Dokument werden nun mit **onmouseover** und **onmouseout** auch zwei der erwähnten Event-Handler eingesetzt, die zum Erfassen des Events dienen, welches das Script bzw. die Funktion startet. Der Event-Handler **onmouseover** nimmt ein Event wahr, wenn sich der Mauszeiger über einem Objekt befindet, während **onmouseout** ein Event wahrnimmt, wenn der Mauszeiger das Objekt wieder verlässt. Beim zuvor aufgeführten Stück Code würde die Strichfarbe beim Überfahren des Rechtecks auf schwarz gesetzt (siehe Abbildung 11) und beim Verlassen des Rechtecks wieder auf grau zurückgesetzt werden.

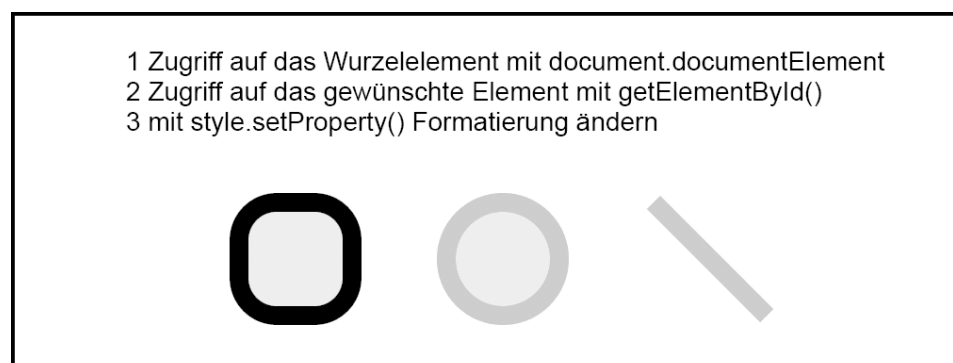


Abbildung 11: Formatierung beeinflussen durch Scripting – script.svg

2.15.3 Eigenschaft `pointer-events`

Für die genaue Beschreibung dafür, wo Objekte Event-sensitiv sein sollen, steht die Eigenschaft `pointer-events` zur Verfügung. Sie kann u. a. folgende Werte besitzen:²⁴

- **visiblePainted:**
das Objekt ist oberhalb seiner sichtbaren Formatierung Event-sensitiv,
- **stroke:**
das Objekt ist oberhalb seiner Umrisslinie Event-sensitiv,
- **fill:**
das Objekt ist oberhalb seiner Füllung Event-sensitiv,
- **none:**
das Objekt ist überhaupt nicht Event-sensitiv,
- **visible:**
das Objekt ist Event-sensitiv, wenn seine Eigenschaft `visibility` den Wert `visible` besitzt,
- **all:**
das Objekt ist über seiner gesamten Oberfläche Event-sensitiv, unabhängig von den Eigenschaften `fill`, `stroke`, `visibility`.

Siehe auch `scriptng.svg`.

2.16 Einbindung in HTML

SVG-Dokumente werden mit dem Element `object` in HTML eingebunden, wobei mit `type` der Inhaltstyp angegeben wird. Angegeben werden sollte auch ein Verweis zum Downloaden des benötigten SVG-Viewers, für den Fall, dass dieser nicht installiert ist:

```
...  
<object data="svgdokument.svg" width="..."  
height="..." type="image/svg+xml">
```

²⁴ weitere Werte siehe [SVG11REC] Abschnitt 16.6 The 'pointer-events' property

```
<!-- Verweis zum Download des benoetigten SVG-  
Viewers -->  
</object>  
....
```

3 Neuerungen in SVG 1.2

3.1 Einleitung

Gut drei Jahre sind nach der Verabschiedung von SVG 1.0 vergangen. Die etwa ein anderthalbes Jahr später verabschiedete Version 1.1 brachte auch nichts grundlegend Neues. Nach außen hin mutet dies wie Stagnation an, was tatsächlich aber nicht der Fall ist. Angesichts dieses Zeitraums ist auch die Frage, ob SVG überhaupt auf Akzeptanz trifft, nicht unberechtigt:

Nach [ADAM04], S.34, fehle den beiden Spezifikationen ein Ziel, eine klare Ausrichtung sowie ein wirklich zwingender Anwendungsfall. Die Spezifikation zu SVG 1.0 sei »gewiss noch etwas unausgereift«, habe aber ein großes Potenzial. Dennoch sei es bisher bei diesem Potenzial geblieben, »da noch viele grundlegende Dinge für die sinnvolle Nutzung von SVG fehlten«.

Diesen Aussagen ist durchaus zuzustimmen, zeigen sie doch auch den Weg, den das W3C mit der SVG-Entwicklung beschreitet. Am aussagekräftigsten sind in diesem Zusammenhang die letzten beiden Working Drafts zu SVG 1.2. Sie lassen bei genauerer Betrachtung erkennen, dass das W3C versucht, SVG in Version 1.2 um die vermisste Funktionalität zu ergänzen und damit die sinnvolle Nutzung übergreifend zu gewährleisten. Zudem zeigen die Working Drafts auch, dass SVG mit Version 1.2 darüber hinaus noch an Funktionalität hinzugewinnt.

Mit der Aufteilung von SVG in verschiedene Bereiche sollte nun auch die gewünschte gezieltere Ausrichtung auf einzelne Anwendungsgebiete möglich sein. Die auf der gleichen Grundlage basierenden einzelnen SVG-Module können so unabhängig voneinander verwendet werden (nach [ADAM04], S.34):

- *SVG 1.2*: Das Standardformat für alle Arten von Web- und Desktop-Publishing,
- *SVG Tiny 1.2* [*sic*; korrekt: *SVG Basic 1.2*²⁵]: Untermenge von SVG, die alle notwendigen Elemente für das Mobile Publishing, z. B. für Notebooks, enthält,

25 vgl. <http://www.xml.com/pub/a/2004/06/16/mobilesvg.html> (16.06.2004)

- *SVG Basic 1.2* [*sic; korrekt: SVG Tiny 1.2*]: Kleinste Version von SVG für das Publishing auf Kleinste geräten wie z. B. Mobiltelefonen und PDAs,
- *SVG Print*. Diese Spezifikation ist im Gegensatz zu den vorherigen drei noch eher eine Vision als eine konkrete Spezifikation; sie soll es zukünftig ermöglichen, SVG nativ ausdrucken zu können.

Die Aufteilung in Module erscheint sinnvoll, da so auf die Eigenheiten verschiedener Medien differenziert eingegangen werden kann. So kann sich SVG noch vielfältiger entwickeln, als es bisher schon ist und es kann sich außerdem neue Anwendungsgebiete erschließen.

In den folgenden Ausführungen sollen die Möglichkeiten, die in SVG 1.2 hinzukommen, erläutert und anhand von Beispielen veranschaulicht werden. Die Veranschaulichung kann hierbei aber nur bei den wenigsten Beispielen auch mit einer grafischen Ausgabe vervollständigt werden, da die dargestellten Features noch nicht in SVG-Programmen implementiert sind. Ausnahmen hiervon sind Fließtext und Rendering Custom Content, welche bereits im *ASV 6.0 preview 1* verfügbar sind. Bei den meisten Beispielen muss sich die Veranschaulichung aber auf erläuternde Texte und Beispiel-Codes beschränken.

3.2 Bereich Text

3.2.1 Fließtext mit Bildern

Textfluss und automatischer Zeilenumbruch sind in SVG bisher nicht möglich gewesen. Als Ersatz konnte das Element `tspan` verwendet werden. Hiermit ließen sich wenigstens mehrzeilige Texte verwirklichen, wobei Zeile für Zeile vorgegangen werden musste (siehe [mehrzeiliger Text](#), Listing 3, S.22). Bei längeren Texten wurde dies aber enorm aufwändig.

Mit Version 1.2 ist nun Fließtext bzw. das automatische Umbrechen von Text und Bildern möglich. Dafür werden eine ganze Reihe von Elementen eingeführt, die Textfluss ermöglichen bzw. da-

mit verbundene Aufgaben erfüllen. Gemeinsam ist diesen Elementen **f**low als ein Bestandteil ihres Element-Namens.

3.2.2 Fließtext-Elemente und ihre Aufgaben

- **flowRoot:**
definiert einen Block für Fließtext mit automatischem Zeilenumbruch,
- **flowRegion:**
definiert den Fließbereich; enthält grafische Grundformen, Pfade, welche den Fließbereich für Texte/Bilder vorgeben oder auch **flowRegionExclude**,
- **flowRegionExclude:**
enthält Formen, die vom Textfluss ausgenommen sind und die vom Text umflossen werden,
- **flowDiv:**
bildet einen Block aus Text/Bildern,
- **flowPara:**
bildet einen (logischen) Absatz mit Text/Bildern,
- **flowSpan:**
beschreibt einen untergeordneten, inneren Block von Text/Bildern,
- **flowRegionBreak:**
beendet das Fließen von Texten innerhalb eines Fließbereiches an der Stelle, wo es eingesetzt ist,
- **flowLine:**
erzwingt einen Zeilenumbruch,
- **flowTref:**
dient zum Einfügen von Text, der einem referenzierten Element entnommen wird,

- **flowImage:**
definiert einen Abschnitt für Bilder; ermöglicht das gemeinsame Fließen von Bildern und Text in einem Fließbereich,
- **flowRef:**
referenziert ein **flowRegion**-Element.

3.2.3 Fließtext-Elemente im Zusammenhang

Das Element **flowRoot** fungiert als Eltern-Element, das alle weiteren Elemente umschließt. Es muss **flowRegion** enthalten.

Die Kind-Elemente von **flowRegion** erzeugen eine Abfolge an Formen, in die der Text eingetragen wird. Ist eine Form mit Text ausgefüllt, fließt der Text in der nächsten Form weiter. Die Kind-Elemente können auf übliche Weise transformiert werden, wobei der Text sich aber am Koordinatensystem von **flowRoot** orientiert.

Die in **flowRegionExclude** enthaltenen Formen wirken sich abhängig von dessen Stellung unterschiedlich aus: Ist es Kind von **flowRoot**, gelten die Formen für alle **flowRegion**-Elemente. Ist es Kind eines **flowRegion**-Elementes, gelten die Formen nur für dieses eine **flowRegion**-Element.

Durch **flowDiv** können einzelne Blöcke innerhalb von **flowRoot** variabel gestaltet, angeordnet und in ein Layout eingesetzt werden. Kinder von **flowDiv** werden als Einheit gerendert und mit Abstand vor und nach den Geschwistern platziert.

Durch **flowPara** (Absätze) lässt sich **flowDiv** weiter unterteilen.

Mit **flowSpan** können Bestandteile der Absätze spezifisch gestaltet werden.

Mit **flowRegionBreak**, das innerhalb von **flowDiv** und **flowPara** angewendet werden kann, wird der aktuelle Textfluss innerhalb eines Fließbereiches (**flowRegion**) beendet. Texte nach **flowRegionBreak** beginnen in einem neuen Fließbereich. Ist kein neuer Fließbereich vorhanden, werden nur die Inhalte vor **flowRegionBreak** dargestellt.

Das Element **flowLine** erzwingt einen Zeilenumbruch in einem Textfluss. Wird es angewendet, ohne dass sich vor ihm Zeichen befinden, bleibt es wirkungslos und es wird keine neue Zeile eröffnet.

Analog zum **tref**-Element aus SVG 1.0 kann mit **flowTref** ein referenzierter Text eines anderen Elementes eingefügt werden.

Das einen Abschnitt für Bilder beschreibende **flowImage** eröffnet gleichzeitig einen neuen Bildausschnitt (*viewport*). Der neue Bildausschnitt wird jedoch nur eröffnet, wenn **flowImage** eine absolute oder relative Größenangabe erhält.

Das Element **flowRef** trägt die referenzierte Geometrie eines **flowRegion**-Elements mit dem Fließtext in das aktuelle Koordinatensystem ein.

3.2.4 Fließtext-Beispiel

In Listing 5 auf S.45 sind die Möglichkeiten bei Fließtext soweit angewendet, wie es die Implementierung von SVG 1.2 im *ASV 6.0 preview 1* zulässt. Zum Einsatz kommen die Elemente **flowRegion**, **flowDiv**, **flowPara** und anstatt von **flowRoot** das Element **flow** sowie das Element **region**. Diese Elemente werden bereits unterstützt. Da jedoch noch keine entsprechende DTD existiert, wurde die vorhandene DTD wie in *flowtext.svg* in [MEIN02] zu Testzwecken erweitert. Abbildung 12 auf S.46 zeigt die entsprechende Ausgabe des SVG-Viewers.

