Flash Weather

Vektorisierung von raum- und zeitbezogenen Daten zur Visualisierung mit Macromedia Flash

Diplomarbeit

im Fachbereich Mathematik/Informatik der Universität Osnabrück

betreut von

Prof. Dr. Oliver Vornberger
Prof. Dr. Horst Malchow

Benjamin Stark

Osnabrück, im Februar 2001

Vorwort

Diese Diplomarbeit entstand an der Universität Osnabrück im Fachbereich Informatik. Sie bildet den schriftlichen Teil meiner Prüfung zum Diplom-Systemwissenschaftler.

Das behandelte Thema kam auf Anregung von Herrn Prof. Dr. Vornberger zustande. Auf die Gestaltung der Arbeit nahmen mein langjähriges Interesse in Softwareentwicklung und Internettechnologien Einfluss.

Danksagung

An dieser Stelle möchte ich folgenden Personen für ihre Unterstützung bei der Erstellung dieser Diplomarbeit danken:

- Herrn Prof. Dr. Oliver Vornberger für die Betreuung der Arbeit, seine Hinweise und Anregungen und die Beschaffung der Wetterdaten,
- Herrn Prof. Dr. Horst Malchow für die Betreuung der Arbeit seitens der Systemwissenschaftler,
- Olaf Müller, Carsten Wilhelm, Stefan Rauch und Ralf Kunze für konstruktive Vorschläge und das Korrekturlesen dieser Arbeit,
- Michael Dirska für die Bereitstellung eines Computers und der benötigten Software im Fachbereich Informatik.

Warenzeichen

In dieser Arbeit werden eingetragene Warenzeichen, Handelsnamen und Gebrauchsnamen verwendet. Auch wenn diese nicht als solche gekennzeichnet sind, gelten die entsprechenden Schutzbestimmungen.

Die Verwendung des Macromedia Flash file format (SWF) Software Development Kit (SDK) unterliegt den Bestimmungen des Macromedia Flash File Format (SWF) SDK License Agreement. Apache Xerces und Apache Xalan unterliegen den Bestimmungen der Apache Software License. Die Verwendung des Saxess Wave Flash-Generators unterliegt den Bestimmungen der Saxess SoftwareDesign Licence.

Inhaltsverzeichnis

Realisierung und Anwendung

Resumee und Ausblick

Anhang

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1 Einleitung

Das Internet. Unendliche Weiten? Zumindest ein Schlagwort, das in aller Munde und allen Medien ist. Kaum eine Information, die nicht im Internet beziehungsweise im WorldWideWeb - der bunten und neben E-Mail am häufigsten genutzten Anwendung des Internets - zu finden ist.

Schon lange existieren Suchdienste wie Altavista und Yahoo. Und noch länger sind Universitäten und wissenschaftliche Einrichtungen über das Internet miteinander verbunden, die zu den ersten gehörten, zwischen denen vor allem in den USA ein Computer-Netzwerk aufgebaut wurde, aus dem sich später das heutige Internet entwickelte. Während dieser Entstehungsphase des Internets gab es nur sehr rudimentäre Dienste wie Email, Telnet und FTP, doch die Entwicklung von HTML (Seitenbeschreibungssprache für des WorldWideWeb) im Jahr 1989 vereinfachte das Darstellen, Verknüpfen und Auffinden von Informationen im Netzwerk dank miteinander verbundenen Dokumenten und deren Darstellung in einem Webbrowser. Heute ist es weder für Firmen noch für Privatpersonen ein technisches oder finanzielles Problem, sich eine Präsenz im WorldWideWeb zu beschaffen. Innerhalb der letzten zehn Jahre entstand so ein weltumspannendes Netzwerk und zur Zeit sind weltweit mehr als 100 Millionen Computer über das Internet miteinander verbunden [Matr2000].

Die Möglichkeiten der Informationspräsentation waren im WorldWideWeb zu Beginn relativ eingeschränkt, weshalb dem Wunsch nach weiteren Fähigkeiten von HTML mit der Einführung von Tabellen, Frames, JavaScript und animierten GIF-Grafiken nachgekommen wurde. Zu einem Universalwerkzeug macht den Webbrowser aber erst dessen sogenannte Plugin-Schnittstelle. Eine Schnittstelle, die es ermöglicht, spezielle Darstellungsprogramme für beliebige Dateiformate wie z.B. Audio und Video in den Browser zu integrieren - daher der Name Plugin. Der Nachteil dieser Plugins ist aber, dass sie den Webbrowser zwar befähigen, neben HTML und einigen Grafikformaten ein weiteres Fremdformat (eventuell eingebettet in eine Webseite) darzustellen, jedoch zuerst aus dem Internet heruntergeladen und installiert werden müssen - ein Vorgang der vom Endbenutzer gerne vermieden wird.

