Einsatzmöglichkeiten von SVG zur Informationsvisualisierung auf mobilen Endgeräten

Studienarbeit

Eingereicht von:	Michael Zornow
Studiengang:	Informationstechnik
Matrikelnummer:	201671
Betreuer:	Prof. Dr. Heidrun Schumann
	Dipl.-Inf. Christian Tominski

Bearbeitungszeit:	von	01.03.2006
	bis	31.05.2006

Universität Rostock
Institut für Informatik
Albert-Einstein-Strasse 21, D-18051 Rostock

Abstract

In dieser Arbeit wird untersucht inwiefern sich das Vektorgrafikformat SVG (Scalable Vector Graphics) zur Visualisierung von Datenbeständen auf mobilen Endgeräten eignet. Es wird überprüft, wie sich die begrenzte Leistungsfähigkeit mobiler Endgeräte auf die Anzahl der noch interaktiv darstellbaren SVG Primitive auswirkt, das heißt wie komplex eine SVG-Darstellung werden darf. In diesem Zusammenhang spielen auch die Interaktionsmöglichkeiten von SVG eine Rolle. Des Weiteren wird überprüft welche Schnittstelle zwischen einer Datenbank und SVG zum Einsatz kommen kann (direkte Datenbankanbindung, Softwareschnittstelle). Diese Schnittstelle soll als Grundlage für die entwickelten Softwareprojekte dienen, die am Ende dieser Arbeit vorgestellt werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Die Visualisierung von Informationen ist ein wichtiges Gebiet in der Entwicklung und Forschung. Durch die geeignete Visualisierung eines abstrakten Sachverhaltes sind Menschen in der Lage relevante Informationen schneller und besser aufzunehmen, als dies durch eine verbale Beschreibung möglich wäre. Dabei ist die Visualisierung ein sehr komplexer Prozess, bei dem viele Einflussfaktoren zu berücksichtigen sind. In den vergangenen Jahren waren in der Visualisierung viele Entwicklungen zu beobachten. Eine dieser Entwicklungen ist der Versuch, Bilder mit Hilfe einer textbasierten Geometriebeschreibung darzustellen.

Grundlage zur Entwicklung einer textbasierten Geometriebeschreibung war unter anderem die Entwicklung des Standards SGML[1], mit der IBM bereits 1960 begann. Ziel von SGML war es, einem Text neben der optischen Gestaltung (Formatierung) auch eine inhaltliche (semantische) Struktur zu geben. SGML benutzt dazu bedeutungstragende Zeichenketten, die so genannten Tags oder auch Elemente. Mit Hilfe dieser Tags, welche von spitzen Klammern eingeschlossen sind, werden Texte umschlossen. SGML ist dabei generell frei von jeglicher plattformabhängiger oder ausgabespezifischer Formatierung, weil SGML-Dokumente ausschließlich mittels ASCII-Text erstellt werden.

Die eXtensible Markup Language (XML) wurde vom W3C[2] 1998 ins Leben gerufen. Sie entspricht zu großen Teilen der Syntax ihres Vorgängers SGML, wurde aber speziell für die Bedürfnisse des Internets optimiert [Fib02]. XML

---

[1] Standard Generalized Markup Language

[2] World Wide Web Consortium

ist, genau wie SGML, eine Metasprache, die es erlaubt weitere Sprachen zu definieren. Eine dieser abgeleiteten Sprachen, den so genannten Instanzensprachen von XML, ist SVG (Scalable Vector Graphics).
Bereits 1998 wurden erste Konzepte für SVG erarbeitet. Bis zum heutigen Tage wird die Sprache SVG weiter entwickelt. Dabei findet sie vielfache Anwendung. So wird sie als Austauschformat, für interaktive Benutzerschnittstellen, im Bereich des eCommerce oder auch für die Kartografie verwendet.
Ziel dieser Arbeit ist es zu überprüfen, inwiefern sich SVG zur Visualisierung von Datenbeständen auf mobilen Endgeräten eignet. Hier ergeben sich, aufgrund der begrenzten Leistungsfähigkeit und auch wegen der begrenzten Darstellbarkeit grafischer Primitive durch die geringe Displaygröße mobiler Geräte, spezielle Anforderungen an die Visualisierung.

Die Arbeit ist dazu folgendermaßen gegliedert:

Kapitel 2 charakterisiert SVG als Vektorgrafikformat und gibt eine kurze Einführung in für die Arbeit wichtige Begriffe. Es dient so als Grundlage für das Verständnis nachfolgender Kapitel. Abschließend erfolgt eine Einführung in die Interaktionsmöglichkeiten von SVG, weil sie ein wesentliches Hilfsmittel der Visualisierung darstellen.

Kapitel 3 beschreibt den Vektorgrafikstandard SVG speziell für mobile Geräte. Es werden zwei mobile Profile, SVG Tiny und SVG Basic, vorgestellt und verglichen.

Kapitel 4 untersucht, aufbauend auf die vorhergehenden Kapitel, die Möglichkeiten von SVG zur Visualisierung von Datenbeständen auf mobilen Geräten. Abschließend werden hier die Ergebnisse der Untersuchungen zusammengefasst. Es wird bewertet, ob SVG eine geeignete Möglichkeit zur Informationsvisualisierung für mobile Geräte darstellt.

Kapitel 5 stellt die im Rahmen dieser Arbeit implementierten Visualisierungsmöglichkeiten von SVG für mobile Geräte vor.

Kapitel 6 gibt eine Zusammenfassung der Ergebnisse dieser Arbeit, bevor am Ende ein kurzer Ausblick in die Zukunft von SVG diese Arbeit beenden soll.

Kapitel 2

SVG - Grundlagen und Begriffsbildung

SVG wurde vom W3C im Jahr 2001 als Vektorgrafikstandard verabschiedet. Momentan liegt SVG in der Version 1.1 als Recommendation vor. Die Version 1.2 ist zurzeit in Vorbereitung und befindet sich in der Standardisierungsphase des Working Draft seit April 2005.
Um untersuchen zu können, ob SVG ein geeignetes Visualisierungswerkzeug für Datenbestände auf mobilen Geräten darstellt, wird im folgenden Abschnitt SVG grundlegend charakterisiert. Es werden Begriffe eingeführt, welche für das Verständnis der weiteren Untersuchungen von Bedeutung sind. Im Abschnitt 2.2 erfolgt eine Einführung in die fundamentale Struktur eines SVG–Dokuments. Den Abschluss des Kapitels bilden die Vorstellung der Grundelemente und die Darstellung der Interaktionsmöglichkeiten von SVG.

2.1 Was ist SVG?

Immer, wenn es gilt abstrakte Daten in Bilder zu konvertieren, werden mehrere Stufen einer so genannten Visualisierungspipeline durchlaufen (vgl. u. a. [Sch00]). Dazu gehören:

- die Datenaufbereitung (Filtering)
- die Erzeugung eines Geometriemodells (Mapping) und
- die Bildgenerierung (Rendering)

SVG spezifiziert, auf welche Art und Weise eine Grafik durch geometrische Primitive beschrieben werden soll und definiert ein Referenzmodell für den Prozess der Bildgenerierung. Es unterstützt also die in Abbildung 2-1 dargestellte zweite und dritte Stufe der Visualisierungspipeline – das Mapping und das Rendering.

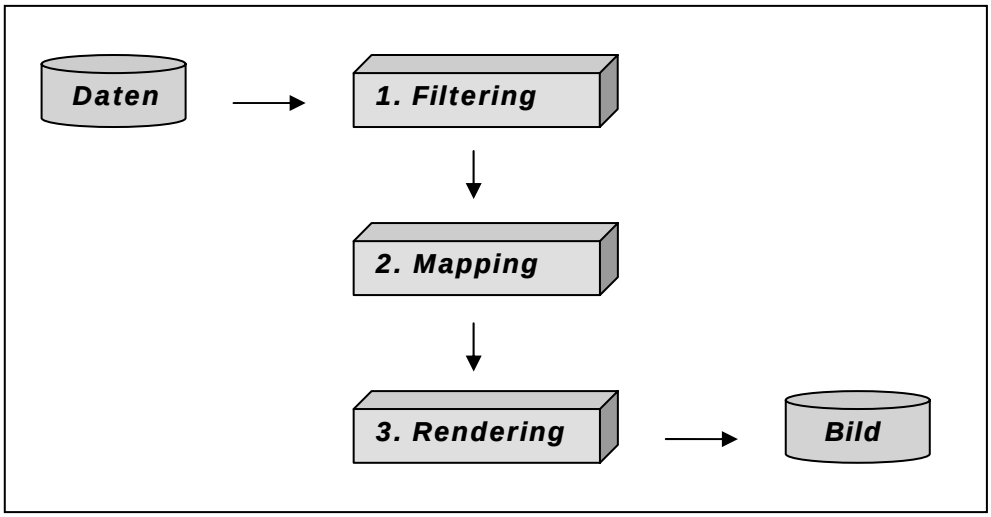

Abbildung 2-1: Die Visualisierungspipeline

Ziel der Entwicklung von SVG war es, möglichst plattform- und anwendungsunabhängig die Daten-zu-Geometrie-Abbildung (Mapping) vorzunehmen. Als eine von XML abgeleitete Sprache ist SVG dazu ebenfalls textbasiert.

Auch die Weiterverwendbarkeit für alle Formen der Ausgabe auf unterschiedlichen Medien (Print, Bildschirm, etc.) war ein wichtiges Kriterium bei der Spezifikation. Um dieses Designziel zu verwirklichen nutzt SVG Vektorgrafiken, damit skalierbare Grafiken beschrieben werden können. Im Gegensatz zu Rastergrafiken, welche Bildinformationen pixelweise enthalten, werden SVG–Vektorgrafiken aus geometrischen Grundformen zusammengesetzt. Um diese eindeutig zu beschreiben, reicht es aus, eine mathematische Beschreibung der Objekte zu definieren. Beispielsweise wird ein Kreis eindeutig über die Koordinaten seines Mittelpunktes und über seinen Radius bestimmt. Dieser Beschreibung hinzugefügt werden dann nur noch die Farbe und Form der Begrenzungslinie und eventuell die Farbe der Füllung des Kreises. Diese Beschreibungsform bietet einen weiteren großen Vorteil: SVG-Dokumente

haben bis zu einer bestimmten Anzahl an Primitiven eine geringere Größe als Rastergrafiken. Diese Dateigröße kann mit Hilfe des Kompressionsprogramms GZIP weiter verringert werden. Komprimierte SVG-Dokumente erhalten das Suffix *.svgz* – im Gegensatz zu Standard-SVG-Dokumenten die am Suffix *.svg* erkennbar sind.
Die Darstellung von SVG-Grafiken (Rendering) erfolgt durch so genannte User Agents, das heißt durch Browser, Browser–Plugins oder eigenständige SVG–Viewer. Diese orientieren sich dabei am Rendering Model, welches Teil der Spezifikation von SVG ist [Jac03].
Neben Vektorelementen erlaubt SVG auch die Einbindung von Rastergrafiken und Text als grafische Objekte. Sie werden im Abschnitt 2.3 näher beschrieben, nachdem nun zunächst die fundamentale Dokumentstruktur von SVG im Mittelpunkt stehen soll.

2.2 Fundamentale Dokumentstruktur von SVG

Wie bereits erwähnt ist SVG ein von XML abgeleitetes Sprachkonzept. Man spricht hier von einer Instanz von XML. Diese Instanzen zeichnen sich unter anderem dadurch aus, dass deren Aufgabe durch die jeweils gewählte Semantik der einzelnen Tags zum Ausdruck gebracht wird. XHTML[3] beispielsweise, ebenfalls eine XML-Instanz, ist dazu konzipiert Hypertext-Dokumente zu erstellen. Hier ist es von Bedeutung, den einzelnen Inhalten eine äußere Gestaltung (Formatierung) zu geben.
In SVG werden Tags zur Strukturierung eines Dokuments bzw. zur Beschreibung von Vektorelementen benutzt. Einzige Ausnahme bilden hier Server-Tags. Sie definieren Effekte wie Farbverläufe oder Filter, auf die andere Elemente später zugreifen können. Innerhalb der Tags werden Formatierungen von SVG-Dokumenten angegeben. Formatierungen werden in SVG über Attribute beschrieben. Ein Attribut unterteilt sich in zwei Bestandteile:

[3] Extensible Hypertext Markup Language

- den Attributnamen und
- den Attributwert

Der Wert wird durch Anführungsstriche eingeschlossen und dem Attributnamen durch ein Gleichheitsszeichen (=) zugewiesen (siehe zum Beispiel SVG-Wurzelelement mit Attributen in Abb. 2-2).

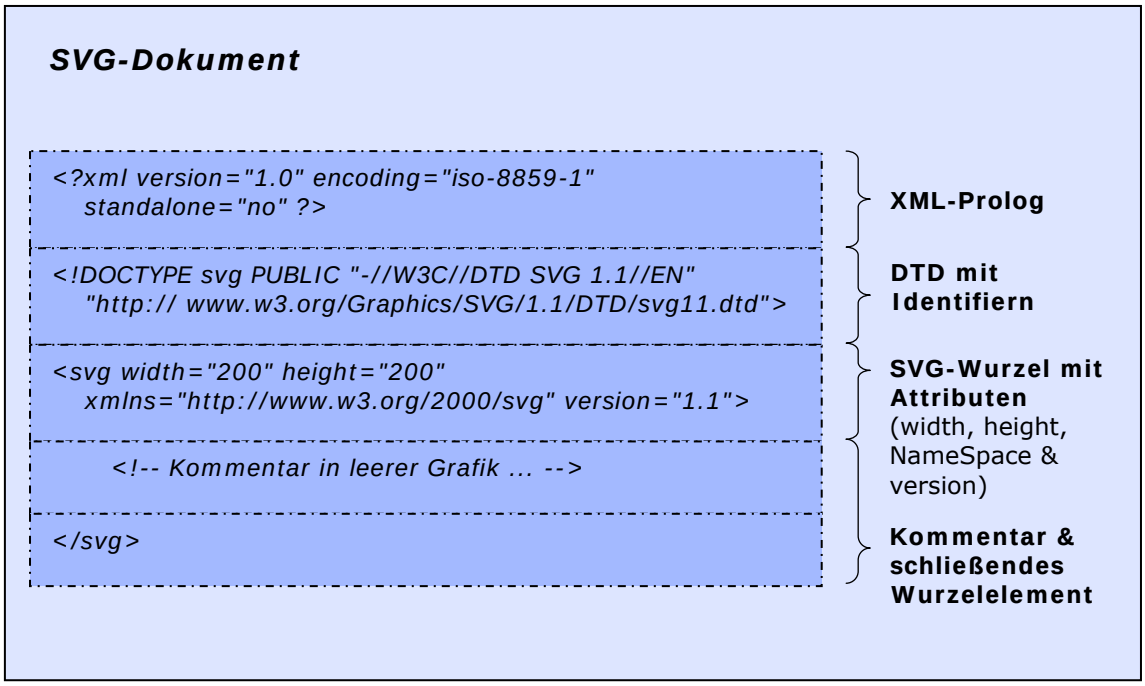

Abbildung 2-2: fundamentale Dokumentstruktur mit einem Kommentar als einzigem Inhalt der SVG-Grafik

Nachdem Tags (Elemente) und Attribute, als für eine SVG-Grafik wichtige Begriffe eingeführt wurden, soll jetzt die konkrete Struktur einer solchen Grafik beschrieben werden, da sie von tragender Bedeutung einer standardisierten Sprache wie SVG ist.

Abbildung 2-2 zeigt den nun beschriebenen Aufbau eines SVG–Dokuments, welches als einzigen Inhalt einen Kommentar enthält. Um eine SVG-Grafik als XML-Dokument auszuweisen, beginnt jedes SVG-Dokument grundsätzlich mit dem XML-Prolog. Dieser lautet für alle Sprachkonzepte gleich. Neben einer Nummer, welche die jeweilige Version von XML spezifiziert, können hier weitere Attribute enthalten sein. Ein Attribut des XML-Prologs beispielsweise kann den verwendeten Zeichensatz festlegen. Erfolgt hier keine Angabe, verwendet der jeweilige User Agent den Default-

Zeichensatz UTF-8[4]. Die darauf folgende Zeile ist die Dokument-Typ-Definition. Eine Dokument-Typ-Definition (DTD) ist eine Deklaration, welche die Struktur eines Dokuments bestimmt. Sie bildet die Grundlage für die Schaffung einer gültigen Dokumentstruktur. Mit dieser wird es möglich, dass das SVG-Dokument von einem Programm eingelesen werden kann. Das Programm kann die enthaltenen Daten nur verarbeiten, wenn es weiß wie das Dokument aufgebaut ist. In der DTD sind sowohl jedes Element und jedes Attribut festgelegt, als auch Verschachtelungsregeln der Elemente und weitere Eigenheiten von SVG. Es muss für jedes SVG-Dokument angegeben werden, auf welche DTD es sich bezieht, deshalb wird zuerst der allgemeine Name, der *Public Identifier*, und zusätzlich noch der reale Speicherort, der *Document Identifier*, der DTD angegeben.