Listing 5: Fließtext – flow.svg

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd"
[ <!ATTLIST svg xmlns:xlink CDATA #FIXED
"http://www.w3.org/1999/xlink">
  <!ENTITY % svgExt "|flow">
  <!ELEMENT flow (flowRegion,flowDiv)>
  <!ELEMENT flowRegion (region)>
  <!ELEMENT flowDiv (flowPara*)>
  <!ELEMENT flowPara (#PCDATA)>
  <!ELEMENT region EMPTY>
  <!ATTLIST region xlink:href CDATA #REQUIRED ]>

<svg xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Fließtext</title>
<defs>
  <style type="text/css"><![CDATA[
    .bild {stroke:#CCC; stroke-width:2; fill:none}
    .visual {fill:#DDD}
    flow {fill:#000; font-size:20; font-family:Arial,
Helvetica, sans-serif}
  ]]>
  </style>
  <!-- nicht sichtbarer Fließbereich als Referenz -->
  <polygon id="fliess" points="30,190, 200,190, 200,100,
470,100, 470,300, 30,300"/>
</defs>

<rect class="bild" x="30" y="100" width="155" height="75"/>
<!-- Visualisierung des nicht sichtbaren Fließbereiches -->
<polygon class="visual" points="30,190, 200,190, 200,100,
470,100, 470,300, 30,300"/>

<flow>
  <flowRegion><region xlink:href="#fliess"/></flowRegion>
  <flowDiv>
    <flowPara>Der Text dieses Absatzes fließt innerhalb eines
    vorgegebenen Bereiches. In diesem Beispiel handelt es
    sich dabei um ein grau gefülltes Polygon.</flowPara>
    <flowPara>&#160;</flowPara>
    <flowPara>Der Text dieses Absatzes fließt auch in diesem
    Polygon.</flowPara>
  </flowDiv>
</flow>
</svg>

```

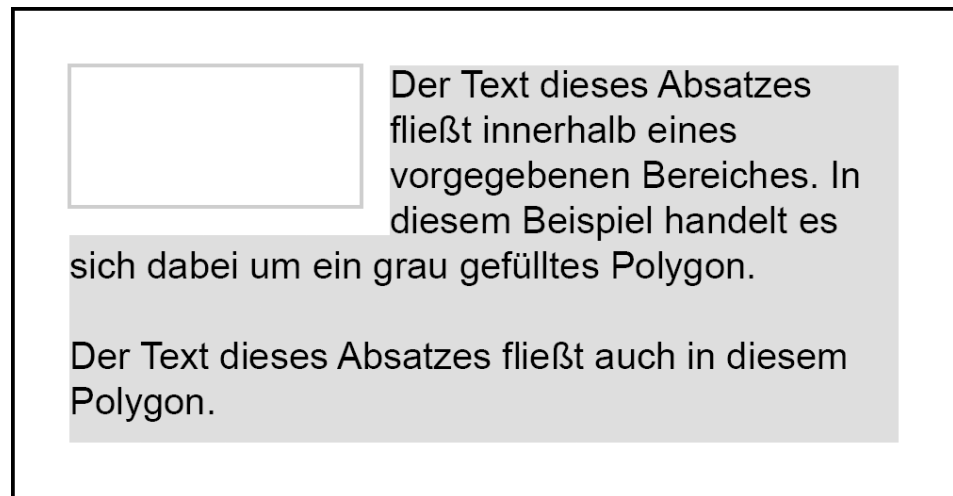


Abbildung 12: Fließtext – flow.svg

3.2.5 Fließtext-Ausrichtung

Für Ausrichtung in Laufrichtung der Schriftzeichen steht die Eigenschaft **text-align** aus CSS3 zur Verfügung. Die Werte **left** und **right** wurden entfernt, da sie im internationalen Kontext nicht sinnvoll sind. Stattdessen treten nun die Werte **start** und **end** an ihrer Stelle. Sie sind abhängig vom verwendeten Schriftsystem. Für Deutsch würde **start** linksbündig und **end** rechtsbündig bedeuten während in einer arabischen Sprache **start** rechtsbündig und **end** linksbündig entspricht. (siehe Tabelle 4)

Für Ausrichtung in Richtung des Zeilenfortschritts wird die Eigenschaft **progression-align** eingeführt (siehe Tabelle 4). Die zugehörigen Werte sind ebenfalls vom Schriftsystem abhängig. Für Deutsch würde **before** oben, **after** unten und **center** vertikal mittig bedeuten.

Eigenschaft	Mögliche Werte	Für die Elemente
text-align	start end center justify	flowRoot flowPara flowDiv

Eigenschaft	Mögliche Werte	Für die Elemente
<code>progression-align</code>	<code>before</code> <code>after</code> <code>center</code>	<code>flowRoot</code> <code>flowPara</code> <code>flowDiv</code>

Tabelle 4: Ausrichtung von Fließtext

3.2.6 Overflow

Für den Fall des »Überfüllens« eines `flowRegion`-Elements, d. h. wenn die Menge an Text/Bildern nicht mehr in den definierten Fließbereich hineinpasst, sind zwei Events vorgesehen: ein `overflow`-Event bzw. ein `underflow`-Event. Das zugehörige Interface ist mit `SVGOverflowEvent` bezeichnet.

3.2.7 Editierbarer Text

In SVG 1.2 soll das Editieren von Text möglich werden. Das soll für `text` und `flowDiv` als Eltern-Elemente gelten. Dafür wird das Attribut `editable` eingeführt. Die zugehörigen Werte `true` oder `false` bestimmen, ob der entsprechende Inhalt dieser Elemente editiert werden kann. Bei `true` sollen sämtliche Kind-Elemente editierbar sein, während `false` kein Editieren zulässt. (siehe Listing 6)

Listing 6: Editierbarer Text mit `editable`

```
...
<text editable="true" x="30" y="100">
  <tspan x="30" dy="1.2em">
    Editierbarer Text von Kind 1</tspan>
  <tspan x="30" dy="1.2em">
    Editierbarer Text von Kind 2</tspan>
</text>
<text editable="false" x="30" y="150">Nicht
editierbarer Text</text>
...
```

Für editierbaren Text sollen SVG-Programme die CSS Pseudo-Klasse **:edited** unterstützen. Damit wird das aktuelle editierbare Textfeld formatiert. Details zur aktuellen Textauswahl soll der Anwender über ein spezielles Interface erhalten.

3.2.8 Attribut **requiredFonts**

Sehr wichtig im Bereich Text ist auch, dass ein verwendeter Font beim Anzeigen auch tatsächlich vorhanden ist. Manchmal ist das Einbetten von Fonts in SVG nicht sinnvoll (z. B. zu viele Glyphen). Hier könnte es bei nicht einheitlichen Voraussetzungen (z. B. verschiedene Fonts bei unterschiedlichen Plattformen) zu Problemen kommen, denen das neue Attribut **requiredFonts** vorbeugen kann. Damit kann festgestellt werden, ob ein spezieller, benötigter Font verfügbar ist. Wenn nicht können alternative Inhalte angeboten werden, die z. B. den System-Font verwenden (siehe dazu Listing 7).

Listing 7: Attribut **requiredFonts**

```
...
<switch>

  <text x="..." y="..." id="speziell"
    requiredFonts="2Tech" font-family="2Tech">
    <tspan x="..." dy="...">.....</tspan>
    <tspan x="..." dy="...">.....</tspan>
    <tspan x="..." dy="...">.....</tspan>
  </text>

  <flowRoot>
    <flowRegion>
      <!-- Form, in der der Text fließen soll -->
    </flowRegion>
    <flowPara>
      <flowTRef xlink:href="#alternativ">
    </flowPara>
  </flowRoot>

</switch>
...
```

3.2.9 Zu erwartende Textlänge

Mit SVG 1.2 gibt es den Wert **auto** für das Attribut **textLength**. Damit wird SVG-Anwendern erlaubt, die Zahl an Zeichen zu spezifizieren, die in einem Text-Element zu erwarten sind. Dies soll zum Abstimmen der Berechnungen des SVG-Programms mit den Erwartungen des Anwenders hilfreich sein.

3.2.10 Anwendungen im Bereich Text

Mit der Einführung von Fließtext in Version 1.2 wird SVG auch für Anwendungen interessanter, für die es als Auszeichnungssprache für Grafiken ursprünglich wohl nicht vorgesehen war: für Anwendungen, die den Umgang mit größeren Mengen an Text erfordern.

Dem Erstellen von Websites, Präsentationen, Schulungen oder vielleicht sogar Dokumentationen auf elektronischen Medien mit SVG wäre die gesteigerte Flexibilität im Bereich Text sehr zuträglich.

Aber auch dem kreativen Umgang mit Text ist durch die Neuerungen geholfen.

3.3 RCC und sXBL

3.3.1 Vorbemerkung

Der sog. *Rendering Custom Content (RCC)* war bis zum siebenten Working Draft eine feste Größe in SVG 1.2. Im achten Working Draft ist dieses SVG-Feature nicht mehr enthalten. An dessen Stelle tritt die *XML Binding Language von SVG (sXBL)*, die aber die Grundideen von RCC übernimmt. Aufgrund dieser Bedeutung soll an dieser Stelle trotzdem auf RCC eingegangen werden. Außerdem können so Veröffentlichungen hinzugezogen werden, die für sXBL noch nicht existieren. Zudem ist sXBL noch in keiner Software implementiert, während RCC im *ASV 6.0 preview 1* angewendet und veranschaulicht werden kann.

3.3.2 Rendering Custom Content

RCC dürfte ein sehr bedeutender Schritt bei der Entwicklung von SVG gewesen sein:

One of the most promising features introduced in SVG 1.2 is Rendering Custom Content (RCC) -- it offers a clean XML-centric extension mechanism to mix and match different XML namespaces within an SVG document. ... [QUINT03].

Im vorherigen Zitat klingt bereits an, woher diese Bedeutung rührt: Mit RCC ist SVG-Anwendern ein Mechanismus bereitgestellt worden, der erlaubt, auf Basis von XML eigene Komponenten (Elemente) zu erstellen, welche das vorhandene SVG-Repertoire erweitern. Diese Komponenten können in eigenen Namensräumen abgelegt und in einem SVG-Dokument kombiniert werden. Damit wird es möglich, Komponenten aus verschiedenen Namensräumen nahtlos in ein SVG-Dokument zu integrieren:

Rendering Custom Content (RCC), [...], offers a new framework for allowing seamless integration of custom XML grammars in SVG documents as well-defined extensions. ... [QUINT03].

Zudem ist dadurch eine modulare Anwendung von Komponenten möglich. Komponenten können so an verschiedenen Stellen (in verschiedenen SVG-Dokumenten) abgelegt, von dort her aufgerufen und wiederverwendet werden:

With this simple but elegant library architecture, nearly anyone can create an RCC library, put it online, and make it available to others on the Web. ... [QUINT03].

3.3.3 Shadow Trees

Die besondere Innovation von RCC ist aber, nicht nur die äußere Gestalt der Komponenten mit SVG-Mitteln beschreiben zu können, sondern auch ihr Verhalten bei bestimmten Ereignissen. Ermöglicht wird dies dadurch, dass für die Elemente in einer Komponente eine eigene Baumstruktur, also ein Komponenten-DOM-Fragment erzeugt werden kann. Dieses DOM-Fragment besteht unabhängig vom DOM des SVG-Dokuments (vgl. DOM-

Baum, Abbildung 10, S.34) und kann mit Script-Sprachen beeinflusst werden. Dies ist aber nur durch ein Script möglich, das innerhalb der Komponente aufgerufen wird. Für Scripte außerhalb der Komponente bleibt das DOM-Fragment verborgen, weswegen es auch als *Shadow Tree* bezeichnet wird. Ein Shadow Tree kann aus Container-, Grafik-, Animations-Elementen sowie Event-Handlern bestehen.

3.3.4 RCC-Elemente und ihre Aufgaben

Zum Erstellen von RCC-Komponenten stehen folgende Elemente zur Verfügung:

- **extensionDefs:**
definiert einen Satz von Komponenten,
- **elementDef:**
definiert eine Komponente, enthält die gesamte Beschreibung der Komponente und ihres Verhaltens,
- **defs:**
dient als Kind von **elementDef** als Behälter für referenzierte Objekte,
- **prototype:**
dient zum Erzeugen von Shadow Trees,
- **script:**
beinhaltet Scripte, mit dem die Komponenten und ihre Shadow Trees beeinflusst werden können,
- **handler:**
ähnlich wie **script**; beinhaltet ausführbare Scripte; erlaubt zudem das Anhängen eines Event-Listeners an ein XML-Element,
- **refContent:**
verhindert, dass Kind-Elemente von Komponenten in die Shadow Trees geklont werden; verweist nur auf die Kind-Elemente,

- **traitDef**:
definiert einen animierbaren Parameter einer Komponente.

Ursprünglich sollten noch die Elemente **transformer** und **param** zur Verfügung stehen. Mit ihnen sollten innerhalb eines SVG-Dokuments XSL-Transformationen möglich gemacht werden, was aber einen zu hohen Aufwand bei der Implementierung erfordert hätte. Dementsprechend sind diese Elemente auch im achten Working Draft zu SVG 1.2 nicht mehr enthalten.

3.3.5 RCC-Elemente im Zusammenhang

Das Element **extensionDefs** ist das Eltern-Element, das verschiedene Komponenten eines Satzes enthält. Es erhält über das Attribut **namespace** einen geeigneten Namensraum.

Innerhalb von **elementDef** wird eine Komponente definiert. Das kann z. B. eine Gruppe von mehreren Grundformen sein, die als Einheit aufgerufen werden soll. Die Komponente wird mit dem Attribut **name**, das einen entsprechenden Wert erhält, benannt.

Wenn das Verhalten der Gruppe beeinflusst werden soll, muss sie in das **prototype**-Element eingesetzt werden, da von diesem Element die Shadow Trees erzeugt werden, die von einem Script innerhalb von **elementDef** angesprochen werden können. Das Script selbst wird im **script**- oder **handler**-Element abgelegt.

Soll die Gruppe Inhalte enthalten, die nicht per Script beeinflusst werden müssen, kann das **refContent**-Element verwendet werden. Es verweist lediglich auf die Inhalte, sodass die Inhalte nicht in die Shadow Trees geklont werden.

Durch das Element **traitDef** wird ein damit definierter Parameter dem Animations-Teil der Software zum Anzeigen von SVG bekannt gemacht.

3.3.6 RCC-Beispiel

In Listing 8 (S.54), ab Zeile 20 ist ein Satz an Komponenten definiert. Ihm ist ein Namensraum (Zeile 20) zugeordnet. Darin ist ab

Zeile 22 eine Komponente, bestehend aus zwei Kreisen und Text, angelegt.

Von den Elementen innerhalb des **prototype**-Elements (ab Zeile 24) wird ein Shadow Tree erzeugt, der durch ein externes Script (Listing 9, S.55) angesprochen wird. Angesprochen und beeinflusst werden dabei verschiedene Attribute, die zur Beschreibung der Komponente nötig sind, wie Koordinaten, Radien, Farben, Texthöhe. Es wird z. B. der Radius des inneren Kreises so beeinflusst, dass er der Hälfte des Radius des äußeren Kreises entspricht. Außerdem erhalten die beeinflussten Attribute eigene Namen, wie z. B. der Kreis-Außen-Radius den Namen **kar**.

In Listing 8 (S.54) ab Zeile 37 wird nun die Komponente angewendet. Hier werden unter Verwendung des in Zeile 9 angegebenen Namensraums den im Script definierten Attributen die gewünschten Werte zugewiesen:

- **kx** und **ky**: die Koordinaten des Kreismittelpunkts,
- **kar**: der Kreis-Außen-Radius,
- **kaf** bzw. **kif**: die Farbe des Außen- bzw. Innenkreises,
- **th**: die Texthöhe,
- **tf**: die Textfarbe.

Damit kann die Komponente auf verschiedene Art beliebig oft gerendert werden. Abbildung 13 (S.56) zeigt die dreimalige Anwendung der Komponente entsprechend von Listing 8 und Listing 9. Die grafische Ausgabe wurde vom *ASV 6.0 preview 1* erzeugt.

Listing 8: Rendering Custom Content – rcckreis.svg