Wie sieht es heute im WorldWideWeb aus? Wirft man zum Beispiel einen Blick auf die Webseiten diverser Wettervorhersagedienste[1], so erhält der Informationssuchende zumeist eine sehr ähnliche Darstellung: Eine textuelle Version der Vorhersage wird veranschaulicht durch einige Grafiken mit Icons, die die Wetterlage darstellen,

[1] siehe z.B. www.wetter.de, www.donnerwetter.de, de.weather.yahoo.com

und Gradzahlen für die Temperatur an einigen Punkten der Karte. Eventuell findet sich auch eine animierte Grafik, die dann eine Art Diashow des vorhergesagten Wetters präsentiert. Werden genauere Informationen zu einem bestimmten Gebiet oder einer Stadt gesucht, findet sich schnell ein Link und eine regionale Vorhersage erscheint. Die Information ist gefunden, doch ihre grafische Darstellung ist eher sparsam gehalten. Der gezeigte Ausschnitt lässt sich nicht vergrößern oder verkleinern, die Darstellung der Information lässt sich nicht beeinflussen, ein benutzergesteuertes Ein- und Ausblenden von Informationen zu Temperatur, Niederschlag, Luftdruck und Bewölkung wird nicht angeboten. Dabei ist die fehlende individuelle Darstellung der Wetterkarte nicht unterlassene Datenaufbereitung des Webseitenanbieters, sondern liegt zum Großteil an mangelnden Darstellungsmöglichkeiten der herkömmlichen Grafikformate.

Standardmäßig können Webbrowser, wie der Netscape Navigator oder der Microsoft Internet Explorer, zweidimensionale Grafiken der Formate GIF, JPG und die neueren Versionen eventuell PNG anzeigen. Dies alles sind pixelorientierte Grafikformate, die außer einer Diashow mittels animiertem GIF keinerlei Animation und erst recht keine Interaktion mit dem Benutzer unterstützen. Browsererweiterungen existieren für das Zoomen von Grafiken auf Webseiten, jedoch zeigen pixelorientierte Grafiken - wie schon der Name sagt - bei zunehmender Vergrößerungsstufe recht bald ihre Zusammensetzung aus einzelnen Pixeln. Das Bild verliert an Qualität. Ein Ausweg wäre das sukzessive Nachladen von neuen Webseiten, die den gewünschten Bereich vergrößert oder verändert darstellen, wie es bei vielen Stadtplänen[2] zu finden ist. Möchte man aber diesen ständigen Datenverkehr vermeiden, bleibt nur ein zoombares d.h. vektororientiertes Grafikformat.

Bei einem vektororientierten Grafikformat setzt sich das Bild nicht aus einzelnen farbigen Bildpunkten zusammen, sondern die Information über den Bildaufbau wird mathematisch beschrieben. Linien haben einen Anfangs- und einen Endpunkt, eine Linienfarbe und -stärke. Polygone und Kreise bekommen zusätzlich eine Füllfarbe. Beim Vergrößern eines Ausschnitts geht die Qualität nicht verloren, da die Bildinformation nicht statisch vorliegt, sondern die pixelbasierte Darstellung der Grafikobjekte, die für die Wiedergabe per Monitor oder Drucker benötigt wird, erst zum Zeitpunkt der Darstellung berechnet wird.

Ein Grafikformat, das genau diesen Vorteil bietet und auch die oben erwähnte Animation und Interaktion mit dem Benutzer unterstützt, ist Macromedia Shockwave Flash - oder kurz Flash. Doch gehört es nicht zu den Grafikformaten, die intern von

[2] siehe z.B. www.braunschweig.de → Stadtplan

den gängigen Webbrowsern unterstützt werden. Man benötigt also ein Plugin zur Darstellung dieser Grafiken. Dies ist ein Nachteil.

Microsoft und Netscape liefern aber mitsamt ihrer aktuellen Browser-Versionen das Plugin für Macromedia Shockwave Flash-Grafiken mit. Damit sprechen nun alle oben genannten Gründe für eine Verwendung von Flash, da es - zumindest bei Webbrowsern ab der vierten Generation - wie ein standardmäßig unterstütztes Grafikformat verwendet werden kann. Ein weiterer Grund für die Verwendung von Macromedia Shockwave Flash ist, dass das Flash-Plugin für jedes nennenswerte Betriebssystem (Windows, Mac, Linux) verfügbar ist und die Darstellung der Grafiken, Animationen und Texte durch die Verwendung eines Plugins auf jedem System und bei jeder Bildschirmauflösung gleich aussieht - was bei einfachen HTML-Seiten und vor allem auch bei Grafiken nicht der Fall ist.

Problemstellung

Seit Macromedia 1998 das Dateiformat von Shockwave Flash offen gelegt und gleichzeitig ein Software Development Kit bereitgestellt hat, können Flash-Grafiken über eine Programmierschnittstelle erzeugt werden, ohne die eigentliche grafische Entwicklungsumgebung namens Flash Studio zum Erstellen solcher Grafiken verwenden zu müssen. Damit dies aber nicht zur Folge hat, dass jede Flash-Grafik quasi programmiert werden muss, existiert ein dem Software Development Kit ähnliches Produkt, dass statt der Programmierschnittstelle eine auf XML (eXtensible Markup Language) basierende Textschnittstelle anbietet. Die Eingabedaten im definierten XML-Format werden mit Hilfe eines Parsers in Flash-Grafiken umgewandelt.