Das anschließende SVG-Tag ist das eigentliche Wurzelelement eines SVG-Dokuments. Das Wurzelelement ist jenes Element, das in einer auf XML basierenden Sprache alle anderen Textbereiche einschließt. Weil XML-Sprachen extensible, also erweiterbar sind, ist es möglich, verschiedene XML-Instanzen in einem Dokument zu integrieren. Dazu ist es nötig einen eindeutigen Namensraum (name space) für die Elemente der Auszeichnungssprache SVG festzulegen. Es könnte immerhin gerade der Fall eintreten, dass Element- oder Attributnamen von SVG auch in anderen, auf XML basierenden Auszeichnungssprachen vorkommen. Die Angabe eines eindeutigen Namensraums, als Attribut im Wurzelelement in Form einer URL besagt, dass alle verwendeten Elemente eindeutig zur Auszeichnungssprache SVG gehören. Um den maximal sichtbaren Bereich eines SVG-Dokuments festzulegen, ist es möglich die Attribute width und height im Wurzelelement <svg> anzugeben. Als Werte für width und height sind sowohl Gleit- als auch Festkommazahlen mit optionaler Maßangabe möglich. Interessant ist, dass Grafikelemente außerhalb dieses Bereichs zwar nicht dargestellt werden, deren Informationen aber nicht verloren gehen. Sollte keine Maßangabe angegeben sein, wird Pixel verwendet.

---

[4] UTF-8: 8-bit Unicode Transformation Format, verbreitetste Kodierung für Unicode Zeichen

2.3 Grundelemente

Nachdem das Grundgerüst eines SVG-Dokuments vorgestellt wurde, werden nun die wichtigsten Grundelemente beschrieben. Als Ausgangspunkt der Betrachtungen ist zu beachten, dass SVG zur Darstellung auf einem Ausgabegerät ein so genanntes „painters model" für das Rendering verwendet [Jac03]. Dies bedeutet, dass das Zeichnen in sequentiellen Operationen auf dem Ausgabegerät geschieht. Die Struktur eines SVG-Dokuments bestimmt die Zeichenreihenfolge. Wenn also ein bestimmter Bereich einen zuvor gezeichneten überlappt und dieser als opak[5] definiert ist, was in SVG der Standardeinstellung entspricht, so verdeckt er diesen.
Unterhalb des SVG–Wurzelelements können zunächst einige Elemente folgen, die ausschließlich der Strukturierung eines SVG–Dokuments dienen. Die Wichtigsten sind mit ihren entsprechenden Aufgaben im Folgenden tabellarisch zusammengefasst. (Tabelle 2-1)

Tabelle 2-1: Überblick der wichtigsten Strukturierungselemente von SVG

Name	Aufgabe des Elements
<title> <desc>	für die Dokumentbeschreibung (siehe Abbildung 2.3)
<g>	für die Gruppierung von mehreren Zeichenobjekten
<defs>	für die Definition von Referenzobjekten, die an anderer Stelle aufgerufen werden sollen
<use>	für die Referenzierung von Objekten (Grundformen, Texte, Bilder)

Neben Strukturierungselementen stellt SVG sechs Grundelemente zur Verfügung, um Grafiken zu beschreiben. Diese sind:

- Linie (line),
- Rechteck (rect),
- Kreis (circle),
- Polygon (polygon),
- Pfad (path) und
- Ellipse (ellipse).

[5] von Opazität: Maß für die Lichtundurchlässigkeit, Gegenteil von Transparenz

Diesen Elementen können Attribute zugeordnet werden, welche die Platzierung des Elements in der Grafik und seine Ausdehnung festlegen. Abbildung 2-3 zeigt ein Beispiel mit einigen Grundformen von SVG.

```
<?xml version="1.0" standalone="no" ?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
 "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="300" height="200" xmlns="http://www.w3.org/2000/svg" version="1.1">
 <title>Abbildung 2.3 - Studienarbeit Michael Zornow</title>
 <desc>einige Grundelemente von SVG</desc>

 <!-- ein gelbes Rechteck mit schwarzer Umrandung -->
 <rect x="10" y="20" width="75" height="100"
  style="fill:yellow; stroke:black; stroke-width:8;" />

 <!-- eine rote Ellipse mit grauer Umrandung -->
 <ellipse cx="200" cy="140" rx="80" ry="40"
  style="fill:red; stroke:grey; stroke-width:.1cm;" />

 <!-- ein Kreis mit Transparenz -->
 <circle cx="120" cy="100" r="90"
  style="fill:blue; stroke:black; opacity:.4;" />
</svg>
```

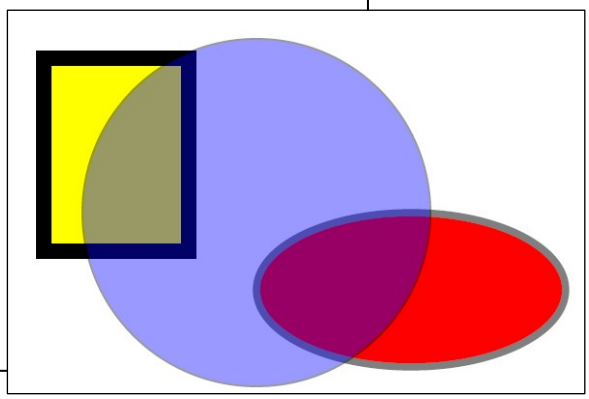

Abbildung 2-3: Die textbasierte Beschreibung einer SVG-Grafik (links) und die Grafik nach dem Rendern (rechts)

Ein weiteres spezielles Grundelement stellt *Text* dar. Mittels vieler Attribute kann dieser formatiert werden. Als Beispiele können hier die Schriftgröße, der Zeichenabstand oder auch die Schriftart angeführt werden. Es ist ebenfalls möglich, Texte an die Laufrichtung eines gegebenen Pfades anzupassen. Dies ist zum Beispiel in der Kartographie unverzichtbar. Abbildung 2-4 zeigt die gerenderte Ansicht einer SVG-Grafik mit zwei Textelementen.

Da SVG als XML-Instanz textbasiert ist, hat es den großen Vorteil, dass Browser, Plugins oder Suchmaschinen[6] die Möglichkeit haben den Textinhalt einer SVG-Grafik zu indexieren.

[6] Google bietet seit März 2005 die Möglichkeit SVG zu indexieren

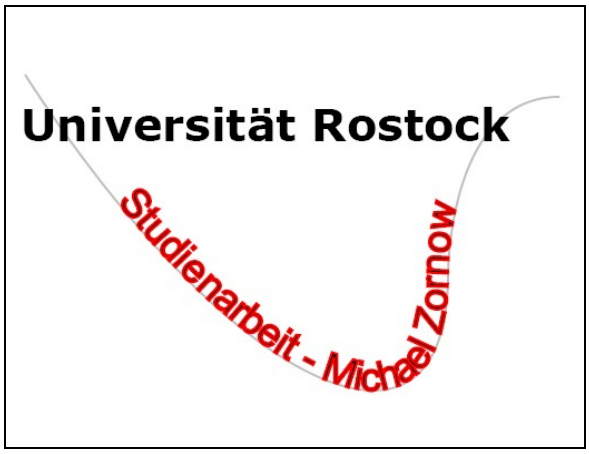

Abbildung 2-4: Standardtext und an einem Pfad ausgerichteter Text

Am Ende des Abschnitts 2.1 wurde bereits kurz erwähnt, dass es ebenfalls möglich ist externe Grafiken in ein SVG-Dokument einzubinden. Da sich hier interessante Möglichkeiten der Kombination von Rastergrafiken und SVG-Dokumenten für die mobile Informationsvisualisierung ergeben könnten, wird auch diese Möglichkeit von SVG kurz vorgestellt.
Um in SVG Bilder zu referenzieren, wird das Tag <imgage> eingesetzt. Damit ist es möglich GIFs, JPEGs und PNGs in ein Dokument einzubinden. Neben der Möglichkeit Rastergrafiken einzubinden, bietet SVG auch die Gelegenheit andere SVG-Dokumente (also Vektorgrafiken) zu referenzieren. Um referenzierte Bilder rendern zu können, wird die Größe und Position der Grafik über ein Rechteck beschrieben. Dieses wird mit Hilfe der Attribute X, Y, width, height festgelegt. Darüber hinaus lassen sich referenzierte Bilder durch weitere Attribute (z.B. style) manipulieren.

2.4 Interaktionsmöglichkeiten von SVG

Weil ein Teilziel dieser Arbeit darin besteht, die Interaktionsmöglichkeiten von SVG (speziell für mobile Geräte) zu untersuchen, ist es nötig die grundlegenden Möglichkeiten von SVG zur Interaktion mit einem Nutzer vorzustellen. Dies ist Gegenstand dieses Abschnitts.
Interaktion bezeichnet das wechselseitige aufeinander Einwirken von Akteuren oder Systemen. Der Begriff ist eng verknüpft mit dem der Kommunikation, manchmal werden diese beiden Begriffe sogar synonym

verwendet. Bei der Interaktion geht es also darum, dass ein Anwender nicht nur passiv etwas konsumiert. Stattdessen greift er aktiv in die dargestellten Informationen ein – hat also die Möglichkeit der Bearbeitung und Manipulation. In SVG sind dies die Möglichkeiten zur:

- Platzierung von Links
- Animation und
- Nutzung von Skripten.

Ein Hyperlink bildet die einfachste Methode, SVG-Dokumente interaktiv zu machen. Unter Verwendung von Links, also der Verbindung zwischen zwei Dokumenten (Hyperlinks) oder von Bereichen in einem Dokument selbst (Anker), Dokumente interaktiv zugreifbar. SVG–Hyperlinks entsprechen dabei beinahe den aus dem Internet bekannten HTML–Links. Anstelle von href wird aber xlink:href zur Referenzierung verwendet. Eingeleitet wird ein Hyperlink durch das Tag <a> wie es auch in HTML–Links verwendet wird. Es ist in SVG möglich, Animationen und Skripte durch Hyperlinks anzustoßen.
Eine Animation ist eine Reihe von Einzelbildern, mit der die Illusion einer Bewegung erzeugt wird. Sie beruht auf dem Prinzip, dass Veränderungen in aufeinander folgenden Einzelbildern im Gehirn als Bewegung interpretiert werden. Eine Animation in SVG verwendet zwar auch Einzelbilder, diese werden aber nicht von Hand erzeugt. Es genügt, einen Startzustand und einen Endzustand einer animierbaren Eigenschaft anzugeben. Die erforderlichen Zwischenbilder werden vom Prozessor mittels eines Interpolationsverfahrens berechnet. SVG stellt eine große Anzahl animierbarer Attribute bereit. So können durch die Verwendung des SVG–Elements <animate> zum Beispiel Objekte an einem Pfad entlang bewegt werden, Farben können sich sichtbar verändern oder Transformationen laufen innerhalb einer Animation ab. Abbildung 2-3 zeigt ein Beispiel zur Animation eines Kreises. Der Radius wird dabei nach zwei Sekunden, in einer ein-sekündigen Animation, von r=40 auf r=80 Einheiten verdoppelt.

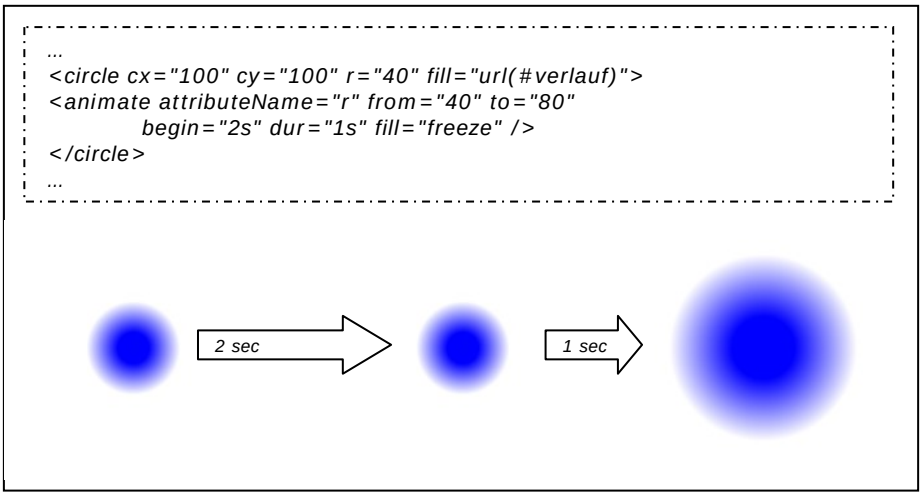

Abbildung 2-5: Beispiel zur Animation des Radius eines Kreises

Die Benutzerinteraktivität wird in Animationen durch so genannte event-values gewährleistet. Sie lösen ein Ereignis aus, wenn das entsprechende Element zum Beispiel den Fokus erhält (event-value: focusin) oder sich der Mauszeiger über dem Element befindet. (event-value: mouseover)
Aber SVG bietet weiterreichende Möglichkeiten, auf Benutzerinteraktionen und Ereignisse zu reagieren. Dies ist nicht nur im beschränkten Rahmen einer Animation möglich. Um eine skriptgesteuerte Interaktion zu gewährleisten, bedarf es zweier Komponenten:

- des Programms (Skripts), das die Aktion ausführt
- des Events, der über einen so genannten Event-Handler erfasst wird und die Ausführung des Skripts veranlasst.

Obwohl SVG standardmäßig ECMAScript[7] verwendet, sind auch JavaScript, JScript oder auch VBScript denkbar, um Interaktionen zu verwirklichen. Der jeweilige Skriptcode kann entweder direkt in das SVG-Dokument, eine referenzierende HTML-Datei oder in eine separate Skript–Datei geschrieben werden. Um mit einer Skriptsprache auf SVG–Elemente zugreifen zu können, muss zunächst sichergestellt werden, dass das Document Object Model (DOM) für das SVG–Dokument verfügbar ist. Das DOM stellt eine Schnittstelle (API) für Skriptsprachen dar. Das DOM eines SVG-Dokuments

[7] objektorientierte clientseitige Programmiersprache, Standard der ECMA (European Computer Manufactures Association), vollständig kompatibel zu JavaScript 1.5 und Jscript 5.5

besteht aus einer Baumstruktur, die zu der des SVG-Dokuments identisch ist. In dieser Baumstruktur stehen die Knoten des SVG-Dokuments, als Objekte (im Sinne einer objektorientierten Programmiersprache) für die Skriptsprache zur Verfügung.

Es ist Ziel dieser Arbeit SVG speziell auf die Nutzbarkeit für die mobile Visualisierung wissenschaftlicher Daten mittels SVG zu untersuchen. Nachdem in diesem Kapitel einiges zu den Grundlagen von SVG gesagt wurde, soll im nächsten Kapitel, SVG speziell für mobile Geräte vorgestellt werden.

Kapitel 3

SVG – speziell für mobile Geräte

Nach Standardisierung von SVG 1.0 im Jahr 2001 wurde schnell klar, dass selbst leistungsstarke PDAs[8] der damaligen Zeit nicht in der Lage sind SVG 1.0 vollständig zu unterstützen. Viele der Funktionen die SVG bietet, wurden als zu rechenzeitintensiv betrachtet, um auf mobilen Klein- und Kleinstgeräten dargestellt zu werden. Im Jahr 2003 entstanden, neben SVG 1.1 als modularisierte Version seines Vorgängers, auch zwei SVG-Varianten speziell für mobile Geräte. Zwei verschiedene mobile SVG-Varianten wurden als nötig erachtet, um der gesamten Vielfalt der mobilen Geräte gerecht zu werden. Im Jahr 2003 war der gesamte Markt mobiler Geräte nämlich in drei großen Säulen vorstellbar:

- Laptopcomputer
- Personal Digital Assistants und
- Mobiltelefone.

Heutzutage ist eine zunehmende Verschmelzung von Handys und PDAs zu verzeichnen, auf die am Ende des Abschnitts 3.3 noch eingegangen werden soll.
2003, im Jahr der Spezifikation der mobilen Profile von SVG 1.1, unterschieden sich Laptopcomputer, PDAs und Handys im Wesentlichen in Hinblick auf CPU-Geschwindigkeit, Displaygröße, Speichergröße und Farbunterstützung. Die Tabelle 3-1 zeigt dies mit Hilfe der technischen Daten jeweils eines Vertreters der mobilen Geräteklasse.

---

[8] Personal Digital Assistants

Tabelle 3-1: Vergleich der wichtigsten Hardwareeigenschaften - drei mobiler Geräteklassen aus dem Jahr 2003

	Laptop – Computer (Sony PCG-GRT815E)	PDA (Compaq IPAQ H3900)	Mobiltelefon (Nokia 3200)
CPU-Geschwindigkeit	2800 MHz	400 MHz	keine Angabe
Displayauflösung in Pixeln	1024 x 768	240 x 320	128 x 128
darstellbare Farben	32 bit	16 bit	12 bit
Speichergröße	RAM: 512 MByte erweiterbar	RAM: 64 MByte ROM: 32 MByte erweiterbar	Gesamtspeicher: 1 MByte, nicht erweiterbar

Nachdem die Leistung von Laptopcomputern durchaus an die von Desktopcomputern heranreicht und für SVG 1.1 ausreichend ist, wurde für stark eingeschränkte Mobilgeräte, wie etwa Handys, die SVG-Tiny (SVGT) Variante entworfen. Die zweite mobile Variante SVG-Basic (SVGB) wurde für mobile Geräte mit etwas weiterführenden Eigenschaften wie zum Beispiel PDAs konzipiert. Im Zuge von SVGT und SVGB spricht man häufig auch von SVG-Profilen anstelle von SVG-Varianten. Welche Charakteristika in den beiden Profilen SVGT und SVGB steckt, ist Gegenstand der Diskussion in den nachfolgenden Abschnitten.

3.1 Die SVG-Pyramide

Beide mobilen Profile, sowohl SVGT als auch SVGB, unterliegen Beschränkungen für Inhalt, Attributtypen, Eigenschaften und Benutzerprogrammverhalten. Dies ist nötig, um SVG mit begrenzten Hardwareressourcen anzeigen zu können.