```
1  <?xml version="1.0" encoding="ISO-8859-1"
   standalone="no"?>
2  <!--
3  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.2//EN"
4     "http://www.w3.org/Graphics/SVG/1.2/DTD/svg12.dtd">
5  -->
6  <svg xmlns="http://www.w3.org/2000/svg" version="1.2"
7     xmlns:xlink="http://www.w3.org/1999/xlink"
8     xmlns:ev="http://www.w3.org/2001/xml-events"
9     xmlns:bsp="http://example.org/bsp">
10
11 <title>Rendering Custom Content</title>
12
13 <defs>
14 <style type="text/css"><![CDATA[
15     .rahmen {stroke: #000; stroke-width: 2; fill: none}
16 ]]>
17 </style>
18 </defs>
19
20 <extensionDefs namespace="http://example.org/bsp"
21     id="beispielkomponenten">
22     <elementDef name="kreiskombination">
23         <prototype>
24             <g>
25                 <circle id="kreisaussen"/>
26                 <circle id="kreisinnen"/>
27                 <text id="text"><refContent/></text>
28             </g>
29         </prototype>
30         <script ev:event="SVGBindEnd" xlink:href="rcckreis.js"
31             type="text/javascript"/>
32     </elementDef>
33 </extensionDefs>
34
35 <rect class="rahmen" x="0" y="70" width="500"
36     height="220"/>
37
38 <bsp:kreiskombination kx="130" ky="180" kar="80"
39     kaf="#CFCFCF" kif="#0F0F0F" th="22" tf="#FFFFFF">
40     Text 1</bsp:kreiskombination>
41 <bsp:kreiskombination kx="285" ky="180" kar="60"
42     kaf="#8F8F8F" kif="#CFCFCF" th="18" tf="#0F0F0F">
43     Text 2</bsp:kreiskombination>
44 <bsp:kreiskombination kx="400" ky="180" kar="40"
45     kaf="#0F0F0F" kif="#FFFFFF" th="12" tf="#0F0F0F">
46     Text 3</bsp:kreiskombination>
47 </svg>
```

Listing 9: Shadow Tree-Beeinflussung – rckkreis.js

```
/* basierend auf Script in rcc-simplebutton.svg von Dr. Thomas
   Meinike 05/04 http://www.datenverdrahten.de/svglbc/ */

var ver=getSVGViewerVersion();

if(ver.indexOf("Adobe")!=-1 && ver.indexOf("6.0")!=-1)
{
var el,st,ka,ki,tx,kx,ky,kar,kaf,kif,th,tf;

// Daten zum aktuellen kreiskombinations-Element abfragen
el=evt.target;
st=el.shadowTree;

ka=st.getElementById("kreisaussen");
ki=st.getElementById("kreisinnen");
tx=st.getElementById("text");

kx=el.getAttribute("kx"); // Kreis-x
ky=el.getAttribute("ky"); // Kreis-y
kar=el.getAttribute("kar"); // Kreis-Aussen-Radius
kaf=el.getAttribute("kaf"); // Kreis-Aussen-Farbe
kif=el.getAttribute("kif"); // Kreis-Innen-Farbe
th=el.getAttribute("th"); // Text-Hoehe
tf=el.getAttribute("tf"); // Text-Farbe

// kreisaussen
ka.setAttribute("cx",kx);
ka.setAttribute("cy",ky);
ka.setAttribute("r",kar);
ka.setAttribute("fill",kaf);

// kreisinnen
ki.setAttribute("cx",kx);
ki.setAttribute("cy",ky);
ki.setAttribute("r",parseFloat(kar)/2);
ki.setAttribute("fill",kif);

// Textinhalt
tx.setAttribute("font-size",th);
tx.setAttribute("font-family","Arial, Helvetica, sans-serif");
tx.setAttribute("fill",tf);
tx.setAttribute("text-anchor","middle");
tx.setAttribute("pointer-events","none");
tx.setAttribute("x",kx);
tx.setAttribute("y",parseFloat(ky)+parseFloat(th)/3);

}
```

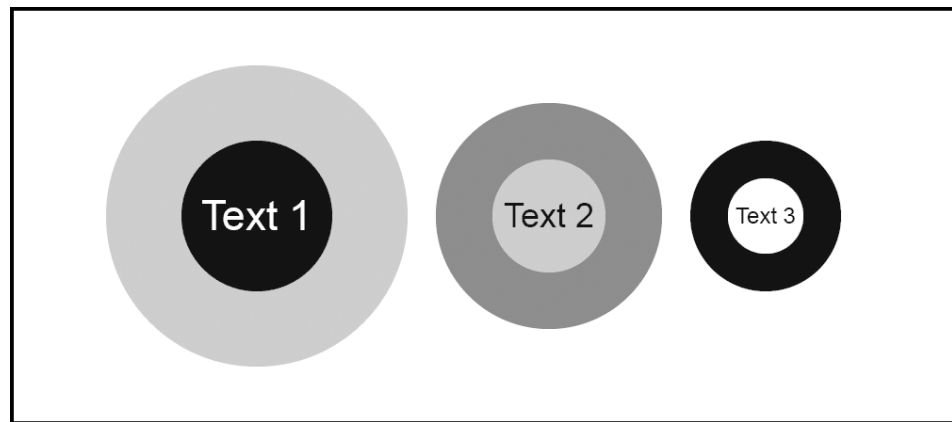


Abbildung 13: Komponenten mit RCC – rckkreis.svg und rckkreis.js

3.3.7 XML Binding Language von SVG

Die *XML Binding Language von SVG (sXBL)* stellt die aktuellste Entwicklung dar, um SVG mit »fremden« XML-Markup zu erweitern.²⁶ Diese neue Sprache beinhaltet zum größten Teil die Ideen und Funktionen, die schon von RCC her bestehen. Das Grundprinzip von RCC bleibt also unverändert; es geht immer noch um das Ablegen bzw. Übernehmen von eigenen XML-Komponenten in bzw. aus einem SVG-Dokument, inklusive ihrem Verhalten. Nach wie vor besitzen die Shadow Trees dabei eine besondere Bedeutung.

Zukünftig soll der SVG-spezifische Anwendungs-Schwerpunkt von sXBL erweitert werden. Die Entwicklung soll dann XML-übergreifend und unter der Bezeichnung *XML Binding Language (XBL)* weitergeführt werden.

3.3.8 Veränderungen in sXBL gegenüber RCC

In sXBL werden einige wichtige, grundlegende Begriffe festgeschrieben. Ein Dokument, in dem Komponenten in einem Namensraum abgelegt sind, wird als *Binding Source Document*

²⁶ die Grundlage aller folgenden Überlegungen ist [sXBLWD2]

bezeichnet. Das Dokument, das auf diese abgelegten Komponenten zugreift, wird als *Bound Document* bezeichnet.

Die Aufgaben der meisten RCC-Elemente sind in sXBL weitestgehend geblieben. Für diese Elemente werden nur neue Element-Namen eingeführt (vgl. Tabelle 5). Einige Elemente aus RCC sind jedoch nicht mehr in sXBL enthalten: **traitDef** bleibt SVG 1.2 vorbehalten; die Elemente **transformer** und **param** sind seit dem achten Working Draft auch in SVG 1.2 nicht mehr vorhanden. Weitere Unterschiede gegenüber RCC bestehen bei der Benennung und beim Übernehmen von Komponenten.

3.3.9 XBL-Namensraum

Um sXBL in ein SVG-Dokument zu integrieren, ist der XBL-Namensraum zu verwenden:

```
xmlns:xbl="http://www.w3.org/2004/xbl" .
```

Wie üblich muss dann den einzelnen sXBL-Elementen im SVG-Dokument das entsprechende Namensraum-Präfix vorangestellt werden.

3.3.10 sXBL-Elemente

In Tabelle 5 sind die in sXBL verfügbaren Elemente aufgeführt, wobei ihnen die entsprechenden Elemente aus RCC gegenüber gestellt sind.²⁷

sXBL-Element	RCC-Element	Aufgabe
xbl:xbl	extensionDefs	definiert einen Satz von Komponenten

²⁷ in Anlehnung an [sXBLWD2] Abschnitt 9. Comparison of sXBL to RCC

sXBL-Element	RCC-Element	Aufgabe
xbl:import	nicht vorhanden	importiert einen xbl:xbl -Komponenten-Satz oder alle xbl:xbl -Komponenten-Sätze
xbl:definition	elementDef	definiert eine Komponente, enthält die gesamte Beschreibung der Komponente und ihres Verhaltens
	defs	dient als Behälter für referenzierte Objekte
xbl:template	prototype	erzeugt Shadow Trees
xbl:handlerGroup	script oder handler	beinhaltet das Script, mit dem die Komponenten und ihre Shadow Trees beeinflusst werden können
xbl:content	refContent	verhindert, dass Kind-Elemente von Komponenten in Shadow Trees geklont werden; verweist nur auf die Kind-Elemente
	traitDef	definiert einen animierbaren Parameter einer Komponente

Tabelle 5: sXBL- und RCC-Elemente

3.3.11 Komponenten-Benennung mit QName

In RCC ist es nötig gewesen, jedem abgelegten Komponenten-Satz mit dem Attribut **namespace** einen Namensraum zuzuordnen:

```
...
<extensionDefs namespace="http://example.org/bsp"
id="beispielkomponenten">
....
```

In sXBL benötigt das Element **xbl:xbl** diesen Namensraum nicht mehr. Stattdessen wird in sXBL für das Element **xbl:definition** das Attribut **element** eingeführt, das als Wert einen *Qualified Name (QName)* erhält, der aus zwei Teilen besteht. Erster Teil ist das Präfix des Namensraums der Komponente; den zweiten Teil repräsentiert der (lokale) Name der Komponente. Damit kann nun jede Komponente eindeutig einem Namensraum zugeordnet werden und die Namensraum-Bezeichnung für den Komponenten-Satz entfällt. Der Namensraum, auf den sich *QName* bezieht, muss im Wurzelement angegeben sein:

```
...
<svg ... xmlns:bsp="http://example.org/bsp">
...
<xbl:xbl>
  <xbl:definition element="bsp:kreiskombination">
    <!-- Komponenten-Definition -->
  </xbl:definition>
</xbl:xbl>
....
```

3.3.12 Übernehmen von Komponenten

In RCC konnten Komponenten in ein SVG-Dokument übernommen werden, indem mit dem Element **extensionDefs** und **xlink:href** auf einen Komponenten-Satz in einem anderen SVG-Dokument zugegriffen wird:

```

...
<extensionDefs
xlink:href="http://example.org/bsp/rcckreis.svg#b
ispielkomponenten"/>
...

```

Der entsprechende Komponenten-Satz in dem anderen SVG-Dokument:

```

...
<extensionDefs namespace="http://example.org/bsp"
id="beispielkomponenten">
  <!-- Komponenten-Definition -->
</extensionDefs>
...

```

In sXBL wird nun zum Übernehmen von Komponenten-Sätzen das Element `xbl:import` mit dem Attribut `bindings` verwendet:

```

...
<xbl:import
bindings="http://example.org/bsp/sxblkreis.svg"/>
...

```

Damit werden alle Komponenten-Sätze des entsprechenden *Binding Source Document* übernommen:

```

<svg ... xmlns:bsp="http://example.org/bsp">
...
<xbl:xbl>
<xbl:definition element="bsp:kreiskombination1">
  <!-- Komponenten-Definition -->
</xbl:definition>
<xbl:definition element="bsp:kreiskombination2">
  <!-- Komponenten-Definition -->
</xbl:definition>
</xbl:xbl>
...

```

Es ist in sXBL nun auch möglich, gezielt auf einzelne Komponenten in einem speziellen Namensraum zuzugreifen und

sie zu übernehmen. Steht in einem SVG-Dokument (*Bound Document*)

```
...  
<xbl:definition element="bsp:kreiskombination1"  
ref="http://example.org/bsp/sxblkreis.svg"/>  
... ,
```

so wird damit genau die eine Komponente mit dem *Qualified Name* `bsp:kreiskombination1` übernommen, die sich in dem *Binding Source Document* befindet, auf das `ref` verweist.

3.3.13 Fokus-Behandlung bei Komponenten

Sind Elemente in Komponenten per Fokus anwählbar (siehe Bereich Navigation, Listing 24, S.89), so werden sie abhängig von ihrer Stellung im SVG-Dokument getrennt behandelt. Jede Komponente besitzt ihre eigene Tab-Reihenfolge (*tab-order*). Zuerst werden alle anwählbaren Elemente außerhalb der Komponenten, d. h. in der obersten Stufe des Dokuments, durchlaufen. Dann wechselt die Fokus-Behandlung in die oberste Komponente. Ist diese durchlaufen, wird zur nächsten Komponente gewechselt usw.

3.3.14 sXBL-Beispiel

In Listing 10 (S.62) ist das RCC-Beispiel aus Listing 8 (S.54) aufgegriffen und in sXBL-Syntax umgesetzt. Hierbei kommt Listing 11 (S.63) zum Einsatz, welches sich nur geringfügig von Listing 9 (S.55) unterscheidet. Die grafische Ausgabe von Listing 10 und Listing 11 sollte dabei Abbildung 13 (S.56) entsprechen, was aber praktisch nicht überprüft werden kann, da sXBL noch in keiner Software implementiert ist.

Listing 10: SVG's XML Binding Language – sxblkreis.svg

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!--
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.2//EN"
"http://www.w3.org/Graphics/SVG/1.2/DTD/svg12.dtd">
-->
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xbl="http://www.w3.org/2004/xbl"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:bsp="http://example.org/bsp">

<title>SVG's XML Binding Language</title>

<defs>
  <style type="text/css"><![CDATA[
    .rahmen {stroke: #000; stroke-width: 2; fill: none;}
  ]]>
</style>
</defs>

<xbl:xbl>
  <xbl:definition element="bsp:kreiskombination">
    <xbl:template>
      <g>
        <circle id="kreisaussen"/>
        <circle id="kreisinnen"/>
        <text id="text"><xbl:content/></text>
      </g>
    </xbl:template>
    <xbl:handlerGroup>
      <handler ev:event="xbl:bound"
        scriptContentType="text/javascript"
        xlink:href="sxblkreis.js"/>
    </xbl:handlerGroup>
  </xbl:definition>
</xbl:xbl>

<rect class="rahmen" x="0" y="70" width="500" height="220"/>

<bsp:kreiskombination kx="130" ky="180" kar="80"
  kaf="#CFCFCF" kif="#0F0F0F" th="22" tf="FFFFFF">
  Text 1</bsp:kreiskombination>
<bsp:kreiskombination kx="285" ky="180" kar="60"
  kaf="#8F8F8F" kif="#CFCFCF" th="18" tf="#0F0F0F">
  Text 2</bsp:kreiskombination>
<bsp:kreiskombination kx="400" ky="180" kar="40"
  kaf="#0F0F0F" kif="FFFFFF" th="12" tf="#0F0F0F">
  Text 3</bsp:kreiskombination>

</svg>

```

Listing 11: Shadow Tree-Beeinflussung – sxbkreis.js

```
/* basierend auf Script in rcc-simplebutton.svg von Dr. Thomas
   Meinike 05/04 http://www.datenverdrahten.de/svglbc/ */

// Daten zum aktuellen kreiskombinations-Element abfragen
el=evt.target;
st=el.xblShadowTree;

ka=st.getElementById("kreisaussen");
ki=st.getElementById("kreisinnen");
tx=st.getElementById("text");

kx=el.getAttribute("kx"); // Kreis-x
ky=el.getAttribute("ky"); // Kreis-y
kar=el.getAttribute("kar"); // Kreis-Aussen-Radius
kaf=el.getAttribute("kaf"); // Kreis-Aussen-Farbe
kif=el.getAttribute("kif"); // Kreis-Innen-Farbe
th=el.getAttribute("th"); // Text-Hoehe
tf=el.getAttribute("tf"); // Text-Farbe

// kreisaussen
ka.setAttribute("cx",kx);
ka.setAttribute("cy",ky);
ka.setAttribute("r",kar);
ka.setAttribute("fill",kaf);

// kreisinnen
ki.setAttribute("cx",kx);
ki.setAttribute("cy",ky);
ki.setAttribute("r",parseFloat(kar)/2);
ki.setAttribute("fill",kif);

// Textinhalt
tx.setAttribute("font-size",th);
tx.setAttribute("font-family","Arial, Helvetica, sans-serif");
tx.setAttribute("fill",tf);
tx.setAttribute("text-anchor","middle");
tx.setAttribute("pointer-events","none");
tx.setAttribute("x",kx);
tx.setAttribute("y",parseFloat(ky)+parseFloat(th)/3);
```

3.3.15 Anwendung von sXBL

Die XML Binding Language von SVG stellt einen sehr sinnvollen Erweiterungs-Mechanismus dar. Überall, wo es sich lohnt, vorhandene Bestandteile wiederzuverwenden, ist auch ihr Einsatz lohnend.