Interessant ist, dass die mobilen SVG-Profile eine echte Untermenge des SVG 1.1 Darstellungsmodells bewahren, um die Kompatibilität mit diesem zu erhalten und existierende Inhalte anzeigen zu können. Zur Sicherstellung der Interoperabilität zwischen Inhalt und Softwareprogramm, die verschiedenen Profilen unterliegen, ist SVGT als echte Untermenge von SVGB spezifiziert. SVGB wiederum ist Untermenge von SVG 1.1. (siehe

dazu Abbildung 3-1). Dies hat den weiteren Vorteil, dass SVGT in SVGB bzw. SVG 1.1 transcodiert (also direkt umgewandelt) werden kann.

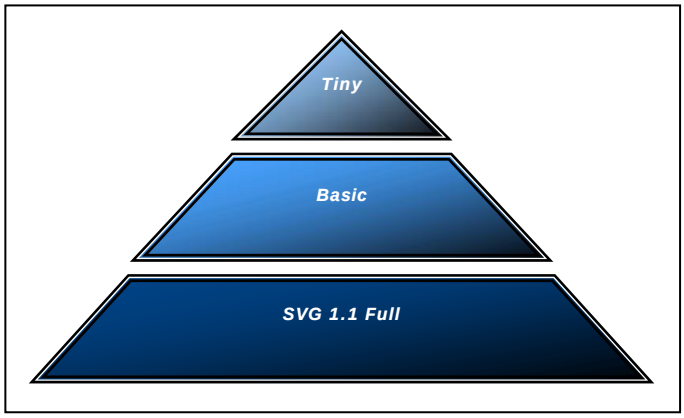

Abbildung 3-1: Die SVG-Pyramide, SVG-Basic als Untermenge von SVG 1.1 und SVG-Tiny als Untermenge von SVGB

3.2 Das SVGB- und SVGT-Profil

Bereits in der Einführung zum dritten Kapitel wurde erwähnt, dass viele SVG-Operationen, wie zum Beispiel Filtereffekte, Farbverlaufsfüllungen oder auch das Ausschneiden als zu rechenzeitintensiv angesehen werden, um auf mobilen Klein- und Kleinstgeräten implementiert zu werden. Neben einer Reihe formaler Unterschiede die daraus in SVGB und SVGT entstanden, wurden beide Profile zusätzlich vor einem unterschiedlichen Anwendungshintergrund entworfen.

SVGB sollte, obwohl es leichten Beschränkungen unterlag, eine breite Unterstützung für die am häufigsten benötigten Anwendungsszenarien liefern. SVGB-Inhalt kann, genauso wie SVG 1.1, entweder als allein stehendes SVG-Dokument, oder eingebettet in ein XML-Dokument vorkommen. SVGB unterstützt Festkommazahlen. Dabei unterstützt SVGB auch Einheitenangaben mit Hilfe von CSS-Einheiten wie: cm, mm, pt oder %. Bei der Definition von Formen macht das Basic-Profil fast keine Einschränkungen gegenüber SVG 1.1 - es unterstützt die sechs in Abschnitt 2.3 vorgestellten Grundformen. Die einzige Ausnahme bildet hier der Befehl für elliptische Kreisbögen, welcher nicht zulässig ist. Neben den meisten Textoperationen wird auch eine Untermenge der in SVG 1.1 möglichen

Filtereffekte vom SVGB-Profil unterstützt. Erwähnenswert ist ebenfalls, dass SVGB keine additiven Schnittpfade unterstützt und das Schnittpfade auf Rechtecke (rect-Elemente) beschränkt sind. Interaktion erlaubt SVGB laut Spezifikation in vollem Umfang.

SVGT ist vor einem anderen Anwendungshintergrund entworfen worden. Es war vor allem für die neue Generation von Mitteilungsdiensten in Form kleiner Trickfilme oder Animationen vorgesehen. [Qui04] Wie auch in Abbildung 3-1 verdeutlicht wird, unterliegt SVGT noch weiterreichenden Einschränkungen als SVGB. Der Hauptgrund dafür ist, neben dem bereits angesprochenen unterschiedlichen Anwendungshintergrund, vor allem die noch knapperen Hardwareressourcen mobiler Kleinstgeräte für die SVGT konzipiert wurde. So ist nun zum Beispiel die Einheitenangabe durch Anwendung von CSS-Einheiten nicht mehr zulässig. Einzige Ausnahmen sind die Einheitenangaben der Attribute width und height des äußeren <svg> Elements. SVGT erlaubt es nicht Texte an Pfaden entlang zu definieren. Auch Textfragmente, gebildet durch die Elemente <tspan> und <tref>, sind nicht definiert. Skripte, Gradienten und Muster sind in SVGT nicht zulässig. Dies bedeutet, dass nur noch einfache Farbfüllungen möglich sind. SVGT unterstützt weder Filtereffekte noch Operationen wie Ausschneiden oder Maskieren.

Da jedoch eine Nennung aller Gemeinsamkeiten und Unterschiede von SVGB und SVBT an dieser Stelle zu weit führen würde, sind im nächsten Abschnitt die wichtigsten Unterschiede schlussfolgernd zusammengefasst.

3.3 Zusammenfassung und Fazit von SVGB und SVGT als mobile Profile

Es ist festzuhalten, dass sowohl SVGB als auch SVGT einer Reihe von Beschränkungen unterliegen. Das Tiny–Profil ist dabei weitaus höheren Einschränkungen unterworfen als das Basic–Profil, wie auch aus Tabelle 3-2 deutlich wird, in der die wichtigsten Merkmale der Unterstützung von SVG 1.1 noch einmal gegenübergestellt werden.

Tabelle 3-2: Zusammenfassung der Möglichkeiten von SVGT und SVGB zur Unterstützung von SVG 1.1

Möglichkeit der Unterstützung	SVG – Tiny	SVG - Basic
Links	teilweise	teilweise
Skripte	---	voll
Animation	teilweise	voll
Transformationen	teilweise	teilweise
Gradienten und Muster	---	voll
Ausschneiden, Maskieren	---	teilweise
Filtereffekte	---	teilweise
Metadaten	voll	voll

Nachdem in den vorausgehenden Abschnitten zwei existierende Profile für mobile Geräte vorgestellt wurden, soll nun erläutert werden, warum diese Trennung der Profile als nicht mehr zeitgemäß angesehen wird.

Heutzutage ist eine zunehmende Verschmelzung von Handys und PDAs zu verzeichnen. So hat man die Möglichkeit mit Hilfe von PDAs zu telefonieren und Handys sind in der Lage, als mobiles Büro zu fungieren. Diese neue Generation der mobilen Geräte wird häufig als Smartphones[9] oder auch Mobile Digital Assistant (MDA) bezeichnet. Je nach Hersteller liegt der Schwerpunkt mehr auf der Funktionalität eines Telefons oder auf der eines PDAs. Abbildung 3-2 zeigt Smartphones der Firmen Nokia und Motorola.

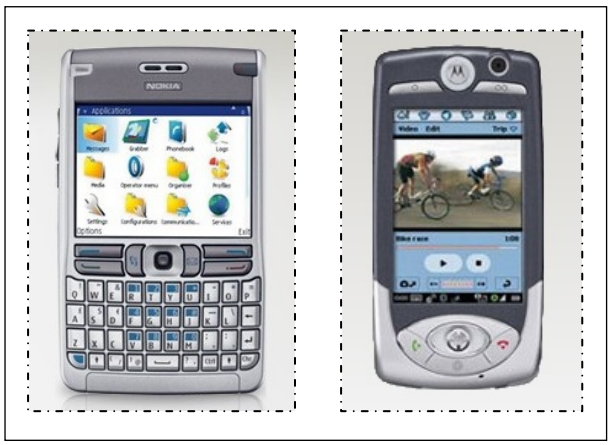

Abbildung 3-2: Nokia E61 (links) & Motorola A1000 (rechts)
([Nok06], [Mot06])

[9] zu deutsch etwa „schlaues Telefon"

Auch ist heutzutage in vielen Implementationen mobiler SVG–Viewer keine klare Trennung von SVGT und SVGB erkennbar. Als Beispiele sind hier die Viewer von Bitflash [Bit06] oder Intesis [Int06] zu nennen, welche nach eigenen Angaben „die meisten" SVGT und SVGB Eigenschaften unterstützen. Ein Grund für diese nicht erkennbare Trennung der Profile ist zum Teil sicherlich auch die bereits genannte zunehmende Verschmelzung der Funktionalitäten von Handys und PDAs.

Aufgrund dieses Trends in der Entwicklung wird im weiteren Verlauf dieser Arbeit der Begriff mobileSVG als Verschmelzung von SVGT und SVGB gebraucht, weil es nicht erforderlich bzw. überhaupt noch möglich ist, in der Praxis diese Begriffe von einander abzugrenzen.

Kapitel 4

Einsatzmöglichkeiten von SVG zur Visualisierung auf mobilen Endgeräten

Eine jede wissenschaftliche Visualisierung hat die Aufgabe auf geeignete Art und Weise wissenschaftliche Erkenntnisse zu repräsentieren. Sie bildet damit die Grundlage für eine angemessene Auswertung von Daten. Das Verständnis und die Analyse wissenschaftlicher Daten, Konzepte und Modelle soll mit Hilfe der wissenschaftlichen Visualisierung erleichtert werden. Bei jeder Visualisierung, also der Umsetzung von Daten und Informationen in anschauliche Darstellungen (Bilder), werden die bereits in Abschnitt 2.1 beschriebenen Schritte einer Visualisierungspipeline durchlaufen.

Dabei umfasst die wissenschaftliche Visualisierung verschiedene Spezialgebiete. Ein Spezialgebiet ist die Informationsvisualisierung, welche Gegenstand dieser Arbeit ist. Ziel der Informationsvisualisierung ist eine expressive und dabei effektive Darstellung der Datenmuster und der darin enthaltenen Informationen. Bilder sind expressiv, wenn die in einer Datenmenge enthaltenen Informationen (und nur diese!) präsentiert werden. Bilder sind effektiv, wenn sie intuitiv wahrgenommen werden können [Sch00].

M-dimensionale Daten vom Typ Skalar (mit $m \geq 2$), die in einem n-dimensionalen Beobachtungsraum gegeben sind werden als Mulitparameterdaten bezeichnet. Die m-dimensionalen Daten sind abhängige Variable, auch als multivariate Daten bezeichnet. Multivariate Visualisierungstechniken können in Abhängigkeit ihrer zugrundeliegenden Visualisierungsprinzipien in fünf Kategorien unterteilt werden -

geometrische Techniken, Icon- und Glyph-basierte Techniken, Pixel- und Voxel-orientierte Systeme, hierarchische Techniken und Techniken, die auf sogenannten Graphen basieren. Darüber hinaus existieren auch zahlreiche hybride Ansätze, die sich durch Kombination verschiedener Visualisierungstechniken aus den genannten Bereichen ergeben [Oel02]. In dieser Arbeit werden ausschließlich die geometriebasierten Ansätze zur Visualisierung mittels mobileSVG untersucht. Die Grundidee geometrischer Visualisierungstechniken basiert auf der Nutzung geometrischer Transformationen und Projektionen, um so die bestmögliche Darstellung multidimensionaler Daten zu realisieren. Dabei können prinzipiell sowohl zwei- als auch dreidimensionale Darstellungstechniken zum Einsatz kommen. In dieser Arbeit werden lediglich zweidimensionale Darstellungstechniken untersucht, da SVG für die Verwendung in 2D vorgesehen ist. Geometrische Techniken wurden zur Untersuchung gewählt, da sie das vielversprechendste Mittel sind, um mit SVG effektiv und angemessen Daten zu visualisieren – eben weil SVG-Dokumente nach Definition über geometrische Grundformen beschrieben werden. Eine pixelorientierte Technik zur Datenvisualisierung wie beispielsweise die Circle-Segment-Technik [Ank00], in der es darum geht jedem einzelnen Dimensionswert einem farbigen Pixel zuzuordnen, wurden aufgrund des vektoriellen Charakters von SVG von Anfang an ausgeschlossen.

4.1 Diagramme

Um mit mobileSVG effektiv wissenschaftliche Informationen zu visualisieren, sollte es möglich sein, seine Grundelemente so zu benutzen, dass eine geeignete Darstellung von Merkmalsausprägungen unterschiedlicher wissenschaftlicher Erhebungen möglich wird. Dazu ist es erforderlich, die unterschiedlichen, im Kapitel zwei vorgestellten, grundlegenden Einsatzmöglichkeiten, kombinieren zu können. Die Untersuchung der Möglichkeiten einer zweckdienlichen Attributierung der SVG-Grundelemente spielen dabei ebenso eine entscheidende Rolle, wie die Betrachtung der in Kapitel drei vorgestellten Einschränkungen denen mobileSVG unterliegt.

Dazu werden nun drei grundlegende Diagrammarten vorgestellt: die Säulendiagramme, die Kreisdiagramme und die Parallele-Koordinaten-Technik. Anhand dieser, werden die genannten Nutzbarkeitsbetrachtungen untersucht und beispielhaft erläutert. Des Weiteren dienen sie als Grundlage für die Projekte zur Visualisierung wissenschaftlichen Daten, welche im fünften Kapitel Gegenstand der Diskussion sein werden.

4.1.1 Säulendiagramme

Säulendiagramme veranschaulichen durch auf der x-Achse senkrecht stehende Rechtecke die Ausprägung von Messwerten. Sie werden hauptsächlich eingesetzt, um Unterschiede in verschiedenen Datenerhebungen über die Zeit oder über unterschiedliche Bedingungen darzustellen. Hier eignen sie sich auch für den Vergleich dieser Unterschiede. Säulendiagramme sollten nicht eingesetzt werden für Datenerhebungen die sehr viel umfangreicher sind als 15 Messwerte [Fow95].

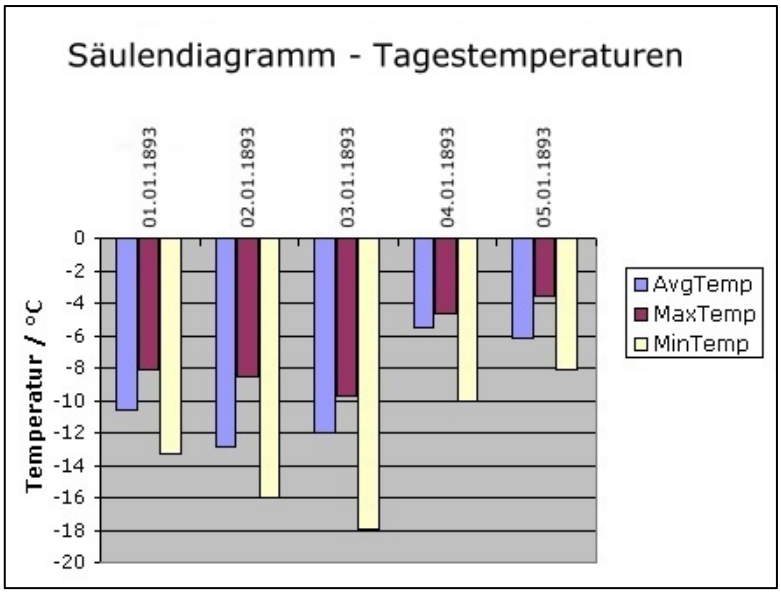

Abbildung 4-1: Säulendiagramm zur Visualisierung der Maximalen-, Minimalen- und Durchschnittstemperaturen unterschiedlicher Tage

In Abbildung 4-2 sind drei Datenkategorien: die Durchschnitts-, die Maximal- und die Minimaltemperaturen über einen Zeitraum von fünf

aufeinanderfolgenden Tagen dargestellt. Auf der rechten Seite erklärt eine Legende die Zuordnung der einzelnen Farben zu den Datenkategorien.
Das Balkendiagramm (Abb. 4-2) lässt sich problemlos mit mobileSVG zeichnen. Die dargestellten Balken und das Koordinatensystem selbst sind zum Beispiel über die geometrische SVG-Grundform <rect> realisierbar. Über die Attribute dieser Grundform lassen sich deren Eigenschaften näher beschreiben. So kann das Attribut „fill" verwendet werden, um die jeweilige Füllfarbe festzulegen. Die Attribute „stroke" und „stroke-width" ermöglichen es, jedem Rechteck einen Rand mit entsprechender Stärke zu geben. Auch der im Diagramm vorhandene Text ist in SVG realisierbar. Um den Text „Temperatur / °C" wie im Diagramm dargestellt auszurichten, sind im SVG-Tag <text> einige zusätzliche Attribute erforderlich. (vgl. Abb. 4-3)

```
…
<text x="120" y="40"
   writing-mode="tb" glyph-orientation-vertical="270"
   unicode-bidi="bidi-override" direction="rtl">
   Temperatur / °C
</text>
…
```

Abbildung 4-2: SVG Beschreibung zur Ausrichtung von Text

Mit dem Attribut „writing-mode" wird dabei die Laufrichtung eines Textes bestimmt. Der Attributwert „tb" bedeutet hier top bottom, also von oben nach unten. Die „glyph-orientation-vertical" bestimmt die Orientierung der einzelnen Zeichen. Der Defaultwert von 90 wird hier mit 270 Grad überschrieben. Das Attribut „unicode-bidi" mit seinem Attributwert „bidi-override" ist nötig, um die normale vom System festgelegte Darstellung der Zeichen aufzuheben. Erst dann ist es möglich über das Attribut „direction" die Direktionalität der Zeichen zu ändern - in Abbildung 4-3 auf den Wert „rtl" also right to left (von rechts nach links). Interessant ist weiterhin, dass bei der Erstellung des Balkendiagramms mit Hilfe von SVG und spezieller mobileSVG die Reihenfolge der Anordnung der Elemente im SVG-Dokument zu beachten ist. Dies ist nötig, weil SVG im Rendering Modell als Teil der Spezifikation eine implizite Abbildungsreihenfolge vorgibt – das in Abschnitt 2.3 eingeführte „painters model" für das Rendering. Im Falle des Säulendiagramms ist also darauf zu achten, dass zuerst die Diagrammgrundfläche, dann das Koordinatensystem und zum Schluss die

Beschriftungen und die Rechtecke gezeichnet werden. Dies wird, wie bereits erwähnt, durch die Reihenfolge der Anordnung der Elemente im SVG-Dokument erreicht.

4.1.2 Kreisdiagramme

Das Kreisdiagramm ist eine Darstellungsform für Teile eines Ganzen. Es ist kreisförmig in mehrere Segmente unterteilt. Jedes Kreissegment entspricht dabei einem Teilwert, und der Kreis der Summe der Teilwerte als Ganzes. Die jeweilige Segmentgröße als Winkel kann wie folgt bestimmt werden:

$$Segmentwinkel = \frac{360 * Teilwert}{Gesamtwert}$$

Kreisdiagramme können eingesetzt werden, um Teile eines Ganzen zu einem bestimmten Zeitpunkt gegenüberzustellen und zu vergleichen. Dabei ist zu beachten, dass die Abschätzung von Winkeln einem Menschen vergleichsweise schwierig fällt. Daher sollte es nur für approximative Darstellungen von Beziehungen verwendet werden [Fow95].
In SVG eignen sich im Moment ausschließlich Befehle für elliptische Kurven, des in Abschnitt 2.3 vorgestellten Grundelements <path>, zum Erstellen von Kreisdiagrammen. Den eigentlichen Verlauf des Pfades bestimmen man durch das, im path-Element zwingend anzugebende Attribut *d*.
Das Attribut *d* erwartet als Wert einen alphanumerischen Wert, wobei Buchstaben jeweils den Verlauf des Pfades festlegen und Zahlen die notwendigen Koordinaten-, Winkel- oder sonstige Angaben für den entsprechenden "Verlaufsbuchstaben" bestimmen. Die Buchstaben und Zahlen können durch ein Leerzeichen und/oder ein Kommazeichen beliebig von einander getrennt werden. So kann man beispielsweise mit dem Buchstaben *A* eine elliptische Bogenkurve erzeugen. Bei einer elliptischen Bogenkurve wird zuerst eine Ellipse festgelegt, um welche dann der Pfad bzw. die Kurve verläuft. Der Anweisung für *A* müssen zwingend 5 Werte folgen, welche trigonometrische Berechnungen erforderlich machen. Dies sind:

1. der Radius der X-Achse der Ellipse
2. der Radius der Y-Achse der Ellipse
3. die Rotation der X-Achse der Ellipse in Grad
4. das Large-Arc-Flag: 0 für den kurzen-, 1 für den langen Weg um die Ellipse
5. das Sweep-Flag: 0 für Zeichnung gegen-, 1 für Zeichnung mit dem Uhrzeigersinn

Abbildung 4-4 zeigt ein Beispiel für die Nutzung dreier elliptischer Kurvenbefehle zur Erstellung eines SVG-Kreisdiagramms.

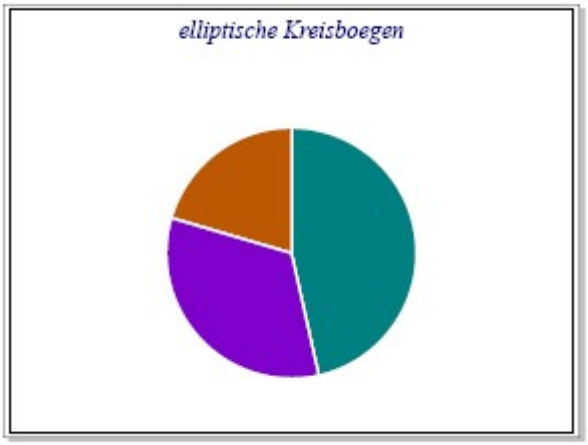

Abbildung 4-3: SVG-Beispiel zur Nutzung von elliptischen Kreisbögen zur Erstellung eines Kreisdiagramms

Neben der Überschrift, erzeugt durch ein SVG-Textelement, wie es bereits im vorigen Abschnitt vorgestellt wurde, sind drei verschiedenfarbige Kreissegmente erkennbar. Abbildung 4-5 zeigt welcher SVG-Inhalt erforderlich ist, um beispielsweise den in Abbildung 4-4 erkennbaren mintfarbenen Kreisbogen nach obiger Vorschrift zu erzeugen.

```
…
<g id="mintfarbener Bogen"
style="stroke:#FFFFFF; stroke-width:20;  fill:#008080;" >
<a  xlink:href="http://www.amazing-michael.de">
<path d="M 2160 1860 L 2365 2775 A 938 938 0 0 0 2160 922 z" />
</a>
</g>
…
```

Abbildung 4-4: SVG-Beschreibung einer elliptischen Bogenkurve zur Erstellung eines Kreisdiagramms

Im Ergebnis heißt das, dass Kreissegmente bisher in SVG nicht berücksichtigt sind. Ihre Benutzung ist nur über den Umweg der elliptischen Kreisbögen möglich. Diese machen eine aufwendige Bestimmung der vorgestellten, erforderlichen Werte vom „Verlaufsbuchstaben" *A* des Pfadattributes *d* nötig. Für die Visualisierung mittels mobileSVG ergeben sich darüber hinaus durch die elliptischen Kreisbögen einige Unsicherheiten. So unterstützt weder das SVGT- als auch das SVGB-Profil laut Spezifikation die Pfadbefehle für elliptische Kreisbögen [Cap03]. Dies bedeutet, dass es nicht möglich ist auf Geräten, welche ausschließlich SVGT und/oder SVGB unterstützen Kreisbögen darzustellen! Für die Visualisierung durch SVG auf mobilen Geräten hat dies weitreichende Konsequenzen. So ist es standardmäßig nicht nur nicht möglich Kreisdiagramme darzustellen, sondern alle Arten von wissenschaftlichen Visualisierungen, welche elliptisch Kreisbögen[10] erforderlich machen. Als prominente Beispiele seien hier die ThemeRiver-Technik [Hav02] oder die Sunburst-Technik [Sta00] genannt.
Trotzdem war es möglich Kreisdiagramme zum Beispiel mit dem Intesis eSVG-Viewer auf einem PDA zu betrachten. Dies hat die Ursache, dass dieser Viewer (wie auch viele weitere mobileSVG-Viewer), neben den mobilen SVG-Profilen, ebenfalls eine Untermenge des SVG1.1-Profils unterstützen. In dieser Untermenge waren, in diesem speziellen Fall des eSVG-Viewers, auch die Befehle für elliptische Kreisbögen enthalten.
Für SVG 2.0 sind Kreissektoren (pie slices) als Grundform angedacht. Es bleibt abzuwarten, ob sie auch in die mobilen Profile Einzug halten werden. Man sollte also auf die Verwendung von elliptischen Kreisbögen auf mobilen SVG-fähigen Geräten, zum Beispiel zur Erstellung von Kreissegmenten, verzichten. Ist es doch einmal zwingend erforderlich auf einem PDA oder anderen mobilen Geräten mit SVG Kreisbögen zu zeichnen, muss abhängig vom verwendeten Viewer und der dadurch unterstützten SVG-Profile eine Nutzbarkeitsbetrachtung durchgeführt werden.

[10] mit dem Spezialfall der Kreissegmente

4.1.3 Parallele-Koordinaten-Technik

Bei dieser Technik werden die einzelnen Variablen durch parallel angeordnete Achsen gleichen Abstands repräsentiert. Der entsprechende Wertebereich der Variablen ist entlang der einzelnen Achsen aufgetragen. Beobachtungsfälle bzw. Datentupel werden dabei durch polygonale Linien dargestellt, welche die Achsen an den entsprechenden Stellen schneiden [Ins85].

Idee dieser Darstellungsform ist, dass durch die Form der Polylinien bzw. durch das Entstehen von Kreuzungspunkten Rückschlüsse auf die Ausgangsdaten gezogen werden können.

Neben der Beschriftung für die Merkmalsachsen, welche durch Textelemente realisierbar sind, werden ausschließlich <path> Elemente für die Visualisierung verwendet. Diese, ausgestattet mit zusätzlichen Attributen für Strichstärke, Strichfarbe usw. reichen aus, um die Darstellungstechnik der Parallelen Koordinaten zu realisieren. Abbildung 4-6 zeigt ein SVG-Beispiel für Parallele Koordinaten, welches aus einem Datensatz von 364 Messreihen mit jeweils 11 Merkmalen erstellt wurde.

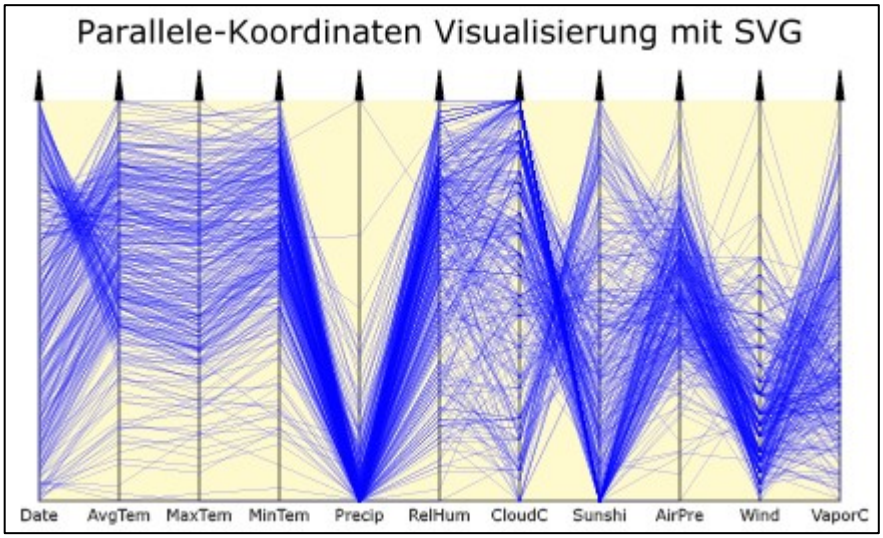

Abbildung 4-5: SVG-Visualisierung von 364 Messreihen mittels Paralleler-Koordinaten-Technik

Die Spezifikation von mobileSVG ist zur Umsetzung der Parallelen-Koordinaten-Technik hervorragend geeignet. Es ist daher festzuhalten, dass

wohl auch jede andere Visualisierungstechnik, dessen Beobachtungsfälle vor allem auf der Darstellung von Pfaden beruhen, sehr effizient und effektiv mit SVG und auch mit mobileSVG dargestellt werden kann. Dabei scheint auch die Darstellung größerer Datenmengen möglich zu sein. Diese Behauptung zu untermauern bzw. kritisch zu hinterfragen wird Aufgabe der Diskussion des nächsten Abschnitts sein.

4.2 Anzahl der Primitive und Darstellbarkeit

In den letzten drei Abschnitten stand die grundsätzliche Nutzbarkeit von mobileSVG zur Visualisierung von Informationen mit Hilfe von Diagrammen im Mittelpunkt. Ziel dieses Abschnitts soll es sein zu untersuchen, welchen Einfluss die Anzahl der Primitive auf die Darstellbarkeit von SVG-Dokumenten auf mobilen Endgeräten hat. Bereits im dritten Kapitel wurde deutlich gemacht, dass mobile Geräte weitreichenden Hardwareeinschränkungen unterliegen. Deshalb ist es unerlässlich zu untersuchen, inwiefern sich der begrenzte Speicher und der vergleichsweise langsame Prozessor eines mobilen Gerätes auf die mögliche Anzahl geometrischer Primitive einer mobilen SVG-Grafik auswirken. Dabei spielt natürlich auch die Darstellbarkeit dieser Primitive auf den vergleichsweise kleinen Displays mobiler Geräte eine Rolle.

Zwei wesentliche Probleme gilt es sich dazu vor Augen zu führen. Das erste Problem ist der langsame Prozessor und der geringe Speicher mobiler Geräte. Das zweite Problem ist die kleine Displayfläche, verbunden mit der geringen Auflösung von PDAs. Die Konsequenz dieser Probleme ist, dass Bilder vergleichsweise langsam gerendert werden. Hinzu kommen hohe Bildladezeiten und stockende Animationen, weil die Bildwiederholrate für kontinuierlich wahrgenommene Bewegungen nicht ausreicht. Darüber hinaus ist immer nur ein sehr kleiner Teil der SVG-Grafik, vergleichsweise schlecht aufgelöst, sichtbar. Um diesen Problemen zu begegnen, sollen im Abschnitt 4.2.1 einige Möglichkeiten vorgestellt werden. Ziel ist es durch kluge Ausnutzung der SVG-Spezifikation, die Anforderungen an die verwendete Hardware zu reduzieren. Der Abschnitt 4.2.2 soll dann einige

Lösungsansätze für das Problem der geringen Auflösung und Displaygröße diskutieren.

4.2.1 Optimierungsmöglichkeiten

Im Rahmen dieser Arbeit stand ein PDA des Typs „Compaq IPAQ H3900" zur Verfügung. (vgl. Tabelle 3-1) Um das in Abbildung 4-6 gezeigte SVG-Dokument vollständig zu rendern, muss ein Anwender bei der Verwendung des mobilen eSVG-Viewers von Intesis [Int06] bereits eine Ladezeit von mehr als 10 Sekunden akzeptieren. Durch geschickte Implementation des mobilen SVG-Inhalts ist es möglich diese Zeit zumindest ein wenig zu reduzieren. Dabei stehen diverse Möglichkeiten zur Verfügung. Ausgangspunkt aller dieser Betrachtungen ist es, dass jedes zusätzliche Zeichen SVG-Code die Ladezeit des SVG-Dokuments verlängert. Durch folgende Maßnahmen ist man in der Lage den Code zu verkürzen [Fib02]:

- Gruppieren der Elemente mit Hilfe des Tags <g>, um eine einmalige Zuweisung gleicher Formatierungen der Elemente zu ermöglichen
- Einmaliges definieren der Grafiken, welche mehrmals benötigt werden und Referenzieren dieser mit Hilfe des Tags <use>
- Wenn möglich, Nutzen der Default-Werte und dabei Verzicht auf deren Auszeichnung
- Ohne die Lesbarkeit zu sehr einzuschränken, Entfernen der unnötigen Leerräume aus dem Dokument

Weiterhin gibt es einige Möglichkeiten den Berechnungsaufwand für die Darstellung der SVG-Grafik zu minimieren. Dies ist aufgrund der verminderten Rechenleistung mobiler Geräte sinnvoll. Maßnahmen dafür sind beispielsweise:

- Bei der Koordinatenbeschreibung so wenig wie möglich Nachkommastellen verwenden, bzw. Nutzen der Nachkommastellen nur dort, wo es erforderlich ist

- Vermeiden der Verwendung von Filtereffekten und Farbverläufen, bzw. Definition dieser Regionen dafür so klein wie möglich

Werden die oben genannten Empfehlungen eingehalten, ist es realisierbar die in Abbildung 4-6 dargestellte SVG-Grafik unter Nutzung der gleichen Hard- und Software auf eine Ladezeit von unter 10 Sekunden zu reduzieren. Selbst wenn man jedoch hohe Ladezeiten akzeptiert und diese durch die bereits beschriebenen geeignete Maßnahmen reduzieren kann, stellt sich ein weiteres Problem: bei der Darstellung der Grafik ist es schwierig oder gar unmöglich Zusammenhänge zu erkennen oder Rückschlüsse aus Informationsvisualisierungen zu ziehen, weil die vergleichsweise geringe Displayfläche und Auflösung mobiler Geräte dies verhindern. Auch dafür bietet mobileSVG einige Lösungsmöglichkeiten. Sie sollen nun näher beschrieben werden.