So dürfte das Ablegen und Übernehmen von immer wieder benötigten Formularen, die mittels SVG beschrieben sind, ein sinnvoller Anwendungsfall sein. Auch der Einsatz für Buttons und ähnliche interaktive Elemente erscheint aufgrund der Möglichkeit, das Verhalten dieser Elemente über Shadow Trees zu steuern, sinnvoll. Der konsistente Einsatz einmal erstellter Komponenten in einem umfangreichen Informationsangebot bringt auch hinsichtlich der Wiedererkennung solcher Komponenten Vorteile und dürfte damit auch einen didaktischen Wert besitzen.

Daraus ergeben sich weitreichende Anwendungsmöglichkeiten. Sie erstrecken sich von der Modellierung von standardisierten Eingabemasken über die Erstellung interaktiver Elemente bis hin zum Einsatz als allgemeiner, flexibler Mechanismus zum Ablegen- und Wiederverwenden von Komponenten im Web. Letztgenanntes dürfte mit der Entwicklung der *XML Binding Language* noch bedeutender werden und weit über den derzeit noch gegebenen SVG-Rahmen hinausreichen.

3.4 Multiple Seiten

3.4.1 Konzept der multiplen Seiten

Die sog. *multiplen Seiten* dürften eine weitere herausragende Neuerung in SVG 1.2 sein. Sie greifen das von den Printmedien oder auch von Präsentations-Programmen bekannte Konzept der Mehrseitigkeit auf. Nun soll auch in SVG-Dokumenten ähnlich flexibel mit Inhalten umgegangen werden können, wie in den Vorbildern.

Mit den multiplen Seiten wird es nun möglich, verschiedene Seiten in einem SVG-Dokument oder einem SVG-Fragment zu erstellen und sie getrennt zu handhaben. Hierbei kann auch von einem einfachen Vorlage-Mechanismus Gebrauch gemacht werden.

Gleichzeitig wird dieses bekannte Konzept erweitert: Das, was in einem SVG-Dokument als Seite bezeichnet wird, kann zeitliche Veränderungen wahrnehmen und interaktiv sein. Bei den multiplen Seiten verschmelzen so das klassische Print- bzw. Präsentations-Konzept und die Möglichkeiten von SVG zu einem leistungsstarken Feature, das in dieser Kombination neuartig ist.

3.4.2 Trennung von Seiten

Seiten sind in SVG 1.2 nichts anderes als separate Gruppen von Inhalt. Diese Gruppen werden jeweils einzeln und nacheinander angezeigt. Die Trennung ist hierbei wichtig, da sie u. a. die fließende Wiedergabe von Animationen erlaubt. Hierdurch wird es möglich, einzelne Animations-Szenen auf separate Seiten zu platzieren, die von der SVG-Software einzeln nachgeladen werden können.

3.4.3 Elemente für Seiten

Das Element **pageSet** beinhaltet eine zusammengehörende Anzahl von Seiten. Es kann (SMIL)-Attribute besitzen, mit denen sein Verhalten über einen bestimmten Zeitraum beschrieben wird, wie z. B. **begin**, **dur**, **end**.²⁸

Das Element **page** entspricht einer Seite, d. h. in SVG einer Gruppe von Inhalt. Jedes **page**-Element besitzt seinen eigenen lokalen Bereich und ist von anderen **page**-Elementen abgegrenzt, was Definitionen und Referenzen angeht. Somit ist es dem Wurzelement **svg** ähnlich; im Unterschied dazu erlaubt es jedoch keine Transformation und Positionierung von Elementen. Als Attribute stehen ebenfalls (SMIL)-Attribute wie **begin**, **dur**, **end** zur Verfügung. Hinzu kommen die Attribute **transIn** und **transOut** (neu in SVG 1.2) für Übergangseffekte zwischen nacheinander angezeigten Seiten. Ein weiteres notwendiges Feature ist die Eigenschaft **page-orientation**, mit der die Seite in 90-Grad-Schritten gedreht werden kann, wobei sie automatisch platziert wird. Damit können Seiten in Hoch- oder Querformat angezeigt werden.

²⁸ vgl. [SMIL20REC] Abschnitt 10.3.1 Attributes

3.4.4 Beispiel für multiple Seiten

In Listing 12 ist das Grundgerüst für einen Satz multipler Seiten aufgeführt. Vor dem Element `pageSet` befinden sich in einem `defs`-Bereich Definitionen, die für alle folgenden Seiten gelten. Das darauf folgende `g`-Element enthält Inhalte, die unabhängig von den Seiten-Inhalten immer gerendert werden, was einem einfachen Vorlage-Mechanismus, wie er von Präsentations-Programmen bekannt ist, gleichkommt. Darauf folgen die eigentlichen Seiten, die jeweils sofort (0 Sekunden nach dem Laden) angezeigt und mit einem Mausklick »beendet« werden.

Listing 12: Multiple Seiten – pages.svg

```
...
<svg width="500" height="375" viewBox="0 0 500 375"
  xmlns="http://www.w3.org/2000/svg" version="1.2"
  streamedContents="discard">
...
<defs><!-- Globale Definitionen --></defs>
<g><!-- Stets sichtbare Inhalte --></g>

<pageSet>
  <page id="seite1" begin="0s;seite3.end"
    end="seite1.click">
    <defs><!-- Lokale Definitionen --></defs>
    <!-- Seiten-Inhalte -->
  </page>
  <page id="seite2" begin="seite1.end"
    end="seite2.click"
    style="page-orientation:90;">
    <!-- Seiten-Inhalte -->
  </page>
  <page id="seite3" begin="seite2.end"
    end="seite3.click">
    <!-- Seiten-Inhalte -->
  </page>
</pageSet>

  <!-- Verboten für Inhalte -->

</svg>
```

Zu achten ist außerdem darauf, dass sich zwischen dem letzten schließenden **pageSet**-Tag und dem schließenden **svg**-Tag keine Inhalte befinden dürfen.

3.4.5 Anwendung der multiplen Seiten

Die multiplen Seiten eignen sich hauptsächlich zum Erstellen von mehrseitigen Präsentationen und für das Handhaben von Seiten für den Druck. Wie im achten Working Draft zu SVG 1.2 auch angedeutet, hören dort die Möglichkeiten aber noch nicht auf. So können die Seiten auch als Behälter für Animations-Szenen dienen und so das schrittweise Herunterladen der Daten (*Streaming*) unterstützen. Da die multiplen Seiten interaktiv und Zeit-sensitiv gestaltet werden können, dürften sie sich auch für interaktive, mehrteilige Anwendungen mit Film-Charakter eignen.

3.5 Vektor-Effekte

3.5.1 Konzept der Vektor-Effekte

Mit den Vektor-Effekten werden in SVG 1.2 die Möglichkeiten erweitert, um grafische Objekte effektiv zu zeichnen. Damit wird es möglich, verschiedene Geometrien mit einer Farbe zu füllen, (Umriss-)Linien variabel zu gestalten und *Marker* zu definieren. Außerdem stehen auch logische Operationen wie Zusammenfügen, Umkehren, Vereinigen von Geometrien zur Verfügung. Wobei alles an einer Stelle und innerhalb eines einzigen Vektor-Effekts definiert werden kann und anschließend auf die verschiedensten Grundformen oder Pfade angewendet werden kann.

3.5.2 Element **vectorEffect**

Das Element **vectorEffect** ist das Eltern-Element aller möglichen Vektor-Effekte. Sein Attribut **vectorEffectUnits** und die dazugehörigen Werte **userSpaceOnUse** (Standard) oder **objectBoundingBox** spezifizieren das Koordinatensystem des Effekts.

Ein weiteres mögliches Attribut ist **clipout**, das **normal** und **clip** als Werte haben kann. Beim Wert **normal** werden die

Resultate von einzelnen Füll-Operationen miteinander verrechnet (mit der Komposition **src-over**, siehe Abschnitt Rendering-Effekte). Der Wert **clip** hat den Effekt, dass bereits gemalte Bereiche nicht übermalt werden.

3.5.3 Globale Attribute

Folgende Attribute sollen für alle Vektor-Effekte möglich werden:

- **result=Output-Name**:
beschreibt den Output der aktuellen Vektor-Effekt-Form,
- **in="SourcePath" | Input-Name** bzw.
in2="SourcePath" | Input-Name:
definieren den Input der aktuellen Vektor-Effekt-Form; bei **SourcePath** wird die Außenlinie der Form verwendet, die diesen Effekt referenziert,
- **transform=<transform>** bzw.
transformPath=<transform>:
unterstützen Koordinaten-System-Transformationen für einige Vektor-Effekt-Elemente.

3.5.4 Elemente für Vektor-Effekte

Die eigentlichen Operationen, die einen Effekt erzeugen, werden mit Hilfe von Elementen mit der Vorsilbe **ve** beschrieben. Diese Elemente erfüllen eine spezielle Aufgabe. Der SVG-Anwender kann/muss aus einer ganzen Reihe von Elementen auswählen und sich so einen gewünschten Effekt zusammenstellen:

- **veStrokePath**:
erzeugt eine Außenlinie von der Input-Pfad-Umrisslinie,
- **veSetback**:
führt eine »Setze-Pfad-Zurück-Operation« aus; bricht den Pfad in einzelne Segmente auf und kürzt beide Enden jedes Segments um die durch **setback-offset** angegebene Distanz,

- **veAffine:**
transformiert einen Pfad unter Verwendung der spezifizierten Transformations-Matrix,
- **veReverse:**
ordnet die Pfad-Segmente in umgekehrter Reihenfolge an; produziert die selbe Form wie der Original-Pfad,
- **veJoin:**
fügt das Ende und den Anfang zweier Pfade aneinander,
- **veUnion:**
erzeugt eine Außenlinie der Vereinigung von zwei Formen,
- **veIntersect:**
erzeugt eine Außenlinie der Überschneidung von zwei Formen,
- **veExclude:**
erzeugt eine Außenlinie des Ausschlusses der zweiten Form (**in2**) von der ersten Form (**in**),
- **veFill:**
füllt eine Form mit der Farbe und Opazität seiner Eigenschaften **fill** und **fill-opacity**,
- **veStroke:**
erzeugt eine Form, die den Pfad einer Umrisslinie einer Form repräsentiert,
- **veMarker:**
zeichnet Marker entlang des Input-Pfades,
- **veMarkerPath:**
verhält sich wie **veMarker**, aber anstatt zu zeichnen, erzeugt es Output durch Herbeiführen einer Vereinigung der Pfade aller Marker,

- **vePath:**
erzeugt den Pfad der Form, die es referenziert; die Referenzen kommen von **vePathRef**-Elementen,
- **vePathRef:**
ist ein Kind-Element von **vePath**, das einen individuellen Pfad referenziert; darf nur Grundformen, Pfade, Text-Elemente referenzieren.

3.5.5 Eigenschaft **vector-effect**

Die Eigenschaft **vector-effect** für grafische Elemente gibt den anzuwendenden Vektor-Effekt an. Als Werte sind **default**, **non-scaling-stroke**, **inherit** bzw. uri-Zuweisungen möglich.

Der Wert **default** entspricht dem folgendem Vektor-Effekt, der auch das Standard Rendering-Verhalten von SVG 1.1 repräsentiert:

```
<vectorEffect>
  <veFill/>
  <veStroke/>
  <veMarker/>
</vectorEffect> .
```

Der Wert **non-scaling-stroke** ist ein vordefinierter Vektor-Effekt, der die Strichbreite eines Objekts von Transformationen und Zoom ausnimmt:

```
<vectorEffect>
  <veFill/>
  <veStrokePath in="SourcePath"/>
  <veFill transform="ref(host)"
    fill="currentStroke"/>
  <veMarker in="SourcePath"
    transform="ref(host)"/>
</vectorEffect> .
```

Um einen selbst zusammengestellten Vektor-Effekt auf ein Element anzuwenden, bieten sich uri-Zuweisungen mit einer **id** an (siehe Listing 13).

3.5.6 Beispiel für einen Vektor-Effekt

Listing 13 zeigt einen definierten Vektor-Effekt, der ein Rechteck und einen Kreis doppelt umranden soll.

Listing 13: Vektor-Effekte – effvecto.svg

```
...
<vectorEffect id="doppelumriss">
  <veStrokePath stroke="#000" stroke-width="9"
    in="SourcePath"/>
  <veStrokePath stroke="#FFF" stroke-width="5"
    in="SourcePath"/>
</vectorEffect>

<rect transform="translate(150,140)" x="0" y="0"
  width="100" height="50" style="fill:#CCC;
  vector-effect:url(#doppelumriss)"/>

<circle transform="translate(300,140)" cx="25" cy="25"
  r="20" style="fill:#CCC;
  vector-effect:url(#doppelumriss)"/>
...
```

Abbildung 14 zeigt die grafische Ausgabe, die SVG 1.2-kompatible SVG-Programme für Listing 13 zukünftig liefern sollten.