4.2.2 mobileSVG - Lösungen zur Darstellung auf kleinen Displays

Um der Forderung der Informationsvisualisierung gerecht zu werden, dass Rückschlüsse aus Daten möglichst rasch und intuitiv erfassbar sein sollen, ist ein erster Ansatz eine Reihe unterschiedlicher Farben zur Unterstützung der Darstellung einzusetzen. Die kleinere Displaygröße und Auflösung mobiler Geräte soll durch die Verwendung nicht nur einer, sondern vieler verschiedener Farben zur Darstellung kompensiert werden. Dies ist am Beispiel der Parallelen-Koordinaten-Technik untersucht worden. Abbildung 4-7 zeigt zehn Pfade der Parallelen-Koordinaten-Technik mit unterschiedlichen Farben. MobileSVG bietet dafür die Unterstützung eines 16 Bit mächtigen Farbraumes und auch PDAs sind in der Lage diese Anzahl an Farben darzustellen. (vgl. z.B. Tabelle 3-1)

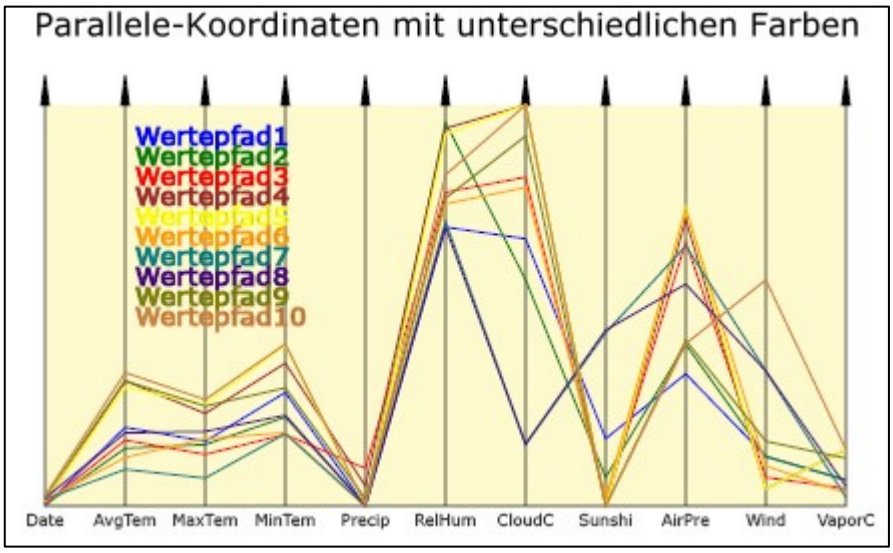

Abbildung 4-6: Beispiel zur Nutzung unterschiedlicher Farben zur Informationsvisualisierung mittels mobileSVG

Der Vorteil des Einsatzes mehrerer Farben ist zum Beispiel, dass Polylinien unter Nutzung der Parallelen-Koordinaten-Technik nun eindeutig Beschriftungen über eine Legende zuweisbar sind. Abbildung 4-7 illustriert diese Möglichkeit. Wesentlicher Nachteil ist, dass der Einsatz zu vieler Farben, zum Beispiel bedingt durch eine hohe Anzahl an unterschiedlichen Datensätzen, schnell zu einer unübersichtlichen SVG-Visualisierung führt. Dieser Effekt wird noch verstärkt durch die Tatsache, dass SVG ein bereits angesprochenes „painter model" zur Darstellung benutzt und Polylinien eventuell vollständig oder auch nur teilweise übermalt werden können. Zur Darstellung von Parallelen-Koordinaten-Pfaden, wie in Abbildung 4-7, ist es deshalb nicht sinnvoll wesentlich mehr als zehn unterschiedliche Farben einzusetzen. Dies gilt auch übertragen auf andere Visualisierungstechniken, welche Pfade zur Darstellung der Beobachtungsfälle nutzen. Das beschränkte Wahrnehmungsvermögen des Menschen in Verbindung mit der vergleichsweise kleinen Displayfläche der PDAs (und natürlich auch Handys) hemmt die Nutzbarkeit zu vieler Farben erheblich. Es ist einzusehen, dass der Einsatz von unterschiedlichen Farben mit SVG allein die kleinere Displaygröße und Auflösung mobiler Geräte zum Erhalt der intuitiven Erfassbarkeit von Informationen aus Daten nicht kompensieren kann.

Dennoch ist es in vielen Fällen erforderlich weit mehr als zehn Wertepfade mit einer Visualisierungstechnik gleichzeitig darzustellen. Dazu gibt es

weitere Lösungsansätze unter Nutzung der skalierbaren Vektorgraphik SVG. Ein möglicher Ansatz wäre es, die Strichstärke der Pfade anzupassen. MobileSVG bietet hier die Möglichkeit, Pfaden (wie auch weiteren Grundformen) über das Attribut „stroke-width" einen Wert zu übergeben. Die Abbildung 4-8 und 4-9 zeigen ein Beispiel: erneut werden in Abbildung 4-8 mit Hilfe der Parallelen-Koordinaten die Wertepfade mit einer Strichstärke von 1.0 Pixeln dargestellt; Abbildung 4-9 zeigt dasselbe Diagramm mit einer reduzierten Strichstärke von 0.2 Pixeln.

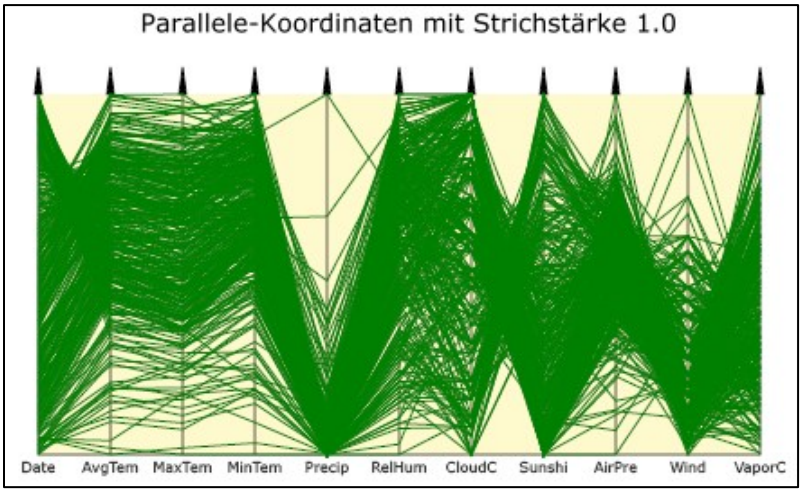

Abbildung 4-7: Beispiel zur Parallelen-Koordinaten-Visualisierung mit einer SVG-Strichstärke von 1.0 Pixeln

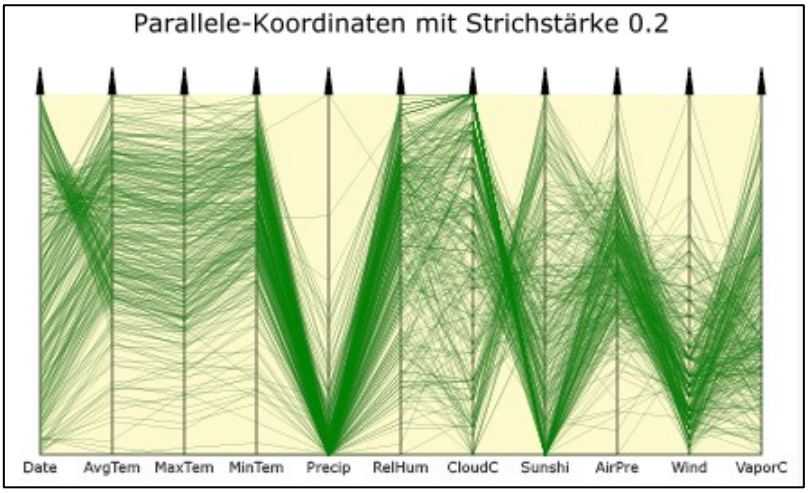

Abbildung 4-8: Beispiel zur Parallelen-Koordinaten-Visualisierung mit einer SVG-Strichstärke von 0.2 Pixeln

Anhand der Abbildungen wird deutlich, dass entstehende Kreuzungspunkte in Abbildung 4-9 wesentlich intuitiver auszumachen sind und Rückschlüsse über die Ausgangsdaten so einfacher erzielt werden können. Die Möglichkeit der Anpassung der Strichstärke in mobileSVG stellt, vernünftig eingesetzt, ein geeignetes Mittel dar, um Informationen effektiv zu visualisieren. Dem steht auch die vergleichsweise geringe Displayfläche mobiler Kleingeräte wie PDAs, die heutzutage zumeist mindestens 320x240 Pixel beträgt, nicht im Wege.

Falls die Auflösung doch einmal wesentlich geringer sein sollte, wie es etwa bei mobilen Kleinstgeräten wie Handys möglich sein kann, ist es nicht sinnvoll die Strichstärke zu sehr zu reduzieren - es wird an dieser Stelle nötig die sichtbare Datenmenge zu reduzieren. Hierfür bietet mobileSVG eine weitere interessante Möglichkeit: Bildausschnitte können über das Attribut „viewBox" festgelegt werden. Bereits aus dem Kapitel über die fundamentale Dokumentstruktur von SVG ist bekannt, dass grundsätzlich die Ausdehnung einer SVG-Grafik durch die Attribute width und height im äußeren SVG-Element festgelegt wird. Damit wird dann auch das Koordinatensystem der Grafik (der Viewport) bestimmt, welches zum platzieren der Objekte verwendet wird. Durch den Gebrauch des Attributs viewBox – normalerweise im SVG-Wurzelelement – kann man innerhalb der SVG-Grafik ein unterschiedliches Koordinatensystem verwenden. Dieses Koordinatensystem kann z.B. größer oder kleiner sein als das der "normalen" SVG Grafik. Der Nutzen des Attributes „viewBox" besteht nun in der so genannten *automatic transformation*. Diese bewirkt, dass ein so definiertes, "neues" Koordinatensystem beim Rendern vom User Agent automatisch auf die, im SVG-Element durch width und height angegebene, normale Größe der Grafik (den "normalen" Viewport) skaliert wird. Lediglich eine Zeile SVG-Code (vgl. Abbildung 4-10 oben) muss geändert werden, um eine angepasste Ausgabe auf unterschiedlichen Geräten zu gewährleisten. Der Rest des SVG-Dokuments muss dabei nicht verändert werden. Abbildung 4-10 zeigt ein Beispiel zur Nutzung des Attributes „viewBox".

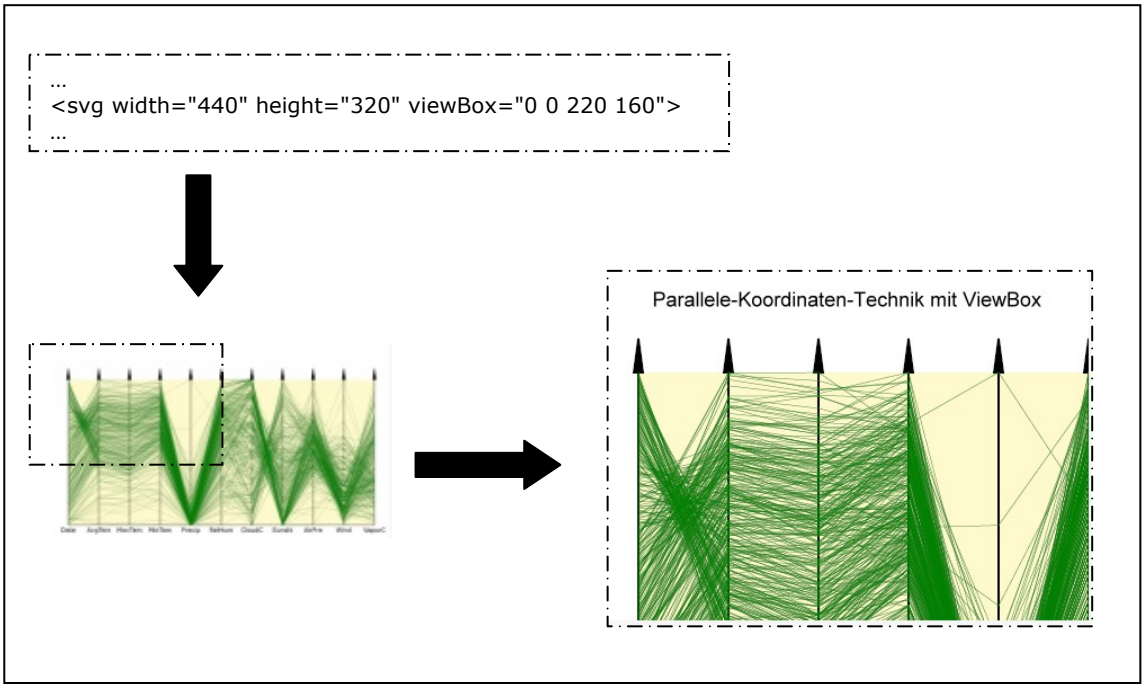

Abbildung 4-9: Die Nutzung von SVG-viewBox zur Darstellung von Bildausschnitten einer SVG-Grafik

Durch den vektoriellen Charakter von SVG ist eine Skalierung möglich ohne dabei Bildqualität einzubüßen. Dies stellt einen weiteren großen Vorteil dar, weil man gerade im mobilen Umfeld oft mit einer Vielzahl unterschiedlicher Gerätedisplayauflösungen und Displaygrößen konfrontiert wird. Die Möglichkeit der qualitätsverlustlosen Skalierung durch Definition von Zielansichten über das Attribut „viewBox" ist an dieser Stelle also besonders nützlich. Dies gilt für die vorgestellte Parallele-Koordinaten-Technik, aber genauso natürlich für jede andere Visualisierungstechnik.

In diesem Abschnitt wurde die Variation einiger Attribute (Farbe, Strichstärke, viewBox) von SVG beispielhaft untersucht, um daraus einen Vorteil für die Visualisierung zu ziehen. Diese Variation der Attributierung der SVG-Grundelemente geschah stets vor der Darstellung der SVG-Grafik – also vor dem Rendering-Schritt (vgl. Abschnitt 2.1). Natürlich ist es möglich die Variation weiterer solcher Attribute zu untersuchen, um sie gezielt für eine SVG-Visualisierung einzusetzen. Dies soll an dieser Stelle jedoch nicht weiter erfolgen. Stattdessen soll im folgenden Abschnitt untersucht werden, welchen Vorteil man für eine Visualisierung mittels der in Abschnitt 2.4 vorgestellten SVG-Interaktionsmöglichkeiten erzielen kann. Hier werden die verschiedenen Variationen der Attribute der SVG-Grundelemente zur

Laufzeit – also während der Darstellung – vorgenommen. Dies erzeugt wahrscheinlich höhere Anforderungen an die Performance des verwendeten Anzeigesystems. Die möglichen Auswirkungen auf eine mobile Visualisierung finden daher nun besondere Beachtung.

4.3 Interaktionsmöglichkeiten

Die Platzierung von Links, die Animationen und die Nutzung von Skripten wurden im Zuge dieser Arbeit als Interaktionsmöglichkeiten von SVG vorgestellt. Die Motivation dieses Abschnitts ist es, zu untersuchen, ob diese Möglichkeiten problemlos auf mobilen Systemen einsetzbar sind. Neben einem leichteren Verständnis der Darstellung, soll auch eine intuitivere Erfassung und Auswertung der Repräsentationen durch den Anwender ermöglicht werden. Im Laufe dieser Arbeit wurden Untersuchungen an ausgewählten Beispielen durchgeführt. Dieses Vorgehen wird nun auch im nächsten Abschnitt so vorgeführt. Da die Betrachtung aller möglichen interaktiven Anwendungsfälle im Rahmen dieser Arbeit zu weit führen würde, werden nun stattdessen einige interaktive Anwendungsfälle betrachtet, um ein allgemeingültiges Fazit für eine Nutzbarkeit ziehen zu können. Zunächst soll nun die Verwendung von Links im Mittelpunkt stehen. In einer SVG-Grafik kann man jedes Element als Hyperlink festlegen. Den Nutzen für die mobile Visualisierung finden Hyperlinks beispielsweise durch die Tatsache, dass zusätzliche erklärende Informationen über die dargestellte SVG-Grafik auf Wunsch angezeigt werden können - die SVG-Grafik wird so für den Nutzer verständlicher. Natürlich ist es auch möglich mehrere Varianten derselben SVG-Grafik, z.B. Erhebungen unterschiedlicher Jahre, untereinander zu verlinken. Eine schnellere Interpretation und ein sehr effektiver Vergleich der Daten ist somit möglich. Abbildung 4-11 zeigt schematisch diese Möglichkeit der Nutzung von Hyperlinks.

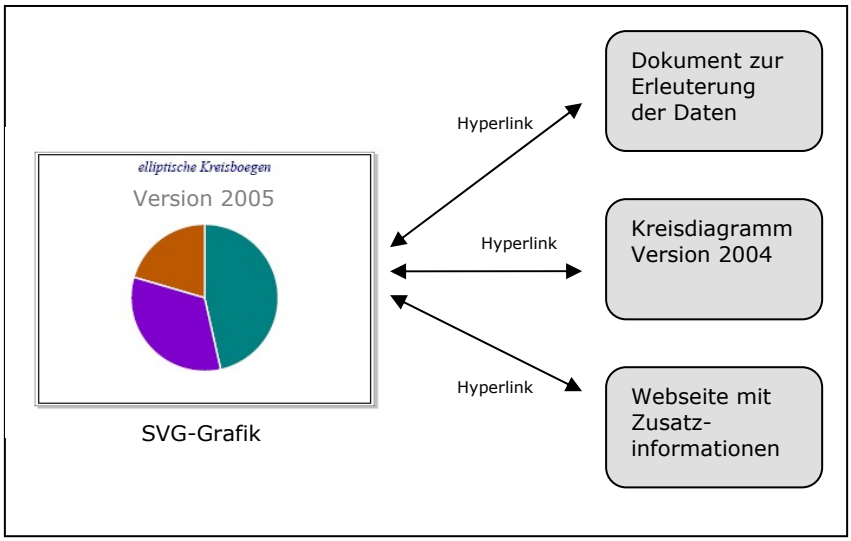

Abbildung 4-10: Schematische Darstellung der Nutzung von Hyperlinks mit einer SVG-Grafik als Ausgangspunkt

Da die bisher beschriebene Art der Nutzung von Hyperlinks jedoch bereits aus dem Umfeld des Internets hinreichend bekannt ist, soll stattdessen an dieser Stelle eine grafische Besonderheit in Bezug auf Interaktivität mit Hyperlinks eingehender betrachtet werden: das SVG-Basic-Profil[11] bietet die Möglichkeit, Zielansichten innerhalb einer SVG-Grafik zu definieren, die über einen Link aufgerufen werden können. Damit ist es möglich, ein einfaches, aber sehr effektives Navigationsmittel für jede Form mobiler SVG-Grafik zu erstellen. Mit Hilfe des Tags <view> wird dabei ein bestimmter Ansichtsbereich spezifiziert, der sich dann verlinken lässt. Dazu ein Beispiel: Die Zeile

```
<view id="viewNr1" viewBox="0 0 220 160"/>
```

definiert eine Zielansicht, welche dann durch einen Link aufgerufen werden kann:

```
<a xlink:href="#viewNr1">
<text x="100" y="100">Ausschnitt oben links</text>
</a>
```

[11] das SVG-Tiny-Profil bietet diese Möglichkeit in der Version 1.1 nicht

Der Nutzen dieses Vorgehens wird zum Beispiel sichtbar, wenn man sich noch einmal die Möglichkeiten des Attributes „viewBox" aus dem Abschnitt 4.2.2 vor Augen führt. Dort war es erforderlich jeweils eine Zeile der SVG-Grafik zu verändern, um eine Anpassung der Darstellung der SVG-Grafik zu erreichen. Diese Restriktion ist nun aufgehoben. Man kann beliebig viele Zielansichten innerhalb eines Dokuments definieren und so interaktiv verfügbar machen. Zielansichten sind durch beliebige SVG-Elemente zur Laufzeit mit Hilfe von Links aufrufbar, ohne dabei den SVG-Code verändern zu müssen. Das dies gerade im mobilen Umfeld mit geringen Displayflächen zur Darstellung einer Grafik von tragender Bedeutung ist, wurde bereits im Abschnitt 4.2.2 angesprochen. Die verfügbare Displayfläche wird durch die interaktive Anpassung der Zielansicht noch effektiver genutzt und dem Anwender wird die Möglichkeit eingeräumt, Rückschlüsse aus Daten intuitiver zu erfassen und zu analysieren. Hyperlinks sind in der Lage Animationen zu starten oder in Verbindung mit einer Skriptsprache spezielle Funktionen anzustoßen. Ob die Interaktionsmöglichkeit durch Nutzung von deklarativer Animation[12] für die mobile Visualisierung mit SVG von Nutzen sein kann soll nun diskutiert werden.