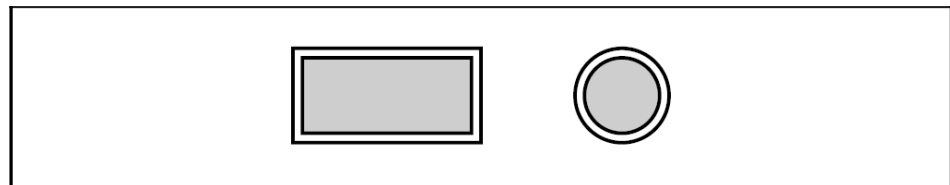


Abbildung 14: Vektor-Effekte – effvecto.svg

3.6 Fixierte Transformationen

SVG unterscheidet prinzipiell drei Gattungen von Transformationen:

- Transformationen, die vom SVG-Programm vollzogen werden,
- Transformationen, die im **svg**-Element mit dem Attribut **viewBox** definiert sind,
- Element-Transformationen mit dem Attribut **transform**, die auf einzelne Elemente angewendet werden (siehe [Transformationen in SVG](#), Abbildung 9, S.32).

SVG 1.2 erweitert die Element-Transformationen, indem es erlaubt, Elemente an der **svg**-Transformation zu fixieren. Das bedeutet, dass sämtliche Transformationen wie Verschiebungen oder Skalierungen, die auf fixierte Elemente angewendet werden, wirkungslos bleiben und diese fixierten Elemente stets die im **svg**-Element definierte Transformation beibehalten. Dieses Feature bietet sich daher für umfangreichere SVG-Anwendungen an, um selektiv einzelne Elemente von Element-Transformationen auszuschließen.

Um das zu realisieren, steht der mit SVG 1.2 neu eingeführte Wert **ref(svg, x, y)** für das Attribut **transform** zur Verfügung. Der Parameter **svg** fixiert das entsprechende Element an die **svg**-Transformation; die Parameter **x** und **y** sind optional. Sie würden eine zusätzliche Verschiebung des Elements veranlassen.

Listing 14 auf S.73 zeigt die Anwendung des Features, das in derzeit aktuellen SVG-Programmen noch nicht implementiert ist. Abbildung 15 auf S.74 zeigt deswegen die Ausgabe eines Codes, der für den *ASV 6.0 preview 1* angepasst wurde. Sie sollte identisch sein mit den Ausgaben zukünftiger SVG 1.2-kompatibler Programme.

Listing 14: Fixierte Transformationen – transfix.svg

```

<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<!--
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.2//EN"
"http://www.w3.org/Graphics/SVG/1.2/DTD/svg12.dtd">
-->
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
viewBox="0 0 500 300">
<title>Fixierte Transformationen (SVG 1.2)</title>
...
<g>
...
<rect x="50" y="110" width="80" height="40"
style="fill:#CCC; stroke:#000; stroke-width:2;"/>
<circle cx="90" cy="180" r="20"
style="fill:#AAA; stroke:#000; stroke-width:2;"/>
<polyline
points="75,215 105,215 105,245 135,245 135,275 75,275"
style="fill: none; stroke: #000; stroke-width: 2;"/>
</g>
<g transform="translate(185,100)">
<g transform="scale(0.7,0.7)">
...
<rect x="0" y="10" width="80" height="40"
style="fill:#CCC; stroke:#000; stroke-width:2;"/>
<circle cx="40" cy="80" r="20"
style="fill:#AAA; stroke:#000; stroke-width:2;"/>
<polyline
points="25,115 55,115 55,145 85,145 85,175 25,175"
style="fill: none; stroke: #000; stroke-width: 2;"/>
</g>
</g>
<g transform="translate(320,100)">
<g transform="scale(0.7,0.7)">
...
<rect x="0" y="10" width="80" height="40"
style="fill:#CCC; stroke:#000; stroke-width:2;"/>
<circle cx="40" cy="80" r="20"
style="fill:#AAA; stroke:#000; stroke-width:2;"/>
...
<text transform="ref(svg)" class="textklein"
x="335" y="205">transform="ref(svg)"</text>
<polyline transform="ref(svg)"
points="335,215 365,215 365,245 395,245 395,275 335,275"
style="fill:none; stroke:#000; stroke-width:2;"/>
</g>
</g>
</svg>

```

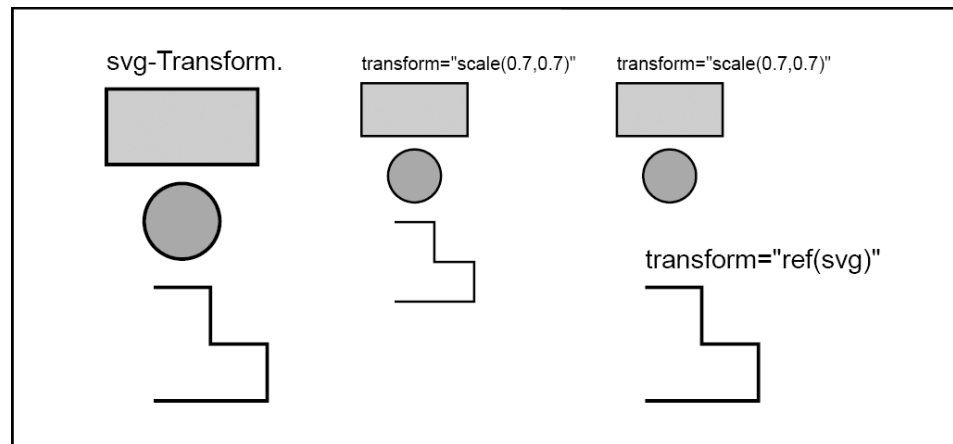


Abbildung 15: Fixierte Transformationen – transfix.svg

3.7 Rendering-Effekte

3.7.1 Konzept der Rendering-Effekte

SVG 1.2 erweitert die Möglichkeiten zur Gestaltung von grafischen Objekten durch neue, vielfältige Rendering-Effekte. Dahinter verbergen sich grafische Berechnungen (erweitertes *Alpha Compositing*) nach einem bestimmten Modell (erweitertes *Porter-Duff-Modell*), Überblendungen und andere spezielle Effekte. Rendering-Effekte werden auf bereits gerenderte Objekte auf der Zeichenfläche angewendet. Damit könnte SVG 1.2 zukünftig auch für Grafik-Profis, denen die bisherigen Möglichkeiten in SVG nicht ausreichen, interessant werden.

3.7.2 Eigenschaft `comp-op`

Mit der Eigenschaft `comp-op` (für *compositing operation*) kann Container- und Grafik-Elementen eine Vielzahl vorgefertigter Berechnungen zugewiesen werden, wofür über 20 Werte existieren. Die Namen der Werte stehen zum Zeitpunkt des Schreibens dieser Arbeit noch nicht fest, sodass sich hierbei zukünftig noch Änderungen ergeben können:

```
clear | src | dst | src-over (Standard) | dst-over
| src-in | dst-in | src-out | dst-out | src-atop
```

```
| dst-atop | xor | plus | multiply | screen |  
overlay | darken | lighten | color-dodge | color-  
burn | hard-light | soft-light | difference |  
exclusion | inherit .
```

Auf eine weitere Beschreibung der Werte wird an dieser Stelle verzichtet, da dies wegen der Vielzahl der Werte, als auch aus fachlichen Gründen nicht sinnvoll wäre.²⁹

3.7.3 Eigenschaft **clip-to-self**

Mit der Eigenschaft **clip-to-self**, die für Container- und Grafik-Elemente zur Verfügung steht, kann eine Beschneidungs-Maske erzielt werden. Dazu dient der Wert **true**, der bewirkt, dass »sich« das Element, dem die Eigenschaft zugewiesen wurde, aus dem im Code vorher aufgeführten Element »ausschneidet«. Weitere mögliche Werte sind **false** (Standard) und **inherit**.

3.7.4 Eigenschaft **enable-background**

Mit der Eigenschaft **enable-background** für Container-Elemente und dem Wert **accumulate** (Standard) kann ein Effekt erzielt werden, der durch einen zusätzlichen Opazitäts-Kanal zustande kommt und mit dem Ansammeln und Abbilden von durch **comp-op** spezifizierten Effekten zusammenhängt. Weitere mögliche Werte sind **new** und **inherit**.

3.7.5 Eigenschaft **knock-out**

Die Eigenschaft **knock-out** für Container-Elemente wirkt in Verbindung mit dem Wert **true** und der Eigenschaft **enable-background**. Dabei ersetzen die Farbe und Opazität eines Objekts in einem Container-Element die Farbe und Opazität anderer Objekte in dem Container-Element. Weitere mögliche Werte sind **false** (Standard) und **inherit**.

²⁹ die fundiertesten Informationen und Beispiele dazu bietet [SVG12WD8] Abschnitt 10.1.6 The comp-op property

3.7.6 Beispiel für Rendering-Effekte

Listing 15 ist der Versuch, die zuvor beschriebenen Effekte ansatzweise anzuwenden.

Listing 15: Rendering-Effekte – effrend.svg

```
<g transform="translate(25,110)">
...
  <g>
    <rect x="0" y="70" width="100" height="50"
      style="fill:#777"/>
    <circle cx="50" cy="120" r="25"
      style="clip-to-self:true; comp-op:src-in;
      fill:#CCC"/>
  </g>
</g>
...
<g transform="translate(125,110)"
  style="enable-background:accumulate">
...
  <g>
    <rect x="50" y="70" width="100" height="50"
      style="fill:#777"/>
    <circle cx="100" cy="120" r="25"
      style="comp-op:multiply; clip-to-self:false;
      fill:#CCC"/>
  </g>
</g>
...
<g transform="translate(335,110)"
  style="enable-background:new">
...
  <g style="knock-out:true">
    <rect x="20" y="70" width="100" height="50"
      style="fill:#777"/>
    <circle cx="70" cy="120" r="25"
      style="comp-op:overlay; clip-to-self:true;
      fill:#CCC"/>
  </g>
</g>
```

3.8 Bereiche Darstellung und Farbe

Die folgenden Abschnitte enthalten spezielle Neuerungen, die auf den vorhandenen Möglichkeiten in SVG 1.1 aufbauen, diese verfeinern oder sie einfach nur ergänzen.

3.8.1 Füllung des Hintergrunds

Um den Hintergrund eines Bildausschnitts (*viewport*) zu gestalten, werden neue Eigenschaften eingeführt:

Die Eigenschaft **background-fill** füllt den Hintergrund eines Bildausschnitts mit einer anzugebenden Farbe. Für Werte können hierbei die üblichen Farb-Notationen verwendet werden. Der Wert **none** setzt den Hintergrund farblos.

Die Eigenschaft **background-fill-opacity** legt die Opazität des Hintergrunds fest. Hierbei können Werte von **0.0** bis **1.0** (Standard) verwendet werden.

Listing 16: Eigenschaften background-fill, background-fill-opacity

```
...
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
  style="background-fill:#000">
  <!-- Inhalte auf schwarzem Hintergrund -->

  <svg x="150" y="50" width="200" height="100"
    style="background-fill:#000;
      background-fill-opacity:0.5">
    <!-- Inhalte auf halbdurchsichtigem
      schwarzen Hintergrund -->

  </svg>
</svg>
```

3.8.2 Element solidColor

Mit dem Element **solidColor** kann eine Farbe und ihre Opazität als Referenz hinterlegt werden. Das Element besitzt dafür die Eigenschaft **solid-color**, der ein Farbwert zugeordnet wird so-

wie die Eigenschaft **solid-opacity**, der Werte von **0.0** bis **1.0** zugewiesen werden können. Listing 17 zeigt ein Beispiel.

Listing 17: Element `solidColor`

```
...
<defs>
  <solidColor id="zartesblau"
    style="solid-color:#00F; solid-opacity:0.2"/>
</defs>
...
<rect x="30" y="80" width="100" height="50"
  style="fill:url(#zartesblau); stroke:#000;
  stroke-width:2"/>
...
```

3.8.3 Eigenschaft `rendering-color-space`

Die Eigenschaft **rendering-color-space** definiert den Farbraum, in dem die Rendering-Operationen eines Elements stattfinden. Als Werte sind möglich:

- **auto** (Standard):
entspricht der Eigenschaft **color-interpolation:linearRGB**; wird in späteren Spezifikationen **color-interpolation** entfernt, soll **auto sRGB** entsprechen,
- **sRGB**, **scRGB**, **linearRGB**:
spezifizieren den verwendeten Farbraum
- **<name>**:
Name eines definierten Farbprofiles in der Farbprofil-Datenbank des SVG-Programms,
- **<uri>**:
eine URI-Referenz zum Farbprofil,
- **inherit**: vererbt.

3.8.4 XLink-Attribut für `style`-Element

Stylesheets können nun alternativ zur *Processing Instruction* auch innerhalb eines SVG-Dokuments aufgerufen werden, da für das Element **style** das Attribut **xlink:href** eingeführt wird:

```
...  
<style xlink:href="styles.css" type="text/css"/>  
....
```

3.8.5 Akzentuierung

Um ein Element zu akzentuieren, steht in SVG 1.2 die CSS-Pseudo-Klasse **:highlight** zur Verfügung. Mit

```
text.optischeranker:highlight  
  {fill:blue; font-weight:bold}
```

würden alle Text-Elemente mit **class="optischeranker"** in der beschriebenen Weise akzentuiert werden.

Diese Pseudo-Klasse eignet sich auch, um Elementen Vektor-Effekte zuzuweisen:

```
element.klasse:highlight  
  {vector-effect:url(#idvectoreffect)}.
```

3.8.6 Attribut shadowInherit

Das Attribut **shadowInherit** regelt die Vererbung von Eigenschaften in Shadow Trees. Mögliche Werte sind:

- **onDefine:**
Vererbung von Eigenschaften in den Shadow Tree vom Element aus, was den Shadow Tree definiert,
- **onUse** (Standard):
Vererbung von Eigenschaften in den Shadow Tree vom Element aus, was den Shadow Tree verwendet,
- **dynamic:**
entspricht in SVG **onDefine**, außer beim **feImage**-Element,
- **none:**
keine Vererbung in den Shadow Tree; erlaubt das Wiederverwenden von SVG-Inhalten ohne Risiko von ungewünschten Eigenschafts-Änderungen durch die SVG-Umgebung.

Listing 18: Attribut shadowInherit

```

...
<defs fill="yellow">
  <rect id="recht1" x="0" y="0" width="50"
height="50"/>
</defs>

<use x="50" y="80" xlink:href="#recht1"
  shadowInherit="onDefine" fill="red"/>
<use x="150" y="80" xlink:href="#recht1"
  shadowInherit="onUse" fill="green"/>
<use x="250" y="80" xlink:href="#recht1"
  shadowInherit="none" fill="blue"/>
...

```

Listing 18 zeigt die Anwendung von **shadowInherit**. Das erste **use** wird gelb gerendert, da gelb mit **onDefine** vom Element, das den Shadow Tree definiert, vererbt wird. Das zweite **use** wird grün gerendert, da grün vom eigenen Element, das den Shadow Tree verwendet, vererbt wird. Das dritte **use** wird schwarz (Standard) gerendert, da weder vom Definitions-Element des Shadow Trees, noch vom Verwendungs-Element des Shadow Trees Eigenschaften übernommen werden.