Wie schon an vielen angesprochenen Bereichen dieser Arbeit steht man im mobilen Umfeld vor dem Problem der geringen Displaygröße und Auflösung mobiler Geräte. Es sind daher Lösungen gefragt, um eine Visualisierung dennoch darstellen zu können. Speziell vor dem Hintergrund der Darstellung großer Datenmengen, wäre es daher eine effektive Möglichkeit, etwa einzelne Cluster der Parallelen-Koordinaten-Technik oder einzelne Variablen von Säulendiagrammen nutzergesteuert zur Laufzeit ausblenden oder hervorheben zu können. Dass mobileSVG hier mit Hilfe der deklarativen Animation eine gute Möglichkeit bietet, soll nun beispielhaft dargestellt werden. Durch die Verwendung des Elements <animate> ist es möglich, genau einen Wert eines Attributs oder einer Eigenschaft innerhalb eines bestimmten Zeitablaufs zu verändern. Es gibt in SVG eine sehr große Anzahl von animierbaren Attributen. Um eine Variable eines Säulendiagramms hervorzuheben wird nun das Attribut „opacity" animiert. Die SVG-Syntax dafür lautet wie folgt:

---

[12] eine Animation, welche keine Scriptfunktionalität (ECMAScript etc.) nutzt

```
<animate xlink:href="#path1w"
    attributeName="opacity"
    attributeType="XML"
    values="0.2; 1; 1"
    dur="10s"
    begin="path1Label.click" end="10s"
    restart="whenNotActive"
    fill="remove" />
```

Im Attribut „attributeName" steht der Name des zu animierenden Attributes. Durch Verwendung des Attributs „values" kann man beginnend mit dem Startwert und abschließend mit dem Endwert, eine Liste von Zwischenwerten angeben, die bei der Berechnung der Animation berücksichtigt werden. Alle Werte werden untereinander durch Semikolon getrennt und sind in ihrer Art abhängig von der animierten Eigenschaft. Die Lichtundurchlässigkeit (opacity) hat also einen Startwert von 0.2, steigt dann auf 1 (vollständig lichtundurchlässig) und hält diesen Wert über die Dauer der Animation. Dabei bestimmt das Attribut „dur" (duration) diese Dauer. Das Attribut „fill" hat in Animationselementen eine besondere Bedeutung. Hiermit wird festgelegt, ob am Ende der Animation das Element mit dem Anfangswert oder dem Endwert für die Eigenschaft angezeigt werden soll. Der Wert „remove" bedeutet hier, dass der Anfangswert der Animation wieder hergestellt wird, nachdem die Animationsdauer von 10s beendet ist. Mit dem Attribut „begin" legt man den Beginn der Animation fest. Ein möglicher Wert ist ein Ereignis - ein *Event*, wie zum Beispiel das Überfahren eines Elementes mit einem Zeigegerät oder ein Klick auf ein Element. Abbildung 4-12 zeigt das Säulendiagramm aus Abschnitt 4.1.1, welches durch die Verwendung der beschriebenen Animation des Attributes „opacity" erweitert wurde.

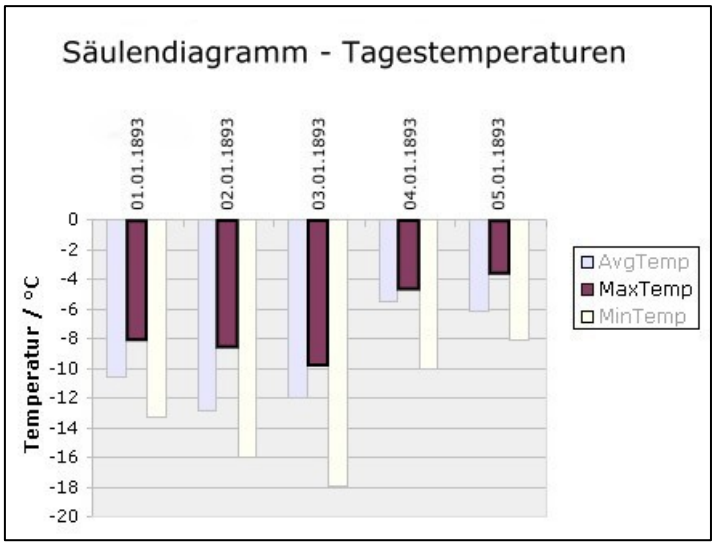

Abbildung 4-11: SVG-Grafik eines Säulendiagramms unter Nutzung des animierten Attributes "opacity"

Hier wurde die Variable „MaxTemp" durch Animation für eine Dauer von zehn Sekunden hervorgehoben. Es ist erkennbar, dass auf diese Weise eine rasches Erfassen der einzelnen Tageswerte dieser Variable möglich wird. Ein visueller Vergleich der einzelnen Tageswerte von „MaxTemp" wird besser unterstützt, als dies ohne die erfolgte nutzergesteuerte Animation möglich wäre. Auch wenn an dieser Stelle nur ein Attribut animiert wurde und die Kombination verschiedener Animationen hier gar nicht betrachtet wurde, wird doch deutlich welchen Nutzen die Möglichkeit der interaktiven Animation mittels mobileSVG bietet: der Anwender ist in der Lage einzelne Bereiche hervorzuheben, also seine Aufmerksamkeit darauf zu fokussieren. Auf diese Weise wird der Prozess des Verständnis und der Analyse von Daten wesentlich vereinfacht.

Natürlich gilt dieses Konzept nicht nur für die betrachteten Säulendiagramme. Es ist problemlos auf weitere Visualisierungsarten übertragbar. Hinsichtlich der Nutzbarkeit dieser Animationsmöglichkeit konnte auf dem für diese Untersuchungen genutzten mobilen Gerät keine nennenswerten Einschränkungen festgestellt werden. Zwar müssen kurze Verzögerungszeiten nach dem Aufruf der Animation akzeptiert werden, diese belaufen sich jedoch im Mittel deutlich unter einer Sekunde. Der erzielte Vorteil ist die bessere Erfassbarkeit und individuellere Nutzersteuerung. Dieser Vorteil überwiegt der entstanden Zeitverzögerung bei der Darstellung. Die Ursache für diese Zeitverzögerung liegt in der

höheren Rechenleistungsanforderung durch die Animation in Verbindung mit der vergleichsweise niedrigen Performance des PDAs gegenüber Desktop- und anderen PCs.

Von Vorteil aus Nutzersicht ist, dass die bisherigen Möglichkeiten der interaktiven mobilen SVG-Nutzung durch Links und durch deklarative Animation gänzlich ohne Programmierkenntnisse dynamisch gestaltet werden konnten. In der Praxis ist es häufig jedoch notwendig SVG-Darstellungen an spezielle, individuelle Funktionalitäten anzupassen. Hier bietet SVG die Möglichkeit Skripte zu integrieren. Diese dritte Möglichkeit der Interaktion soll im mobilen Umfeld mit SVG-Basic[13] an dieser Stelle an einem Beispiel betrachtet werden.

Ausgangspunkt dafür ist, wie bereits bei den Anwendungsfällen zuvor, eine mobile SVG-Visualisierung noch aussagekräftiger zu gestalten. Die dynamische Beschriftung von Informationsobjekten, wie sie zum Beispiel in [Zor04] diskutiert wird, ist dazu eine interessante Möglichkeit. Um die Nutzbarkeit von dynamischen Beschriftungen zu überprüfen, wurde deshalb der *Infotip* als dynamischer Beschriftungsalgorithmus in mobileSVG realisiert. Dieser wurde für mobile Geräte ausgewählt, da er einfach umzusetzen ist und vergleichsweise wenig Rechenleistung erfordert. [Zor04] Beim *Infotip* erscheint ein Label[14], wenn sich der Mauszeiger über das zu beschriftende Objekt bewegt. Er verschwindet, wenn der Mauszeiger den Bereich über dem zu beschriftenden Objektes verlässt. Abbildung 4-13 zeigt einen Ausschnitt der Beschriftung eines Pfades der Parallelen-Koordinaten-Technik mittels Scripting.

---

[13] das SVG-Tiny-Profil bietet diese Möglichkeit in der Version 1.1 nicht

[14] deutsch: Beschriftung

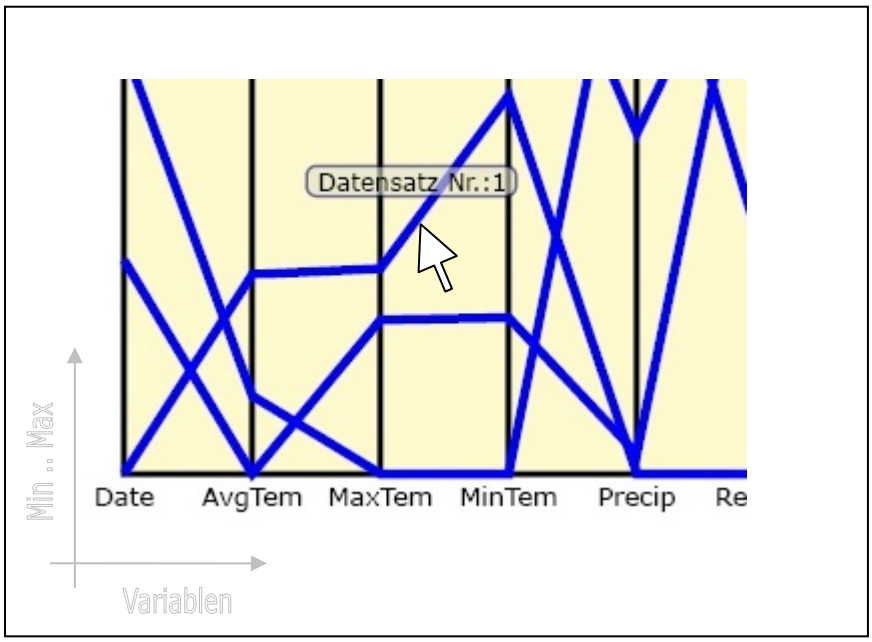

Abbildung 4-12: Pfadbeschriftung durch Scripting bei der Parallelen-Koordinaten-Technik

Das ursprünglich geforderte Ziel, die SVG-Grafik noch aussagekräftiger zu machen, wurde erreicht: Einzelne Pfade sind jetzt leichter zu identifizieren und zu unterscheiden. Durch gezielte Bewegung des Mauszeigers kann man die Beschriftung des jeweiligen interessierenden Datensatzes angezeigt bekommen. Bei diesem Vorgehen tritt aber ein nicht hinnehmbarer Verlust der Steuerung auf! Es war keine akzeptable Interaktion mit der SVG-Grafik mehr möglich. Dies ist jedoch Grundlage einer dynamischen Beschriftung. Im Einzelnen zeigte sich dies dadurch, dass Labels zeitverzögert angezeigt wurden und es dem Nutzer schwer fällt überhaupt eine Zuordnung zum jeweiligen Datensatz vorzunehmen. Einige Labels wurden aufgrund der maximalen CPU-Auslastung des PDAs während des Darstellungsprozesses überhaupt nicht angezeigt. - Bereits bei der Forderung mehr als zehn Pfade der Parallelen-Koordinaten-Technik per Skript dynamisch zu beschriften, war es nicht mehr möglich mit dem PDA vernünftig zu interagieren. Es wurden zwar einzelne Labels mit einer Zeitverzögerung von mehreren Sekunden angezeigt, aber von einer zuverlässig nutzbaren dynamischen Beschriftung kann hier keine Rede sein. Dabei ist der Infotip ein dynamischer Beschriftungsalgorithmus, welcher als wenig rechenzeitintensiv angesehen wird [Zor04]. In der Konsequenz heißt das: Weitere dynamische

Beschriftungsalgorithmen die das Scripting mittels SVG auf mobilen Geräten nutzen, sind kaum realisierbar bzw. nicht praktisch nutzbar. Der Grund dafür ist die mangelnde Rechenleistung und der zu geringe Speicher mobiler Geräte, um für jedes Objekt während der Darstellung festzustellen, ob es ausgewählt (gepicked) wurde oder nicht.

Die Ergebnisse dieses Abschnitts ist zusammenfassend festzustellen, dass die Interaktionsmöglichkeiten von SVG ein wichtiges Hilfsmittel zur Informationsvisualisierung sind. Dazu wurden die Vor- und Nachteile und die Möglichkeiten der Nutzung an passender Stelle diskutiert.

In gewisser Weise erlauben die SVG-Interaktionsmöglichkeiten eine Erweiterung des klassischen Visualisierungskonzeptes (Abb.:2-1). Statt der unidirektionalen Abbildung von Daten auf ein Bild, ist nun eine bidirektionale Abbildung möglich. (Abb.:4-14) Ein Bild, genauer eine Informationsvisualisierung, kann durch die interaktive Verwendung von Hyperlinks, Animation und Skripten jetzt auch die ursprünglichen Daten als Text abbilden. Dabei ist zu beachten, dass mobile Geräte aufgrund ihrer begrenzten Leistungsfähigkeit, nicht immer eine akzeptable Antwortzeit auf Nutzerinteraktionen ermöglichen können. Die Verwendung von Hyperlinks ist in allen Fällen auf mobilen Geräten problemlos möglich. Auch die deklarative Animation ist sowohl auf Klein- als auch auf Kleinstgeräten realisierbar. Die Verwendung von Skripten ist nach SVGT-Spezifikation 1.1 nicht vorgesehen. Auch wenn das SVGB-Profil das Scripting unterstützt, ist dies sehr oft mit einem nicht hinnehmbaren Interaktionsverlust auf mobilen Kleingeräten, wie PDAs, verbunden.

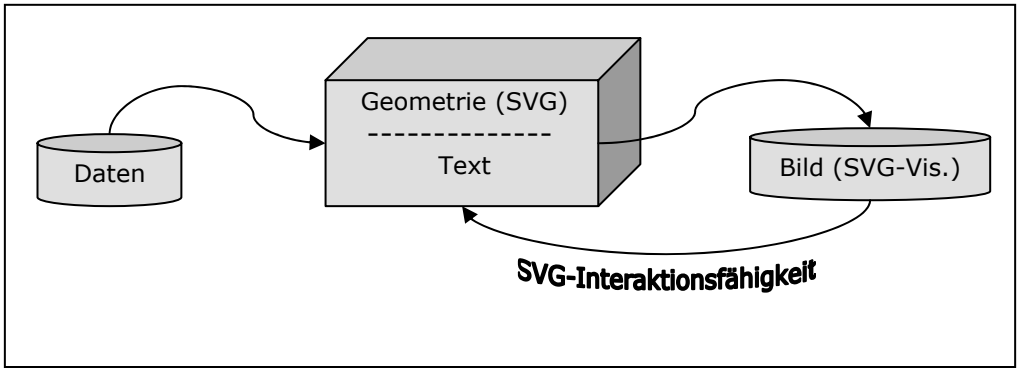

Abbildung 4-13: Schematische Darstellung der Erweiterung einer klassischen Visualisierung durch die Interaktionsfähigkeit von SVG

4.4 Client- vs. Serverseitige Visualisierung

Im bisherigen Verlauf dieser Arbeit wurde davon ausgegangen, dass der gesamte Prozess der Erzeugung einer SVG-Grafik clientseitig, also auf dem mobilen Gerät selbst, erfolgt. Ein Client ist eine Anwendung oder auch ein PC, der in einem Netzwerk den Dienst eines Servers in Anspruch nimmt. Man spricht dann vom Client-Server-Prinzip. Bei einer Client-seitigen Visualisierung werden alle Schritte der Visualisierungspipeline zur Erzeugung einer SVG-Grafik (vgl. Abschnitt 2.1) auf dem mobilen Gerät realisiert. Diese mobilen Geräte unterliegen aber, wie bereits bekannt ist, wesentlichen Einschränkungen gegenüber PCs. Deshalb wird im Folgenden die Frage diskutiert, ob es sinnvoll ist, Teile der Bilderzeugung oder dynamischen Bildgenerierung auf leistungsstärkere PCs auszulagern. Wenn dies so ist, würde der Darstellungsprozess einer SVG-Grafik wesentlich beschleunigt.

Um in einem Netzwerk serverseitig speziell an mobile Geräte angepassten SVG-Inhalt zu erzeugen ist es zunächst erforderlich, dass die serverseitige SVG-Anwendung feststellen kann, von welchem Teilnehmer des Netzwerkes die Anfrage stammt. Stellt ein mobiles Gerät eine solche Anfrage, sollte die serverseitige SVG-Anwendung in der Lage sein, spezielle Anpassungen des SVG-Inhaltes vorzunehmen, um die für das Rendering benötigte Zeit zu minimieren. Dabei sollte die Größe der SVG-Grafik so bemessen sein, dass sie über ein drahtloses Netzwerk verbreitet werden kann, um die mobilen Clients in akzeptabler Zeit zu erreichen.

Die erste Forderung kann zum Beispiel wie folgt gelöst werden: Zur Feststellung, ob eine Anfrage von einem Gerät mit dem Betriebssystemkern Windows CE stammt, welches typisch ist für mobile Geräte, kann die folgende ASP[15]-Syntax verwendet werden:

[15] Active Server Pages

```
var userAgent = new String(Request.ServerVariables("HTTP_USER_AGENT"));
if (userAgent.indexOf("WinCE") >= 0)
{   // mobile WinCE Gerät – passe Inhalt an }
else
{  // benutze Standard SVG 1.1 }
```

Abbildung 4-14: ASP - Code zur Feststellung, ob Serveranfrage von WinCE Gerät stammt [Rob02]

Die Forderung der serverseitigen Anpassung des SVG-Inhalts ist trivial zu erfüllen, da es sich bei SVG um eine standardisierte, von XML abgeleitete Sprache handelt. Wie eine Erzeugung und Manipulation dieser SVG-Dokumente erfolgen kann, wurde in dieser Arbeit bereits deutlich gemacht. Die Anpassung für mobile Geräte sollte dabei die in Abschnitt 4.2.1 erarbeiteten Richtlinien beachten. Die letzte Forderung der akzeptablen Größe einer SVG-Datei zur Verbreitung in einem drahtlosen Netz kann ebenfalls positiv beantwortet werden: Die Größe von SVG-Dateien ist laut Spezifikation des W3C so klein wie möglich ausgelegt. Sie ist in den meisten Fällen wesentlich kleiner als die Größe vergleichbarer Rastergrafikformate [Jac03a]. Hinzu kommt, dass die Größe von SVG-Dateien durch Kompression mittels GZIP weiter verringert werden kann. Der Mehraufwand zur serverseitigen Kompression und clientseitigen Dekompression ist lohnend durch die Ersparnis der Zeit bei der drahtlosen Übertragung [Rob02].

Abschließend kann gesagt werden, dass es möglich und auch sinnvoll ist SVG-Inhalte serverseitig zu erzeugen. Deshalb gibt es bereits viele Ansätze SVG, serverseitig dynamisch zu generieren (z.B. [Wil05]). Für die serverseitige Generierung oder Datenbankanbindung zur clientseitigen Darstellung von SVG auf mobilen Geräten gibt es dagegen weniger Ansätze, obwohl das Prinzip bei beiden gleich ist: eine SVG-Grafik wird "On Demand" generiert, an den Client angepasst und an ihn übermittelt. Dies kann mittels CGI-Script, als Java Servlet aber auch aus Datenbanken und/oder Applikationen erfolgen.

Kapitel 5

Realisierung der Visualisierungsmöglichkeiten

In den vorangegangenen Kapiteln wurden viele Möglichkeiten der Nutzung einzelner SVG-Features zur mobilen Visualisierung betrachtet. In diesem Kapitel soll SVG aus der Praxis der mobilen Informationsvisualisierung heraus betrachtet werden. Dazu erfolgt, jeweils nach einer kurzen Einführung, die Vorstellung von zwei Projekten die im Zuge dieser Arbeit implementiert wurden. Oberstes Ziel ist es dabei, möglichst viele Vorteile und Features von SVG zur mobilen Visualisierung zu vereinen. Anschließend soll bewertet werden, ob und gegebenenfalls wie SVG im mobilen Umfeld als Mittel zur Visualisierung von wissenschaftlichen Daten verwendet werden kann.

5.1 Entwicklungsgrundlagen und Hardware

Zunächst soll kurz dargestellt werden, welche Hard- und Software für beide Projekte zur Verfügung standen. Dies ist nötig, um für den weiteren Verlauf dieses Kapitels eine einheitliche und vor allem transparente Diskussionsgrundlage zur Verfügung zu stellen.
Der Server auf dem die Implementierung erfolgte ist ein Sony PCG-GRT815E, ausgestattet mit einem Pentium 4 Prozessor, getaktet mit 2,8 GHz. Er verfügt über 512 MB RAM sowie eine 64 MB große Grafikkarte. Als mobiles Gerät stand ein PDA des Typs „Compaq iPAQ H3900" zur

Verfügung. Dieser verfügt über einen CPU-Takt von 400 MHz. Als Programm- und Datenspeicher stehen ihm insgesamt 64 MB zur Verfügung. Sein TFT-Display ist in der Lage 65.536 Farben mit einer Auflösung von 240 x 320 Pixeln darzustellen. (siehe auch Tabelle 3-1)

Es ist Teilaufgabe dieser Arbeit zu untersuchen, welche Schnittstelle zwischen einer Visualisierung mit SVG und zu Grunde liegenden Datenbeständen zum Einsatz kommen kann. Dabei sind eine Reihe von Schnittstellen denkbar. Geht man von einer Client-Server-Architektur aus, wie sie im Abschnitt 4.4 behandelt wurde, bietet sich eine Datenbankanbindung per CGI-Programm an. In diesem Fall verbleibt die Logik der Anbindung serverseitig, obwohl sie clientseitig gestartet und gesteuert wird. Datenbankaktionen können ausgelöst und Ergebnisse im Viewer präsentiert werden. Diese Möglichkeit soll an dieser Stelle aus Komplexitätsgründen nur genannt, jedoch nicht weiter betrachtet werden. Eine andere Möglichkeit ist es, eine direkte Datenbankanbindung mit Hilfe einer Schnittstelle in Verbindung mit der Nutzung einer Programmiersprache herzustellen. Dazu seien zwei Softwareschnittstellen genannt (obwohl weitere existieren):

1. ODBC[16] ist eine standardisierte Datenbankschnittstelle, die SQL[17] als Datenbanksprache verwendet. ODBC bietet eine Programmierschnittstelle (API), die es einem Programmierer erlaubt, seine Anwendung relativ unabhängig vom verwendeten Datenbankmanagementsystem (DBMS) zu entwickeln, wenn dafür ein ODBC-Treiber existiert.
2. JDBC[18] ist eine API der Java-Plattform, die eine einheitliche Schnittstelle zu Datenbanken verschiedener Hersteller bietet und speziell auf relationale Datenbanken ausgerichtet ist.

Im Zuge der Implementierungen dieser Arbeit wurden beide Schnittstellen beispielhaft in Verbindung mit der Programmiersprache Java untersucht. Für beide konnte eine mySQL-Datenbank angebunden werden. Für die

---

[16] ODBC: Open DataBase Connectivity

[17] SQL: Structured Query Language

[18] JDBC: Java DataBase Connectivity

Anbindung von Datenbanken im mobilen Umfeld ist dabei die JDBC-Schnittstelle zu bevorzugen. Dies hat folgende Gründe: Obwohl die JDBC-Schnittstelle als langsamer angesehen wird als die ODBC-Schnittstelle von Microsoft, ist sie in Verbindung mit Java auf allen Geräten völlig unabhängig vom verwendeten Betriebssystem. Es muss lediglich eine Java Virtual Machine (JVM) installiert oder installierbar sein. Dies ist für nahezu alle, speziell auch mobile Geräte wie PDAs, der Fall. Es ist also nicht nur möglich die SVG-Dateien auf einem mobilen Gerät mit Hilfe eines mobilen SVG-Betrachters (engl.: Viewer) anzuzeigen, sondern prinzipiell auch die SVG-Dateien direkt auf dem mobilen Gerät, auf der Grundlage gesammelter Daten zu erzeugen. Trotz ihrer beispielhaften Untersuchung fand keine der beiden Datenbankschnittstellen in den folgenden Softwareprojekten Anwendung. Dies hat die Ursache, dass aus Effizienzgründen die zur Verfügung stehenden Rohdaten nicht als eine solche Datenbankquelle vorlagen.

5.2 SVG zur Visualisierung auf mobilen Geräten mittels Paralleler-Koordinaten-Technik

Bereits im Abschnitt 4.1.3 ist die Parallele-Koordinaten-Technik genauer beschrieben worden. Es soll nun Diskussionsgegenstand sein, in welcher Form diese Technik für ein Softwareprojekt zur Visualisierung auf mobilen Geräten einsetzbar ist. Die Forderung an dieses Softwareprojekt ist dabei, dass gesammelte Rohdaten eingelesen werden, falls erforderlich vorverarbeitet werden und schließlich aus der Software heraus eine möglichst effiziente und für mobile Geräte angemessene SVG-Visualisierung erzeugt wird. (vgl. Abb. 5-1) Dazu fand mit Java eine Programmiersprache Anwendung, für die ein gutes SVG Toolkit[19] mit dem Namen „Batik" zur Verfügung steht. Durch die implementierten Module unterstützte es eine

[19] eine Sammlung von Modulen, die zu Effizienzsteigerung oder Vereinfachung der Handhabung von Software führen soll

effiziente Erstellung, Anzeige und Manipulation von SVG-Dateien als Java-Anwendungen oder Applets.

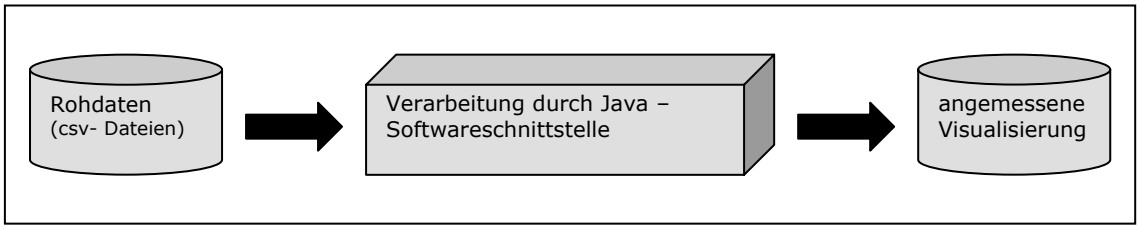

Abbildung 5-1: Workflow zur Visualisierung mittels Java-Softwareschnittstelle

Als Grundlage der Untersuchungen standen Rohdaten in Form von CSV-Dateien[20] zur Verfügung. Dies sind einfach strukturierte Textdateien zur Speicherung oder zum Austausch von Daten. Die einzelnen Werte sind durch ein spezielles Zeichen, zum Beispiel ein Semikolon, voneinander getrennt. Im Speziellen handelte es sich hier um Klimadaten. Das heißt es wurden für einen geographischen Standort die Tagesminimaltemperatur, die Maximaltemperatur sowie weitere Variablen erfasst.

Im ersten Schritt der Implementierungen galt es nun, die gegebenen Rohdaten in geeigneter Weise durch die Java-Softwareschnittstelle einzulesen. Dabei wurden die in der CSV-Datei vorliegenden Werte in 2-dimensionale Arrays eingelesen. Unter Beachtung der in Abschnitt 4.2.1 vorgestellten Optimierungsmöglichkeiten und ihren Auswirkungen auf den Berechnungsaufwand wurden die Rohdaten nach dem Einlesen in Arrays auf zwei Nachkommastellen gerundet.

Im zweiten Implementierungsschritt galt es, sich für eine geeignete Visualisierungstechnik für die Klimadaten zu entscheiden. Die Wahl fiel an dieser Stelle auf die Parallele-Koordinaten-Technik. Der Grund ist die problemlose Verfügbarkeit und Darstellbarkeit der benötigten SVG-Primitive auch auf mobilen Geräten (vgl. Abschnitt 4.1.3)

Im letzten Implementierungsschritt galt es, eine geeignete Visualisierung speziell für mobile Geräte sicherzustellen. Die Forderung nach der intuitiven Erkennbarkeit von Tendenzen und Zusammenhängen in den Merkmalsausprägungen auch auf kleinen Displays mit geringeren

[20] CSV: Character Seperated Values

Auflösungen selbst für große Datensätze stand hier im Vordergrund. Viele einzelne Features zur Erfüllung dieses Ziels wurden bereits im Kapitel vier vorgestellt. Im Test mit realen Datensätzen wurde darüber hinaus festgestellt, dass es sehr sinnvoll ist, eine automatische softwaregesteuerte Visualisierungsanpassung basierend auf der Menge der dargestellten Datensätze durchzuführen. Trotz der qualitätsverlustlosen Möglichkeit der Skalierung ist es zum Beispiel sinnvoll ab etwa 50 Datensätzen automatisch die Strichstärke der Merkmalspfade bis auf einen Minimalwert von 0.2 Pixel herabzusetzen. Ab etwa 500 Datensätzen ist es aber wenig sinnvoll, das Parallele-Koordinaten Diagramm als Ganzes auf dem PDA-Bildschirm betrachten zu wollen. Aufgrund der geringen Auflösung des PDAs macht auch eine weitere Strichstärkenminimierung hier keinen Sinn. Es sollten Interaktionstechniken (vgl. Ab. 4.3) oder etwa der Einsatz des Attributes „viewBox" (vgl. Ab. 4.2.2) erwogen werden.

Leider war die Verwendung des SVG-Toolkits „Batik" der Grund dafür, dass nur die Anzeige der SVG-Datei auf dem mobilen Gerät selbst erfolgen konnte. Durch die Mächtigkeit seiner Module ließ es „Batik" nicht zu, dass die Erzeugung und Manipulation der SVG-Dateien direkt auf dem mobilen Gerät erfolgte. Hierzu ein Zitat von Vincent Hardy, führender Entwickler von Batik aus [Wol03]:

> *„… The SVG specification is over 500 pages long and uses other specs (DOM, CSS) which are also very large. It takes a lot of code to implement SVG 1.0 fully (which is what Batik is doing) and this explains the size. … This also explains the initial class loading time because there are lots of classes to load…*
>
> *… This is why the SVG working group in the W3C (which I am part of) is working on profiles of SVG (we call them SVG Tiny and SVG Basic) with a restricted feature set which should allow leaner implementations. The Batik project may provide an implementation for these profiles at some point, but this has not been discussed or decided yet"*

Trotz der Ankündigung ist bis heute leider keine „Minimalversion" von Batik erschienen, welche ausschließlich mobile Profile unterstützt. Um eine komplette Umsetzung des Workflows nach Abbildung 5-1 auf dem mobilen Gerät zu gewährleisten, hätte deshalb auf den Komfort der fertigen Batikklassen zur Erstellung, Anzeige und Manipulation von SVG-Dateien verzichtet werden müssen, was im Rahmen dieser Arbeit aus Zeitgründen

nicht erfolgte. Die Erfahrungen aus diesem Projekt nutzend, soll im nächsten Abschnitt ein weiteres, praxisnahes Softwareprojekt zur mobilen SVG-Visualisierung vorgestellt werden.

5.3 Das Deichprojekt

Bereits im Jahre 2002 und auch in diesem Jahr 2006 wurde ein großer Teil Deutschlands vom Hochwasser bedroht. Um menschlichen und wirtschaftlichen Schaden zu verhindern, ist es erforderlich sich vor den Gefahren einer drohenden Überflutung zu schützen. Dazu sind Deiche eine geeignete Maßnahme. Diese werden im Fall einer drohenden Überflutung mit Sandsäcken belegt. Im Rahmen eines Universitätsprojektes des Lehrstuhls für Mikroelektronik und Datentechnik an der Universität Rostock wurde in einem Projektseminar ein Konzept erarbeitet, um den Menschen bei der Überwachung eines mit Sandsäcken belegten Deiches zu unterstützen. Dabei werden einige Sandsäcke mit einem Sensorknoten, bestehend aus einem Feuchtigkeitssensor, einem Positionsbestimmungssensor, einem Prozessor, einem Speicher und einer Batterie, ausgestattet. Diese Sensorknoten bilden ein sich AdHoc vernetzendes Sensornetzwerk. Die Sensoren messen fortlaufend den Grad der Feuchtigkeit in den Sandsäcken als Grad der Gefährdung durch einen Deichbruch und übermitteln ihn über das Netzwerk an eine Basisstation. In der Basisstation laufen alle gesammelten Daten zusammen und werden entsprechend ausgewertet. Als Teilaufgabe dieses Projektes galt es die gesammelten Daten der Sensorknoten in der Basisstation aufzunehmen, auszuwerten und in geeigneter Weise zu visualisieren. Die Ergebnisse dieser Teilaufgabe sind nun Gegenstand der Diskussion im weiteren Verlauf dieses Abschnitts. Es wird untersucht, ob und gegebenenfalls wie gut SVG für die Visualisierung einer Echtzeitüberwachung geeignet ist. Dabei sollen Ereignisse zur Laufzeit aufgenommen und entsprechend in eine sich ständig verändernde SVG-Grafik integriert werden. Des Weiteren soll gezeigt werden, in welcher Weise eine Portierbarkeit der SVG-Visualisierung auf mobile Clients möglich ist.