3.8.7 Attribute für filter-Element

Die neu eingeführten Attribute

- **filterMarginsUnits**,
- **filterPrimitiveMarginsUnits**,
- **dx**, **dy**, **dw**, **dh**

für das **filter**-Element sollen das Problem beheben, dass die Definition eines Filters bisher mit einer bestimmten Fläche verbunden war und man diesen Filter dadurch schlecht global einsetzen konnte. Sie ermöglichen zusätzliche Ränder, die in Filter-Effekte einbezogen werden.

3.8.8 Eigenschaften overlay und overlay-host

Mit den Eigenschaften **overlay** bzw. **overlay-host** kann gesteuert werden, ob Elemente bzw. Kind-Elemente und deren Kind-

Elemente auf der Zeichenfläche oben liegen sollen. Die Eigenschaft **overlay** kann die Werte **top** oder **none** (Standard) erhalten; die Eigenschaft **overlay-host** die Werte **true** oder **false** (Standard).

Listing 19: Eigenschaft overlay – paint12.svg

```

...
<g ...>
  <!-- Menü -->
  <g ...>
    <rect x="50" y="0" width="100" height="20"
      style="fill:#CCC; stroke:#000; stroke-width:2;"/>
    <text class="textklein" x="55" y="15">Menü</text>
    <path id="dreieck" d="M 125,5 L 145,5 L 135,15 Z"/>
  </g>

  <!-- Menü-Auswahl -->
  <g id="menue" ... display="none" style="overlay:top">
    <rect x="50" y="20" width="100" height="50"
      style="fill:#EEE; stroke:#000; stroke-width:2;"/>
    <text class="textklein" x="55" y="35">
      ...
    </text>
  </g>
  <!-- Sichtbarmachen der Menü-Auswahl -->
  <set xlink:href="#menue" attributeName="display"
    to="inline" begin="dreieck.mousedown"
    end="mouseup"/>
</g>

<!-- zu ueberdeckende Objekte -->
<g ...>
  <rect x="0" y="0" width="150" height="20"
    style="fill:#FFF; stroke:#000; stroke-width:2;"/>
  <text class="textklein" x="33" y="15">
    zu überdecken</text>
</g>
...

```

Listing 19 zeigt die Anwendung der Eigenschaft **overlay**, wobei ein Menü simuliert ist, dessen Menü-Auswahl durch Drücken der Maustaste sichtbar wird. Die Auswahl soll darunter liegende Objekte überdecken.

3.8.9 Cursor mit SVG-Mitteln

Bisher war für einen Cursor das Element **cursor** und eine png-Grafik zu verwenden. Nun kann ein Cursor mit SVG-Mitteln beschrieben werden. Zu den bisherigen Attributen **x**, **y**, **xlink:href** des Elements **cursor** kommen die Attribute **width**, **height**, **viewBox** und **preserveAspectRatio** hinzu.

3.9 Audio und Video

Mit Version 1.2 werden die Elemente **audio** und **video** zum Einbinden von Audio- bzw. Video-Dateien verfügbar. Beide Elemente besitzen ein **xlink:href**-Attribut zum Spezifizieren der einzubindenden Datei und können SMIL-Attribute wie **volume**, **begin**, **repeatCount** usw. zur Steuerung der Audio- bzw. Video-Inhalte erhalten (Listing 20). Bedienelemente zur Steuerung können mit vorhandenen SVG-Mitteln vom Anwender nach Bedarf selbst kreiert werden. Dem **video**-Element sind zudem **x**-, **y**-, **width**- und **height**-Attribute zuzuweisen, um den Bereich der visuellen Ausgabe zu beschreiben.

Listing 20: Elemente audio und video – audiovid.svg

```
...
<audio xlink:href="audio.ogg" volume="0.5"
  type="audio/vorbis" begin="audiobutton.click"
  repeatCount="2"/>
<g id="audiobutton"><!-- Beschreibung --></g>
...
<video xlink:href="video.mpg" volume="0.5"
  type="video/mpeg" x="120" y="100" width="150"
  height="100" begin="videobutton.click"
  repeatCount="2"/>
<g id="videobutton"><!-- Beschreibung --></g>
...
```

Neu ist auch die Eigenschaft **audio-level** für **audio**-, **video**- und Container-Elemente. Ihr können numerische Werte zwischen **0** und **1** (Standard) zugewiesen werden. Mit ihrer Hilfe wird die Lautstärke eines Elements berechnet, die sich aus dem Produkt seines **audio-level**-Werts mit der Lautstärke des Eltern-Elements ergibt.

Welche Dateiformate unterstützt werden, ist derzeit noch nicht entschieden. Für Audio wird das Ogg-Format in Betracht gezogen.

Mit der Einführung von Audio und Video sind die Weichen in Richtung Multimedia gestellt. Somit bieten sich in Verbindung mit den anderen SVG-Features sehr vielfältige Möglichkeiten, um SVG für multimediale (Offline-)Anwendungen einzusetzen. Hier sind Anwendungen in den Bereichen Unterhaltung, Werbung oder Schulung denkbar.

3.10 Bereich Animation

3.10.1 Element animation

Mit SVG 1.2 wird das Element **animation** eingeführt. Es spezifiziert ein SVG-Dokument oder ein SVG-Dokument-Fragment mit animierten Vektor-Grafiken. Es stellt ein sog. *SMIL Media Element* dar und definiert damit seine eigene *timeline*.

Als Attribute stehen **xlink:href** zum Verweisen auf zu animierbaren Inhalt; **x**, **y**, **width**, **height** zum Definieren der Position bzw. der Größe zur Verfügung. Zusätzlich können die üblichen SMIL-Attribute wie **begin**, **dur**, **repeatCount** zum Steuern des zeitlichen Ablaufs verwendet werden.

Listing 21 zeigt beispielhaft, wie das Element **animation** angewendet werden kann, um eine im **defs**-Bereich hinterlegte Animation aufzurufen und den (eigenen) Ablauf zu steuern:

Listing 21: Element animation

```
...
<defs>
  <rect id="animrecht" x="0" y="0" width="100"
    height="50" fill="#CCC" stroke="#000">
    <animate attributeName="fill"
      values="#CCC;#000;#CCC" keyTimes="0;1;2"
      begin="0" dur="2" fill="freeze"/>
  </rect>
</defs>
...
<animation x="30" y="90" width="100" height="50"
  begin="2" dur="2" repeatCount="3" fill="freeze"
  xlink:href="#animrecht"/>
...
<animation ...
  xlink:href="#animrecht"/>
...
```

Mit Hilfe von **animation** beginnt die Animation eines Rechtecks, dessen Füllung von grau nach schwarz und wieder zurück nach grau wechselt, nach zwei Sekunden. Die Animation wird drei mal ausgeführt. Hier wäre es nun möglich, die Rechteck-Animation auch an anderen Stellen im SVG-Dokument aufzurufen und den zeitlichen Ablauf mit anderen Attributwerten zu modifizieren.

3.10.2 Element transition

Neu in SVG 1.2 sind auch die von Präsentations-Programmen her bekannten Übergangs-Effekte (*transitions*). Sie können auf die Elemente **image**, **video**, **use**, **page**, **feImage** angewendet werden, in Verbindung mit Animationen des **xlink:href**-Attributs.

Für die Definition von Übergangs-Effekten steht das Element **transition** zur Verfügung, das eine Klasse von Übergangs-Effekten spezifiziert. Damit ist es auch möglich, **transition**-Elemente untereinander zu verschachteln.

Als Attribute stehen zur Verfügung:

- **type** (unbedingt erforderlich) und **subtype**: bestimmen den Typ und den Subtyp des Übergangs-Effekts; stammen aus dem SMIL 2.0 Transition Effects Module³⁰,
- **dur**: Dauer des Übergangs; Wert in Sekunden,
- **startProgress**: Fortschritt zu Beginn; Werte: **0.0** (Standard) bis **1.0**,
- **endProgress**: Fortschritt zu Ende; Werte: **0.0** bis **1.0** (Standard),
- **direction**: Richtung der Bewegung; Werte: **forward** (Standard), **reverse**,
- **fadeColor**: spezifiziert Start- bzw. End-Farb-Werte; nur für Typ **fade** mit Subtyp **fadeFromColor** bzw. **fadeToColor**.

3.10.3 Attribute **transIn** und **transOut**

Um die mit **transition** definierten Übergangs-Effekte anzuwenden, stehen die Attribute **transIn** und **transOut** zur Verfügung. Sie werden animierten Elementen hinzugefügt, bei denen ein Übergangs-Effekt stattfinden soll und referenzieren die Übergangs-Effekte. Bei **transIn** finden die Übergänge am Start der Animation statt; bei **transOut** finden die Übergänge am Ende der Animation statt.

3.10.4 Beispiel für Übergangs-Effekte

In Listing 22 sind im **defs**-Bereich zwei Übergangs-Effekte hinterlegt. Sie werden auf ein mit **set** animiertes **use**-Element angewendet, wobei die Referenz auf die im **defs**-Bereich hinterlegten Rechtecke ausgetauscht wird. Dieses Austauschen stellt die Animation dar, bei deren Start (**transIn**) die **transition**-Elemente wirken und damit die Übergänge ausgeführt werden.

³⁰ Werte siehe [SMIL20REC], Transition Effects Module, Abschnitt 12.8 Appendix: Taxonomy Tables

Listing 22: Übergangs-Effekte – anim12.svg

```

...
<defs>
  <rect id="rechtw" x="0" y="0" width="50" height="50"
    style="fill:#FFF; stroke:#000; stroke-width:2;
    cursor:pointer"/>
  <rect id="rechtg" x="0" y="0" width="50" height="50"
    style="fill:#999; stroke:#000; stroke-width:2;
    cursor:pointer"/>
  <transition id="bar" type="barWipe"
    subType="leftToRight" direction="forward"
    dur="3s"/>
  <transition id="rectangle" type="irisWipe"
    subType="rectangle" direction="reverse" dur="3s"/>
</defs>

<g transform="translate(30,80)">
  <use xlink:href="#rechtw" x="0" y="10">
    <set attributeName="xlink:href" to="#rechtg"
      begin="click" transIn="bar"/>
  </use>
</g>
...
<g transform="translate(135,80)">
  <use xlink:href="#rechtw" x="0" y="10">
    <set attributeName="xlink:href" to="#rechtg"
      begin="click" transIn="rectangle"/>
  </use>
</g>
...

```

3.11 Bereich Navigation

3.11.1 Erweiterte Links

Bisher unterstützte SVG mit dem Element **a** einfache Links, die jeweils zu einem Linkziel führen konnten. Mit SVG 1.2 werden erweiterte Links (*Extended Links*) möglich, die auf mehrere Linkziele verweisen können. Dafür werden das Element **xa**, das dem Element **a** entspricht, und das Element **loc** (*locator*), das die einzelnen Verweisziele enthält, eingeführt.

Wird ein Verweisziel aktiviert, erscheint ein Menü, mit dem die einzelnen Verweisziele ausgewählt werden können. Der (Text-)Inhalt

der Menü-Einträge entspricht dem des Elements `title`, als Kind von `loc`.

Listing 23: Erweiterte Links

```

...
<xa>
  <loc xlink:href="http://example.org/bsp/illustr.svg">
    <switch>
      <title>technische Illustration</title>
      <title systemLanguage="en">
        technical illustration</title>
    </switch>
  </loc>

  <loc xlink:href="http://example.org/bsp/teile.svg">
    <switch>
      <title>Teileliste</title>
      <title systemLanguage="en">list of parts</title>
    </switch>
  </loc>

  <!-- Link-Inhalt -->

</xa>
...

```

Listing 23 zeigt die Anwendung eines erweiterten Links. Der Link enthält zwei Verweisziele, die jeweils zu einer Illustration oder einer Teile-Liste führen. Zusätzlich dient das Element `switch` dazu, die Sprache des Betriebssystems zu identifizieren und wenn diese Englisch ist, die Menü-Einträge in englischer Sprache zu präsentieren.

3.11.2 Fokus und die Eigenschaft `focusable`

Mit dem sog. *Fokus* können Elemente angewählt werden (z. B. Hyperlinks per Tab-Taste), um Tastatur-Eingaben (z. B. Return-Taste) zu empfangen. In SVG 1.2 wird zum Bestimmen der entsprechenden Elemente die Eigenschaft `focusable` eingeführt. Sie kann für alle renderbaren Elemente, inklusive Grafik- und Container-Elemente sowie für sXBL-Komponenten verwendet werden. Mögliche Werte dafür sind `true`, `false`, `auto`. Bei `true` ist das Element anwählbar, bei `false` nicht. Der Wert `auto` entspricht standardmäßig `false`.

Bei einigen ausgewählten Elementen entspricht **auto** jedoch **true**:

- Element **a** (Hyperlink),
- Elemente **text** und **flowDiv**, wenn dort **editable="true"**.

3.11.3 Navigations-Reihenfolge

Zum Bestimmen der Navigations-Reihenfolge stehen die Eigenschaften **nav-index**, **nav-left**, **nav-right**, **nav-up**, **nav-down** aus [CSS3uiCR] zur Verfügung. SVG erlaubt dabei eine »flache« Navigation zwischen Elementen, d. h. dass die Navigations-Reihenfolge unabhängig von der Hierarchie der Elemente in einem SVG-Dokument festgelegt werden kann.

3.11.4 Tooltips

Bisher war in SVG empfohlen, kleine Hilfetexte zur Orientierung mit dem Element **title** und durch Scripte zu realisieren. Dafür werden nun in SVG 1.2 die Eigenschaft **tooltip** und das Element **hint** eingeführt, sodass für Tooltips nicht mehr auf Scripting zurückgegriffen werden muss.

Die Eigenschaft **tooltip** gibt dabei an, ob Tooltips angezeigt werden sollen. Dazu dienen die Werte **enable** für ja, **disable** für nein, **inherit** für vererbt. Der Inhalt des entsprechenden Tooltips wird im **hint**-Element hinterlegt. Im Unterschied zum **title** und **desc**-Element soll das **hint**-Element Instruktionen zur Bedienung des Elements beinhalten, auf das sich **hint** bezieht.

3.11.5 Beispiel für den Bereich Navigation

In Listing 24 sind die meisten der zuvor beschriebenen Features im Bereich Navigation zusammenhängend angewendet.