Zunächst galt es eine geeignete Schnittstelle für die Kommunikation der Basisstation mit den Sensorknoten herzustellen. Eine Funkstation empfängt dabei die Signale der Sensorknoten und gibt diese über eine serielle Schnittstelle an die Basisstation weiter, welche in Java implementiert wurde. Bei diesen Signalen kann es sich um Events, etwa in Form einer Alarmmeldung, Positionsmeldungen der Knoten oder etwa um Daten der Feuchtigkeitssensoren handeln. Die Unterscheidung erfolgt in der Basisstation durch Auswertung des Signalheaders. Dieser enthält als Integerwert codiert den jeweiligen Signaltyp. Entsprechend dieser Angaben erfolgt die Verarbeitung der Nachricht in der Basisstation. Ausgangspunkt der Visualisierung ist dabei ein Basisbild, in dem eine Landschaftsdarstellung des zu überwachenden Szenarios erfolgt. In Abbildung 5-2 wurde dieses Basisbild mit Hilfe von Vektorelementen erstellt. Bereits im Abschnitt 2.3 wurde aber erwähnt, dass es auch möglich ist, eine Rastergrafik in ein SVG-Dokument einzubetten. Eine Luftaufnahme des zu überwachenden Deichgebietes im JPEG-Format ist also ebenfalls denkbar und für die Praxis sehr sinnvoll. Ausgehend von solch einem Basis-SVG-Dokument werden ständig aktualisierte Werte eingetragen: GPS-Positionsmeldungen werden umgerechnet und als Knoten an entsprechender Position in der SVG-Grafik dargestellt; Sensorwerte werden ihren Knoten zugeordnet und neben ihnen dargestellt. Abbildung 5-2 zeigt ein solches Szenario, in dem aus Kostengründen zunächst Temperatursensoren anstatt Feuchtesensoren in den Sensorknoten eingesetzt wurden.

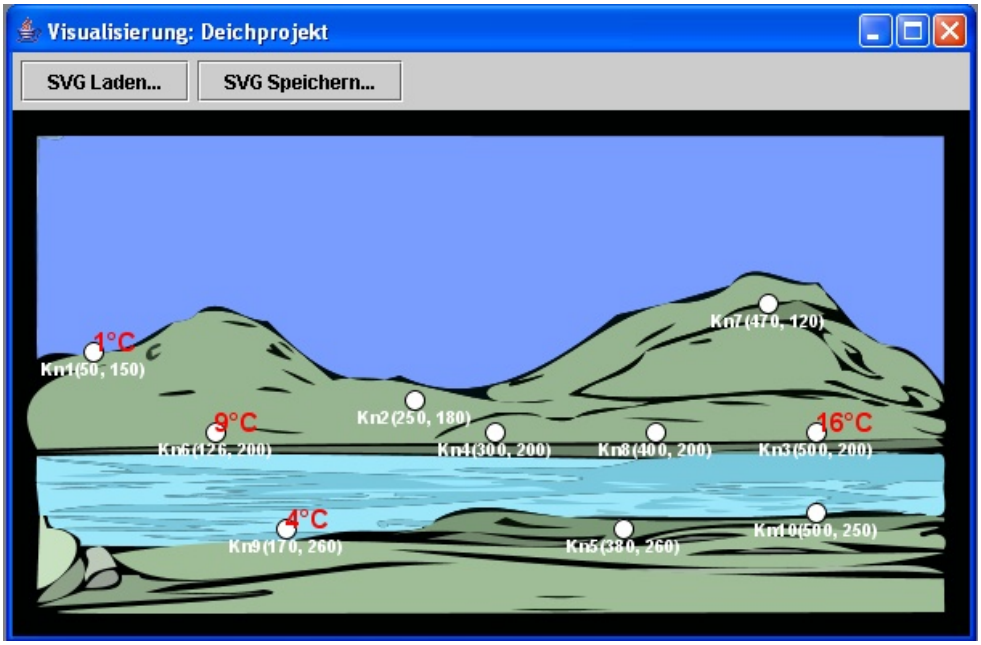

Abbildung 5-2: Visualisierung der Überwachung von zehn Sensorknoten mit einigen Temperatursensorwerten

Nachdem die grundlegenden Anforderungen der Echtzeitvisualisierung der aufgenommenen Daten in einem Fenster erfüllt sind (vgl. Abb. 5-2), soll nun dazu übergegangen werden einige der Vorteile die SVG bietet für dieses Softwareprojekt zugänglich zu machen. Für die Daten der Feuchtigkeitssensoren wurde softwareseitig eine automatische Anpassung der Darstellung zur Laufzeit vorgenommen: Werte im Bereich von 0% – 30% werden grün, Werte im Bereich von 31% - 60% gelb und Werte im Bereich von 61% - 100% Feuchte werden rot dargestellt. Dies hat den großen Vorteil der intuitiven Erfassbarkeit von Gefahrensituationen durch die gewählte Farbkodierung, obwohl zunächst genaue Werte nicht ablesbar sind. Alarmsituationen sind so durch einen Betrachter sofort erkennbar. Abbildung 5-3 zeigt eine solche Alarmsituation im Bereich der Knoten 4, 5 und 9. Um ein genaues Ablesen der einzelnen Feuchtewerte zu gewährleisten, bietet SVG die Möglichkeit der qualitätsverlustfreien Skalierung, wie sie bereits an vielen Stellen angesprochen wurde. Der Nachteil einer solchen Skalierung ist aber, dass man beim hineinzoomen in die Grafik immer die gesamte SVG-Grafik skaliert. Neben der qualitätsverlustfreien Skalierbarkeit der gesamten SVG-Grafik, welche dem Betrachter einer SVG-Grafik immer zur Verfügung steht, wurde deshalb zusätzlich eine Interaktionsmöglichkeit (vgl. Abschnitt 4.3) implementiert.

Dabei soll ein Bereich von besonderem Interesse vergrößert dargestellt werden, ohne dabei den Überblick über den Restbereich der SVG-Grafik zu verlieren. Diese Idee ähnelt dem Ansatz der Fokus+Kontext Techniken, wie sie bereits aus anderen Bereichen der Visualisierung bekannt sind. In SVG wurde diese Idee im Zuge des Deichprojektes als Animation implementiert. Der Anwender ist in der Lage durch Mausbewegungen über die entsprechenden Knoten eine vergrößerte Darstellung der Messdaten des interessierenden Sensorknotens zu erhalten. Er kann genauere Werte einzelner Sensorknoten ablesen, ohne dabei den Überblick über die gesamte SVG-Grafik zu verlieren. Abbildung 5-3 zeigt ein solches Szenario, in dem die Messdaten des Knoten 6 durch eine Maus-Over-Animation vergrößert dargestellt sind.

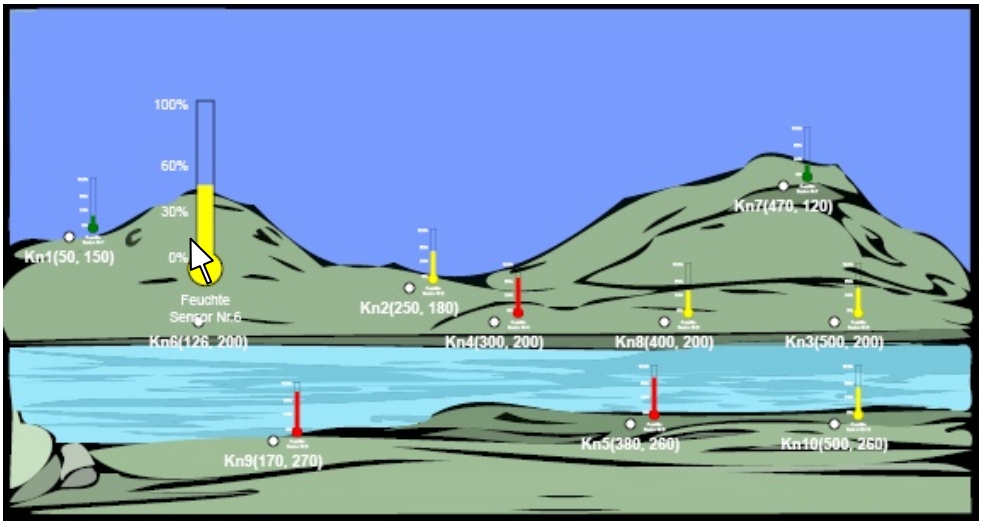

Abbildung 5-3: Verbesserte Echtzeitvisualisierung des Deichprojektes durch eine SVG-Grafik

Abschließend soll nun diskutiert werden, in wie weit es möglich wäre, eine solche Echtzeitvisualisierung auch auf mobilen Basisstationen (PDAs) mit entsprechend verminderter Rechenleistung durchzuführen. Neben der Anbindung eines Funkmoduls und dem Aufnehmen und der Verarbeitung aller eingehenden Nachrichten muss der PDA in der Lage sein, eine sich ständig ändernde SVG-Grafik zeitnah darzustellen.

Wenn der PDA nicht schon intern über einen Funkempfänger verfügt der softwareseitig angesprochen werden kann, sollte es problemlos möglich sein einen entsprechenden Empfänger über die in allen PDAs vorhandene serielle

Schnittstelle anzusprechen. Für die Sicherstellung der Aufnahme und Verarbeitung aller eingehenden Nachrichten ist ebenfalls eine Lösung denkbar. Aufgrund der verminderten Rechenleistung könnte man das Sensornetzwerk so konfigurieren, dass nur in bestimmten minimalen Zeitintervallen Nachrichten gesendet werden dürfen. So stellt man sicher, dass es in der Basisstation nicht zu einer Überlastung durch eingehende Nachrichten kommt. Die letzte Forderung der zeitnahen Darstellung aller eingegangen und ausgewerteten Nachrichten in SVG ist, wenn man allein von der Spezifikation der mobilen Profile ausgeht, ebenfalls möglich. Die Erfahrungen im Rahmen dieser Studienarbeit nutzend, bleibt jedoch festzuhalten, dass wahrscheinlich Einschränkungen im Hinblick auf Interaktionsmöglichkeiten wie etwa Animationen und Skripte unumgänglich wären. Da aus Komplexitäts- und Zeitgründen auch für das Deichprojekt das SVG-Toolkit „Batik" benutzt wurde, kann diese Behauptung jedoch nicht praxisnah belegt werden. Um mobile Geräte dennoch praxisorientiert in einer Weise für das Deichprojekt nutzbar zu machen, wurde der Aspekt von SVG als Dateiaustauschformat abschließend untersucht. Durch die Tatsache, dass SVG eine textbasierte, von XML abgeleitete und standardisierte Sprache ist, kann es auf vielen Plattformen eingesetzt werden. So wurden in das Sensornetzwerk PDAs integriert, welche die Möglichkeit haben eine Nachricht über das Sensornetzwerk an die Basisstation abzusetzen. Darin fordern sie, integercodiert durch den entsprechenden Nachrichtenheader, den aktuellen Systemzustand an. Die Basisstation mit der Echtzeitvisualisierung der SVG-Grafik reagiert auf diese Nachricht. Sie speichert den aktuellen Zustand der SVG-Grafik als SVG-Datei ab und versendet ihn über das Funknetzwerk. Auf diese Weise kann direkt am Standort des jeweiligen PDAs ein aktuelles Bild des Zustandes des Sensornetzwerkes betrachtet werden. Dabei spielt es keine Rolle, welche Programmiersprache auf den PDAs eingesetzt wird, da es sich bei SVG um ein plattformneutrales Dateiformat handelt.

Kapitel 6

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden Einsatzmöglichkeiten von SVG zur Informationsvisualisierung untersucht. Die Nutzbarkeit mobiler Geräte, speziell PDAs, stand dabei im Mittelpunkt. Nach dem Aufzeigen der grundsätzlichen Eigenschaften von SVG im zweiten Kapitel und der mobilen SVG-Profile im dritten Kapitel, wurden im vierten Kapitel konkrete Einsatzmöglichkeiten von SVG zur Informationsvisualisierung betrachtet. Dabei war es das oberste Ziel die Nutzbarkeit von SVG, unter besonderer Beachtung der beschränkten Leistungsfähigkeit mobiler Geräte zu analysieren. Es konnte gezeigt werden, dass SVG ein geeignetes Mittel zur Informationsvisualisierung sein kann. Neben seiner guten Eignung als plattformübergreifendes Dateiaustauschformat, war es vor allem der vektorielle Charakter und die vielen vorgestellten SVG-Features, die es für die mobile Informationsvisualisierung zu einem effizienten Werkzeug machen. Viele dieser Features konnten dabei auch auf mobilen Geräten umgesetzt werden. Zu Problemen kam es immer dann, wenn sehr rechenzeitintensive Operationen auszuführen waren. Die SVG-Interaktionsmöglichkeit durch Skripte war zum Beispiel auf einem PDA nicht oder nur sehr eingeschränkt realisierbar. Für die Zukunft könnte das softwareseitige Rendering auf dem mobilen Gerät durch ein hardwareseitiges Rendering abgelöst werden, um dieses Problem zu umgehen. Die Firma „Bitboys" etwa, welche sich mit graphischen Hardwarelösungen für drahtlose und eingebettete Systeme befasst, bietet dafür einen Vektor-Grafik-Prozessor für mobile Geräte an. Dieser Prozessor mit dem Namen „G12" soll wenig Strom verbrauchen und über 60 Bilder pro Sekunde rendern. Dabei unterstützt er das mobile SVG-Tiny 1.2 Profil,

welches seit Dezember 2005 als Working Draft vorliegt. Dieses Profil ist angepasst auf die ständig wachsende Leistungsfähigkeit mobiler Geräte. So unterstützt es, wenn auch in eingeschränktem Maße, die Verwendung von Skripten. Darüber hinaus bietet es auch viele Neuerungen. So etwa die Möglichkeit, rechteckige Regionen durch Text umhüllen zu lassen. Es bleibt abzuwarten, welche weiteren Neuerungen und Änderungen in die endgültige Spezifikation einfließen werden. Beenden soll diese Arbeit ein Zitat von Antoine Quint, Mitglied und Experte der W3C SVG Working Group [Qui04]:

> *„I'm pretty sure that you will agree that this all sounds pretty good. But the true value of these mobile profiles will only be truly proven in the resonance that service providers leveraging new SVG-introduced graphics and multimedia capabilities for powerful new services will find in the consumer world.“*

Quellen

[Ank00] Ankerst, M., *Visual Data Mining*, Dissertation, Fakultät für Mathematik und Informatik der Ludwig-Maximilians-Universität München, München, 2000 dissertation.de - Verlag im Internet GmbH, Berlin, 2001.

[Bit06] Bitflash Viewer: http://www.bitflash.com/products/vis.asp

[Cap03] Capin, T.: *Mobile SVG–Profiles: SVG Tiny and SVG Basic*, W3C Recommendation, 14. Januar 2003, *http://www.w3.org/TR/2003/REC-SVGMobile-20030114/* Stand: 03/2006

[Fib02] Fibinger, I.: *SVG – Scalable Vector Graphics*, Markt+Technik Verlag, München 2002

[Fow95] Fowler S.L.; Stanwick V.R.: *The GUI style Guide*, Morgan Kaufman Pub, Oktober 1994

[Hav02] *Havre, S.*; et al*:ThemeRiver: Visualizing Theme Changes overTime.*, Proceedings of InfoVis 2000, IEEE Computer Society, Los Alamitos, 2000

[Ins85] Inselberg, A.: The Plane with Parallel Coordinates, The Visual Computer, 1985

[Int06] Intesis eSVG-Viewer: www.embeddedsvg.com

[Jac03] Jackson, D. et al: *Scalable Vector Graphics 1.1 Specification*, W3C Recommendation, 14. Januar 2003, *http://www.w3.org/TR/SVG/* Stand: 10/2005

[Jac03a] Jackson, D. et al: *Scalable Vector Graphics 1.1 Specification*, Appendix J: Minimizing SVG File Sizes, 14. Januar 2003, *http://www.w3.org/TR/SVG/* Stand: 03/2006

[Mot06] http://direct.motorola.com/ger/Web_full_specs.asp?Country=DEU&language=GER&productid=29249&strPrimaryOption=FS&lSecondaryOption=-3

[Nok06] http://www.nokia.de/de/mobiltelefone/modelluebersicht/e61/startseite/185058.html

[Oel02] Oellin, F.: *Algorithmen und Applikationen zur interaktiven Visualisierung und Analyse chemiespezifischer Datensätze*, Dissertation, Naturwissenschaftlichen Fakultäten der Friedrich-Alexander-Universität Erlangen-Nürnberg, 2002

[Qui04] Quint, A.: *Going mobile with SVG: Standards*, XML Kolumne, http://www.xml.com/pub/a/ 2004/06/16/mobilesvg.html, Juni 2004

[Rob02] Robinson, B.: *Creating and Implementing Mobile SVG*, SVG Open, Zürich 2002

[Sch00] Schumann, H.; Müller, W.: *Visualisierung*, Springer Verlag, Berlin Heidelberg 2000

[Sta00] Stasko, J; Zhang, E.: *Focus+Context Display and Navigation Techniques for Enhancing Radial, Space-Filling Hierarchy Visualization*, paper, InfoVis 2000

[Wil05] Williams, J.; Neumann, A.: *Dynamic Loading of Vector Geodata for SVG Mapping*, paper, version 1.1.2, 10/2005 http://www.carto.net/papers/svg/postgis_geturl_xmlhttprequest/

[Wol03] Wolfschläger, T.: *Implementierung einer horizontalen Baumdarstellung als JavaBean unter Verwendung des SVG (XML) Grafikformates*, Diplomarbeit, Konstanz April 2003

[Zor04] Zornow, M.: *Entwicklung eines Konzeptes zur angemessenen Beschriftung von Informationsobjekten*, Bachelorarbeit, Rostock 2004

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Abschnitte, die wörtlich oder annähernd wörtlich aus einer Veröffentlichung entnommen sind, als solche kenntlich gemacht habe, ferner, dass die Arbeit noch nicht veröffentlicht und auch keiner anderen Prüfungsbehörde vorgelegt worden ist.

Rostock, den 30. Mai 2006