Listing 24: Navigation in SVG 1.2 – navi12.svg

```
...
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
      xmlns:xlink="http://www.w3.org/1999/xlink">
<title>Navigation (SVG 1.2)</title>
...
  <a xlink:href=""
    style="focusable:true; nav-index:2">
    <text ... x="34" y="20"> Link 1 - style="focusable:true;
      nav-index:2"
    ...
    </text>
  </a>
  <a xlink:href=""
    style="focusable:false">
    <text ... x="35" y="36">Link 2 - style="focusable:false"
    ...
    </text>
  </a>
  <a xlink:href=""
    style="focusable:auto; nav-index:1">
    <text ... x="35" y="52">Link 3 - style="focusable:auto;
      nav-index:1"
    ...
    </text>
  </a>
  <a xlink:href=""
    style="focusable:true; nav-down:#link5">
    <text ... x="35" y="68">
      Link 4 - style="focusable:true; nav-down:#link5"
    ...
    </text>
  </a>
...
  <a xlink:href="" id="link5">
    <rect x="170" y="20" width="80" height="60" rx="5" ry="5"
      style="tooltip:enable; pointer-events:stroke; fill:#EEE;
      stroke:#CCC; stroke-width:20">
      <hint>Hierauf klicken, um zu ... zu gelangen</hint>
      <set attributeName="opacity" begin="mouseover" to=".4"/>
      <set attributeName="opacity" begin="mouseout" to="1"/>
    </rect>
    <text ... x="192" y="25" style="pointer-events:none">
      Link 5
    </text>
  </a>
...
</svg>
```

3.12 Streaming

3.12.1 Zielsetzung

Mit SVG 1.2 wird *Streaming* möglich. Damit können SVG-Dokumente bereits angezeigt werden, wenn noch nicht alle zum Dokument gehörenden Daten übertragen worden sind. Die restlichen Daten strömen nach, währenddessen das Dokument bereits sichtbar vorliegt.

Im Print-Bereich lässt dies den Einsatz von Druckern zu, die über geringen Speicher verfügen. So sollen diese Drucker Dokument-Teile drucken und nachdem dies geschehen ist, die entsprechenden Daten aus ihrem Speicher entfernen. Zu diesem Zeitpunkt treffen die Daten des nächsten Dokument-Teils ein und der Druck-Prozess kann fortgesetzt werden.

Im Bereich Bildschirm-Anwendungen für das Web soll mit Streaming erreicht werden, dass Zeit-basierte Elemente wie Animationen bereits beginnen können, während die restlichen Daten noch heruntergeladen werden.

Für die Zielsetzung im Print-Bereich führt SVG 1.2 das Attribut **streamedContents** ein; für die Zielsetzung im Bildschirm-Bereich das Attribut **timelineBegin**, was nicht bedeutet, dass **streamedContents** im Bildschirm-Bereich keine Anwendung finden würde.

3.12.2 Attribut **streamedContents**

Das Attribut **streamedContents** erlaubt einem SVG-Anwender die Inhalte eines Dokuments dahingehend zu kennzeichnen, ob sie verfügbar bleiben sollen oder aus dem Speicher entfernt werden können. Der Wert **discard** kennzeichnet die Inhalte als entfernbar, während der Wert **keep** zu erkennen gibt, dass die Inhalte im Speicher verbleiben sollen. Dieses Attribut ist nur für das **svg**-Element erlaubt.

3.12.3 Attribut `timelineBegin`

Mit dem Attribut `timelineBegin` kann ein SVG-Anwender den Beginn der sog. *timeline* von bestimmten Elementen für Animationen (*time container*) festsetzen.³¹ In SVG 1.2 sind das die Elemente `page` und `pageSet` (siehe Abschnitt Multiple Seiten).

Bei `timelineBegin="onStart"` startet die *timeline*, wenn das öffnende Tag eines Elements verarbeitet worden ist. Der Wert `onStart` bietet sich damit für das `svg`-Tag bei gestreamten Animationen an, da das `svg`-Tag die globale *timeline* steuert, auf deren Grundlage interne *timelines* beginnen.

Bei `timelineBegin="onLoad"` startet die *timeline*, wenn das jeweilige Element voll verarbeitet wurde, d. h. das *SVGLoad-Event* für das jeweilige Element auftritt. Der Wert `onLoad` bietet sich damit bei `page`-Elementen für Animations-Szenen an, um sicherzustellen, dass die gesamte Szene geladen ist, bevor die *timeline* beginnt.

3.13 Schrittweises Rendern

Unter der Bezeichnung *Progressiv Rendering* werden in SVG 1.2 zweierlei Neuerungen eingeführt. Erstens werden Regeln dafür aufgestellt, wie SVG-Programme die Elemente in SVG-Dokumenten schrittweise rendern sollen. Zweitens wird auf dieser Basis geschildert, wie der SVG-Anwender das schrittweise Rendern steuern kann. Anhand von Listing 25 soll dies hier erklärt werden.

³¹ nähere Informationen dazu in [SMIL20REC] Abschnitt 10.3.2 Elements

Listing 25: Schrittweises Rendern – renprog.svg

```
1 ...
2 <g externalResourcesRequired="true">
3 <text class="mehrzeiler" x="50" y="80">
4   <tspan x="50" dy="1.2em">Zeile 1</tspan>
5   <tspan x="50" dy="1.2em">Zeile 2</tspan>
6   <tspan x="50" dy="1.2em">Zeile 3</tspan>
7 </text>
8 <image x="50" y="160" width="120" height="65"
9   externalResourcesRequired="true"
10  xlink:href="bild.svg"/>
11 <text class="mehrzeiler" x="30" y="250">
12   <tspan x="50" dy="0">Zeile 4</tspan>
13 </text>
14 </g>
15 ...
16 <use xlink:href="#recht2" x="0" y="-100"/>
17 ...
18 <rect id="recht1" x="292" y="140" width="100"
19   height="40" style="fill:none; stroke:#000;
20   stroke-width:2"/>
21 ...
22 <rect id="recht2" x="292" y="190" width="100"
23   height="40" style="fill:none; stroke:#000;
24   stroke-width:2"/>
25 ...
```

Die erste Möglichkeit ist die Bildung von Gruppen, denen das Attribut **externalResourcesRequired** mit dem Wert **true** zugewiesen wird (Zeile 2). Damit werden die gruppierten Inhalte erst gerendert, wenn alle Inhalte der Gruppe verarbeitet werden konnten und das schließende **g**-Tag (Zeile 14) *geparst* wurde. Eine weitere Möglichkeit ist die Verwendung des Elements **use**. Damit kann das Rendern einer Form an die Verarbeitung des Quell-Elements gekoppelt werden. In Listing 25 ist das Rendern einer Rechteck-Form in Zeile 16 an die Verarbeitung von **recht2** (Zeile 22) gekoppelt. Somit wird die Form erst gerendert, wenn **recht2** verarbeitet worden ist.

In Listing 25 würde also zunächst die Gruppe gerendert werden, wenn die Verarbeitung bis Zeile 14 erfolgreich abgeschlossen ist. Danach würde **recht1** (Zeile 18) gerendert werden. Danach **recht2** (Zeile 22) und erst zuletzt das **use**-Element (Zeile 16).

3.14 Auflösungsabhängige Bild-Inhalte

3.14.1 Element **multimage**

Das Element **multiImage** soll verschiedene Bild-Inhalte anzeigen, die abhängig von der (Display-)Auflösung automatisch vom SVG-Programm ausgewählt werden sollen. Dabei sollen der aktuelle Zoom und die angezeigte Größe des Multi-Image berücksichtigt werden.

3.14.2 Element **subImageRef**

Das Element **subImageRef** referenziert alternative Bilder, wobei durch seine Attribute **min-pixel-size** und **max-pixel-size** Pixelgrößen definiert werden. Als Bilder sind png-, jpeg- und svg-Dateien sowie **symbol**-Elemente in svg-Dateien erlaubt.

3.14.3 Element **subImage**

Das Element **subImage** ist ein Container-Element und enthält alternative Bilder (Attribute **min-pixel-size** und **max-pixel-size**).

3.14.4 Beispiel für auflösungsabhängige Bild-Inhalte

Listing 26 zeigt ein Multi-Image, das ein Raster-Bild mit geringer Auflösung und ein Raster-Bild mit hoher Auflösung enthält. Beim Bild mit geringer Auflösung ist die minimale Pixelgröße mit 1 angegeben, während beim Bild mit höherer Auflösung die maximale Pixelgröße mit 1 angegeben ist. Ein SVG-Programm soll nun z. B. für PDAs das Element mit **min-pixel-size="1"** auswählen und anzeigen, während z. B. für Notebooks das Element mit **max-pixel-size="1"** ausgewählt und angezeigt werden soll.

Listing 26: Auflösungsabhängige Bild-Inhalte

```
...
<multiImage x="0" y="0" width="180" height="180">
  <subImageRef xlink:href="aufl_gering.jpg"
    min-pixel-size="1"/>
  <subImage max-pixel-size="1">
    <svg width="100%" height="100%"
      viewBox="0 0 360 360">
      <image width="180" height="180"
        xlink:href="aufl_hoch.jpg"/>
    </svg>
  </subImage>
</multiImage>
...
```

3.15 Technische Details und Optimierung

3.15.1 Element `deviceColor`

Das Element **`deviceColor`** dient hauptsächlich zum Verbessern der Druckqualität bei Print-Anwendungen in bestimmten Workflows. Damit kann einer speziellen Farbe eine alternative, Gerätespezifische Farbe zugeordnet werden. Als Attribute sind

- **`xlink:href`**, das per URI auf eine Gerätespezifisches Farbprofil verweist und
- **`name`**, das den Namen für das Farbprofil erhält, möglich.

Listing 27 zeigt die Anwendung des Elements **`deviceColor`**.

Listing 27: Element deviceColor

```
...
<svg xmlns="http://www.w3.org/2000/svg" version="1.2"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns:bsp="http://www.example.org/bsp/druckerx">

<defs>
  <deviceColor name="druckerx"
    xlink:href="http://www.example.org/bsp/druckerx"
    bsp:value="farbname1, farbname2, farbname3, ..." />
</defs>

<rect x="50" y="80" width="50" height="50"
      fill="rgb(204,204,204)
      device-color(druckerx, wert1, wert2, wert3, ...)"/>

</svg>
```

3.15.2 Element prefetch

Das Element **prefetch** dient zum Vorladen von Ressourcen und zum Abstimmen anderer Größen, damit die Ressourcen reibungslos und störungsfrei wiedergegeben werden können. Als Attribute sind möglich:

- **mediaSize:**
vorzuladende Menge an Daten; Zahlenwert in bytes bei Ressource im selben Dokument oder Prozentangabe 0 bis 100% (Standard),
- **mediaTime:**
vorzuladende Menge in Bezug zur Dauer; Zeit- oder Prozentangabe 0 bis 100% (Standard),
- **bandwidth:**
Bandbreite; Zahlenwert oder Prozentangabe 0 bis 100% (Standard),
- **mediaCharacterEncoding:**
Zeichen-Kodierung von **mediaSize**; Werte **UTF-8**, **ISO-8859-1**, ...,

- **mediaContentEncodings**:
Kodierung des Inhalts von **mediaSize**; durch Leerzeichen getrennte Liste (...="gzip compress ...").

Listing 28 zeigt ansatzweise die Anwendung von **prefetch**.

Listing 28: Element prefetch

```

...
<defs>
  <prefetch xlink:href="#seite1" mediaTime="10s"
    mediaCharacterEncoding="ISO-8859-1"
    mediaSize="73"/>
  <prefetch xlink:href="#seite2" mediaTime="10s"
    mediaCharacterEncoding="ISO-8859-1"
    mediaSize="113"/>
  ...
</defs>

<pageSet>
  <page id="seite1" begin="0s;seiteX.end"
    end="seite1.click">
    <!-- Seiten-Inhalte -->
  </page>
  <page id="seite2" begin="seite1.end"
    end="seite2.click">
    <!-- Seiten-Inhalte -->
  </page>
  ...
</pageSet>
...

```

3.15.3 Element switch als Kind-Element

Mit SVG 1.2 kann das Element **switch** Kind-Element von jedem SVG-Element sein.

3.15.4 Attribut **requiredFormats**

Das neue Attribut **requiredFormats** definiert eine Liste von Formaten und deren Definitionen (z. B. mit Hilfe des MIME-Typs), die vom SVG-Programm unterstützt werden müssen, damit es ein Dokument(-Fragment) verarbeitet.

3.15.5 Häufigkeit von grafischen Elementen

Die Eigenschaften **cache**, **static** und **snap** können bei grafischen Elementen verwendet werden, um mit dem SVG-Programm abzustimmen:

- wie oft ein Element voraussichtlich gezeichnet wird (**cache**),
- ob ein Element voraussichtlich oft modifiziert angewendet wird (**static**),
- ob ein Element an verschiedenen Stellen im Dokument bzw. in verschiedenen Displays oft identisch angezeigt wird (**snap**).

3.15.6 Attribut **snapshotTime**

Das Attribut **snapshotTime** teilt dem SVG-Programm mit, wann der beste Zeitpunkt für das Rendern eines unbewegten Bildes ist, wenn Animationen vorhanden sind. Das könnte z. B. für eine Vorschau-Funktion des SVG-Programms nützlich sein. Das Attribut ist nur für das **svg**-Wurzel-Element erlaubt.

3.16 Nicht-grafische Erweiterungen

3.16.1 XLink-Attribut

Mit SVG 1.2 kann das Attribut **xlink:href** bei den Elementen **title**, **desc**, **metadata** verwendet werden. Damit wird es für diese Elemente möglich, auf externen Inhalt zuzugreifen.

3.16.2 Element **metadata**

Mit dem Element **metadata** können eindeutige Angaben zu Lizenzierungs-Angelegenheiten eines SVG-Dokuments gemacht

werden sowie mit diesem verknüpft werden. Dabei bezieht sich ein **metadata**-Element im äußersten **svg**-Element auf das gesamte Dokument. Ein **metadata**-Element kann aber auch für spezielle Dokument-Teile eingesetzt werden, die eine spezielle Lizenz besitzen (z. B. eingebettete Fonts).

Vorgeschlagen für die Angaben ist das »Creative Commons Metadata Set«, das Angaben (**cc:permits**, **cc:requires**, **cc:prohibits**) zur Vervielfältigung, Verteilung, kommerziellen Nutzung usw. des SVG-Dokuments erlaubt. Angaben können aber auch von gewünschten Quellen her bezogen werden, wie Listing 29 zeigt.

Listing 29: Element metadata

```
...
<metadata>
  <rdf:RDF xmlns:cc="http://web.resource.org/cc/"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    <cc:work rdf:about="">
      <cc:license
        rdf:resource="http://www.example.org/copyright"/>
      </cc:work>
    </rdf:RDF>
</metadata>
...
```

3.17 Scripting und DOM

3.17.1 Elemente script und handler

Zum Verwirklichen von Dynamik und Interaktivität durch Scripting stehen in SVG 1.2 die Elemente **script** und **handler** zur Verfügung. Das Element **script** gibt es seit SVG 1.0 (siehe SVG-Grundlagen, Abschnitt Scripting). Beide Elemente dienen als Ablage für ausführbaren Inhalt bzw. zum Zugreifen auf ausführbaren Inhalt. Ausführbarer Inhalt können Skripte (Text-Code) oder kompilierter Code sein.

3.17.2 Neuerungen bei script und handler

Mit **handler** wird in SVG 1.2 ein Element eingeführt, das einen ähnlichen Zweck wie das **script**-Element erfüllt. Das **handler**-Element hat aber zudem noch weitere Aufgaben und unterscheidet sich in einigen Details und Neuerungen vom **script**-Element.

Neu ist das Attribut **scriptContentType** für den Typ des auszuführenden Inhalts. Es kann eingesetzt werden, wenn ein Script innerhalb eines **handler**-Elements abgelegt wird. In diesem Fall ist als Wert **text/ecmascript** anzugeben:

```
...
<handler scriptContentType="text/ecmascript">
  <!-- Script-Inhalte -->
</handler>
....
```

Das Attribut **scriptContentType** ist nicht zulässig, wenn **xlink:href** als Attribut von **handler** eingesetzt wird, um auf ausführbaren Inhalt zu verweisen. Dabei darf **xlink:href** nur auf lokale **script**-Elemente verweisen. Diese **script**-Elemente besitzen mit **type** oder **scriptContentType** selbst ein Attribut, das den Typ des ausführbaren Inhalts spezifiziert:³²

```
...
<handler id="handler1" xlink:href="#script1"/>
<script id="script1" type="text/ecmascript">
  <![CDATA[ /* Script-Inhalte */ ]>
</script>
...
<handler id="handler2" xlink:href="#script2"/>
<script id="script2" xlink:href="..."
scriptContentType="application/java-archive"/>
....
```

³² dass das Element **script** ein Attribut namens **scriptContentType** besitzen kann, ist rein hypothetisch, da in [SVG12WD8] Abschnitt 16.1 The script element keine Angaben zum entsprechenden Attributnamen gemacht werden

Wie zuvor ersichtlich, kann dem Attribut **scriptContentType** auch der Wert **application/java-archive** zugewiesen werden. Das ist dann der Fall, wenn kompilierter (Java-)Code ausgeführt werden soll. Dieser muss in einer externen Ressource vorliegen. Diese wiederum kann per **xlink:href** im **script**-Element referenziert werden.

Der eigentliche Fortschritt, der mit dem Element **handler** erzielt werden soll, ist die Möglichkeit, unter Verwendung des *XML Events*-Standards an (SVG-)Elemente *Event-Listener* und dazugehörige *Event-Handler* anzuhängen.³³

3.17.3 Beispiel für Element handler

In Listing 30 wird unter Verwendung des XML Events-Namensraums (Zeile 3) ab Zeile 9 ein *Event-Listener* definiert. Dieser kann Mausklicks auf die Umrisslinie des Rechtecks **recht1** wahrnehmen. Zusätzlich wird dem *Event-Listener* ein *Event-Handler* namens **EigenerHandler** zugeordnet.

Dieser eigene *Event-Handler* und das Script, das ausgeführt werden soll, sind als Element **handler** ab Zeile 12 aufgeführt. Bei Mausklick auf die Umrisslinie des Rechtecks **recht1** soll sich die Strichbreite **stroke-width** um 1 Pixel verringern.

³³ siehe auch [XMLEventsREC]

Listing 30: Element handler – script12.svg

```

1  ...
2  <svg xmlns="http://www.w3.org/2000/svg" version="1.2"
3      xmlns:ev="http://www.w3.org/2001/xml-events">
4
5  <rect id="recht1" x="80" y="80" width="80" height="60"
6      rx="5" ry="5" style="pointer-events:stroke;
7      fill:#EEE; stroke:#CCC; stroke-width:20"/>
8
9  <ev:listener ev:event="click" ev:observer="recht1"
10     ev:handler="#EigenerHandler"/>
11
12 <handler id="EigenerHandler"
13     scriptContentType="text/ecmascript">
14     var recht, strich;
15     recht=document.getElementById("recht1");
16     strich=parseFloat(recht.getPropertyValue("stroke-width"));
17     recht.setProperty("stroke-width", (strich-1));
18 </handler>
19
20 </svg>

```

3.17.4 DOM-Erweiterungen

Das Document Object Model (DOM) von SVG 1.2 baut auf dem DOM Level 3 auf und unterstützt dabei die Teilbereiche Core, Events, XPath. In allen anderen Bereichen entspricht es dem DOM von SVG 1.1.

Außerdem werden mit SVG 1.2 die folgenden Neuerungen eingeführt:

- erweiterter Satz an Interfaces für Media-Elemente,
- neue Methode für das Umwandeln von Koordinaten im Interface *SVGLocatable*,
- Interface *SVGEventFilter*, damit Event Listener nicht alle Events empfangen, sondern nur die wirklich nötigen,
- Aktualisierung der **getScreenCTM()**-Methode zum Zurückgeben der Transformations-Matrix vom aktuellen Benutzer-Koordinatensystem zum Screen-Koordinatensystem,

- neue Methode **getClientCTM()** im Interface *SVGLocatable* zum Zurückgeben der Transformations-Matrix vom aktuellen Benutzer-Koordinatensystem zum Client-Koordinatensystem,
- *Wheel Event* für Eingabegeräte mit Bedienelementen zum Drehen,
- Methoden **getResultBBox()**, **getRegionBBox()** im Interface *SVGLocatable* zum Zurückgeben der Objekt-umgebenden Box,
- Event *Shape Change* und Event *RenderedBBoxChange* zum Melden der Veränderung einer Form,
- Event *Progress* zum Steuern der Anzeige von Inhalten abhängig vom Download-Fortschritt.

3.17.5 Erweiterte Programmier-Schnittstellen

- Interface *SVGTimer* (Zeitsteuerung),
- Interface *URLRequest* um über eine URI auf Daten zuzugreifen; zum Nachahmen der Methoden **getURL()** und **postURL()** vom Adobe SVG Viewer,
- Interface *Connection*,
- Interfaces für das Hochladen von Dateien (*File Upload*),
- Interface *Global*.

4 Zusammenfassung

Als XML-Dialekt bietet SVG von vornherein die Vorzüge einer mächtigen Informations- und Dokumentations-Technologie. Wesentlich ist dabei vor allem die Möglichkeit zur strukturierten Ablage von Daten bzw. Inhalten. In Kombination mit den Vorteilen, die Vektorgrafiken bieten, wurde durch das W3C mit den SVG-Versionen 1.0/1.1 ein flexibler Grafikstandard entwickelt, mit dem ansprechende und zugleich strukturierte Visualisierung möglich wurde.

Mit SVG 1.2 wird dieser Grafikstandard ein bedeutendes Stück weiterentwickelt. Die herausragendsten Neuentwicklungen sind hierbei Fließtext mit Bildern, die XML Binding Language von SVG (sXBL), multiple Seiten sowie erweiterte Vektor- und Rendering-Effekte.

Fließtext mit Bildern wird den Umgang mit längeren Texten wesentlich erleichtern und ist daher eine essentielle Verbesserung. Damit dienen Texte in SVG nicht mehr nur zum Bezeichnen und Erläutern von Grafiken, sondern können auch als integrierte, einzelne Informations-Einheiten bestehen. Somit erweitert SVG hier auch insgesamt sein Anwendungsgebiet, indem die Beschränkung auf Grafiken zugunsten kombinierter Text-/Bild-Anwendungen aufgehoben wird, was ein entscheidender Anreiz dafür sein könnte, SVG als alleiniges Werkzeug für Anwendungen im Bereich Web, Schulung, Technische Dokumentation u. Ä. zu verwenden. Zudem würde für diese Anwendungen möglich sein, Grafiken und Text zusammen in einem plattform- und anwendungsunabhängigen Format auszutauschen.

Mit der XML Binding Language von SVG (sXBL) wird ein sehr sinnvoller Erweiterungs-Mechanismus eingeführt. Auch hier zeigt sich das Bemühen nach größtmöglicher Flexibilität: Oft wiederkehrende Dokument-Bausteine (Komponenten) können einmal erstellt, in SVG-Dateien abgelegt und beliebig oft wiederverwendet werden; der dynamische Zugriff auf die Baumstruktur des Dokument-Bausteins durch Shadow Trees inbegriffen. Dadurch wird strukturierte Dokumentation auch auf Redaktions-Ebene möglich, da SVG-Anwendungen bzw. Informations-Angebote modular aufgebaut und zusammengestellt werden können, was sicherlich auch hinsichtlich von Erstellungs-Kosten interessant sein dürfte.

Mit den multiplen Seiten werden in SVG 1.2 Print-/Präsentations-Konzepte integriert, die das Handhaben von mehreren Seiten in einem SVG-Dokument erlauben. Dadurch entspricht der Umgang mit Inhalten bewährten Konzepten und wird mit SVG 1.2 komfortabler gestaltet. Die multiplen Seiten eignen sich zudem auch für sequentielle, Zeit-basierte Anwendungen wie Animationen.

Mit den neuen Vektor- und Rendering-Effekten erweitern sich die Möglichkeiten zum Zeichnen und Gestalten von grafischen Objekten. Damit werden vor allem Grafiker angesprochen, die eine sehr genaue Kontrolle über das Aussehen von grafischen Objekten beanspruchen. Entsprechend speziell sind die in diesen Bereichen eingeführten Neuerungen.

Ein relativ großer Teil an Neuerungen, die mit SVG 1.2 eingeführt werden, knüpft an die Möglichkeiten in SVG 1.0/1.1 an oder ergänzt sie sinnvoll. Diese Neuerungen sind relativ speziell; es handelt sich meist um einzelne Elemente, Attribute oder Eigenschaften. Besondere Aufmerksamkeit verdienen hierbei vielleicht die neuen Elemente **audio** und **video**, die SVG 1.2 in Zukunft möglicherweise für multimediale Anwendungen interessant machen; das Element **transition** und die Attribute **transIn** und **transOut** für Übergangs-Effekte zwischen mehreren Seiten in einem SVG-Dokument; die erweiterten Links mit mehreren Verweiszielen oder auch das Element **prefetch** zum Vorladen von Ressourcen.

Die genannten Neuerungen zusammen mit den Möglichkeiten aus SVG 1.1 dürften SVG 1.2 zu einem flexiblen und vielseitigen Grafikstandard machen, dessen Anwendung weit über die Erstellung von Grafiken hinausgehen kann. Die sich mit SVG 1.2 bietende Funktionalität ist jedenfalls größer denn je; sie lässt Anwendungen in vielerlei Bereichen zu. Egal, ob allein eingesetzt oder in Kombination mit anderen Web-Technologien, mit Version 1.2 dürfte SVG vielen Anforderungen gerecht werden. Dabei ist zu beobachten, dass es sich immer mehr zu einem universell einsetzbaren Werkzeug für Informations-Vermittlung entwickelt.

Voraussetzung dafür, dass die Vorzüge von SVG 1.2 zur Geltung kommen können, sind aber entsprechende Programme zum Erstellen und Anzeigen, in welche die viel versprechenden Neue-

rungen implementiert werden. Für die Zukunft ist zu hoffen, dass dies so bald als möglich geschieht.

Mit der Vollendung des Recommendation Process voraussichtlich im Mai 2005 sollte das W3C die endgültige Basis für Implementierungen geschaffen haben: Laut Roadmap (vgl. Abbildung 1, S.5) tritt SVG 1.2 im Dezember 2004 mit der Candidate Recommendation in die Recommendation-Phase ein. Darauf soll im März 2005 die Proposed Recommendation erscheinen. Im Mai 2005 schließlich soll die finale Recommendation vorliegen und SVG 1.2 damit als offizieller Standard verabschiedet werden.

Print- und Online-Ressourcen

[ADAM02] ADAM, Alexander: SVG – Scalable Vector Graphics; das Praxisbuch. Poing: Franzis', 2002.

[ADAM04] ADAM, Alexander: Go with the Flow – Version 1.2 der SVG-Spezifikation des W3C. In: XML & Web Services Magazin 02/2004 (März 2004): S.34–36.

[ASV302] Adobe SVG Viewer 3.02. Download unter:
<http://www.adobe.com/svg/viewer/install/main.html>

[ASV60p1] Adobe SVG Viewer 6.0 Development Release 1. Download unter: <http://www.adobe.com/svg/viewer/install/beta.html>

[CSS3uiCR] W3C: CSS3 Basic User Interface Module Candidate Recommendation. URL:<http://www.w3.org/TR/2004/CR-css3-ui-20040511/> (11.05.2004)

[FIBI02] FIBINGER, Iris: SVG – Scalable Vector Graphics: Praxiswegweiser und Referenz für den neuen Vektorgrafikstandard. München: Markt+Technik, 2002.

[MEIN02] MEINIKE, Thomas: SVG – Learning By Coding. Beispielsammlung zu SVG. URL:<http://svglbc.datenverdrahten.de/> (02.11.2004)

[MEIN03] MEINIKE, Thomas: Mobile Vektoren – Animationen und Effekte mit SVG. In: Internet Professionell 06/2003 (Mai 2003): S.72–75.

[QUINT03] QUINT, Antoine: SVG and XForms: Rendering Custom Content. Using the new SVG 1.2 XML-based extension mechanism. URL:<http://www-106.ibm.com/developerworks/xml/library/x-svgxf2/> (25.11.2003)

[SELF80] MÜNZ, Stefan: SELFHTML 8.0. URL:<http://de.selfhtml.org/> (27.10.2001)

[SMIL20REC] W3C: Synchronized Multimedia Integration Language (SMIL 2.0) Recommendation. URL:<http://www.w3.org/TR/2001/REC-smil20-20010807/> (07.08.2001)

[SVG11REC] W3C: SVG 1.1 Recommendation. URL:<http://www.w3.org/TR/2003/REC-SVG11-20030114/> (14.01.2003)

[SVG12WD7] W3C: SVG 1.2 Working Draft. URL:<http://www.w3.org/TR/2004/WD-SVG12-20040510/> (10.05.2004)

[SVG12WD8] W3C: SVG 1.2 Working Draft. URL:<http://www.w3.org/TR/2004/WD-SVG12-20041027/> (27.10.2004)

[SVGRoad] W3C: SVG Roadmap. URL:<http://www.w3.org/Graphics/SVG/Roadmap> (24.11.2004)

[sXBLWD2] W3C: sXBL Working Draft.

URL:<http://www.w3.org/TR/2004/WD-sXBL-20041122/> (22.11.2004)

[WATT02] WATT, Andrew; LILLEY, Chris u.a.: SVG unleashed.

Indianapolis: Sams Publishing, 2002.

[XMLEventsREC] W3C: XML Events Recommendation.

URL:<http://www.w3.org/TR/2003/REC-xml-events-20031014/> (14.10.2003)