Die Aufgabenstellung dieser Diplomarbeit ist die komplette Vorbereitung der Eingabedaten für den Prozess der Erzeugung von Flash-Grafiken. Ausgehend von Wettervorhersagedaten des Deutschen Wetterdienstes und geografischen Kartendaten sollen Schnittstellen und Module entwickelt werden, die die binären Daten auslesen und verwalten können, eine geografische und computergrafische Bearbeitung ermöglichen und das Ergebnis schließlich in einem frei konfigurierbaren XML-Format ausgeben. Die Datenausgabe und -formatierung stützt sich daher sehr stark auf die XML-Technik.

Ein Schwerpunkt der computergrafischen Bearbeitung der Daten wird die Umwandlung der im Rasterformat vorliegenden Wettervorhersagedaten in das Vektorformat sein.

Die vorliegende Diplomarbeit beschäftigt sich exemplarisch mit der Darstellung von Wetterprognosedaten. Die Erkenntnisse sollen jedoch auf die Vektorisierung

beliebiger Rasterdaten und die Visualisierung beliebiger raum- und zeitbezogener Daten übertragbar sein.

Aufbau der Arbeit

Im ersten Abschnitt werden die Grundlagen dieser Arbeit behandelt und erste Ansätze angesprochen. Kapitel 2 beginnt mit der Vorstellung von Macromedia Flash und zeigt seine Vorteile auf. In diesem Zusammenhang kommt ebenfalls das Flash-Dateiformat sowie das Software Development Kit zur Sprache, um den Aufbau von Flash-Grafiken und ihre Erzeugung darzustellen. Kapitel 3 widmet sich dem Thema XML und erklärt, warum die eigenen Datenformate darauf basieren werden. Aufgrund der Aktualität des Themas und des bisher noch geringen Bekanntheitsgrades von XML wird es hier ausführlich beschrieben. Kapitel 4 enthält Überlegungen zu möglichen Eingabeformaten, aus denen Flash-Grafiken entstehen können und beschreibt exemplarisch die vorhandenen Wettervorhersage- und Kartendaten. Die benötigten Schritte für eine Überführung der Daten in das auf XML basierende Format, das dem Flash-Grafik erzeugenden Parser als Input dient, werden kurz vorgestellt. Der erste Abschnitt schließt mit der Betrachtung der Wetterkartendarstellung.

Der zweite Abschnitt der Arbeit geht auf die Realisierung der Aufgabenstellung ein. Kapitel 6 betrachtet den Datenfluss und beschreibt die selbst entwickelten Klassen und Pakete und deren Zusammenspiel. Einen Schwerpunkt stellt dabei der Algorithmus zur Umwandlung von Raster- in Vektordaten dar. Die Verknüpfung der Module zu einer Anwendung und die Konfiguration derselben beschreibt Kapitel 7.

Den Schluss dieser Arbeit bilden Resumée und Ausblick. Dabei werden im Resumée (Kapitel 8) die während der Bearbeitung dieser Diplomarbeit gewonnenen Erkenntnisse genannt und die erreichten Ziele bewertet. Kapitel 9 spricht einige der Ideen an, die nicht verwirklicht werden konnten, und nennt Möglichkeiten der Fortführung und Erweiterung dieser Arbeit.

Im Anhang befinden sich eine Auflistung der benötigten Software für die Verwendung der entwickelten Module und deren Konfiguration, Beschreibungen der Dateiformate, die Klassenhierarchie der eigenen Pakete und eine Erläuterung zum Inhalt der beigefügten CD-ROM.

I Grundlagen und erste Ansätze

2 Macromedia Flash

Macromedia Flash wurde im Rahmen dieser Diplomarbeit aus guten Gründen zur Visualisierung von Wetterdaten gewählt. Das folgende Kapitel beleuchtet seine Vor- und Nachteile, untersucht das Dateiformat und geht auf verschiedene Möglichkeiten ein, Flash-Grafiken über eine Programmierschnittstelle zu erzeugen. Zuerst aber einige Informationen zur Entstehungsgeschichte von Macromedia Flash.

Das Softwareunternehmen Macromedia mit Sitz im kalifornischen San Francisco entstand im Jahr 1992 durch den Zusammenschluss der Firmen MacroMind und Authorware. Drei Jahre später konnte Macromedia seinen Kundenstamm erstmals bedeutend vergrößern - durch die Übernahme von Altsys, den Hersteller des Grafikprogramms Freehand.

Ebenfalls 1995 veröffentlichte die Firma FutureWave aus San Diego das vektororientierte Illustrationsprogramm SmartSketch und ein dazugehöriges Plugin namens FutureSplash-Player, um die mit SmartSketch erzeugten Illustrationen mit einem Webbrowser betrachten zu können. Aufbauend auf SmartSketch entwickelte FutureWave 1996 das Animationsprogramm FutureSplash-Animator - den Vorläufer der heutigen Entwicklungsoberfläche Flash Studio.

Ende 1996 übernahm Macromedia die Firma FutureWave. Die Produkte FutureSplash Animator und FutureSplash-Player wurden unter dem Namen Flash und Shockwave Flash-Player weiterentwickelt. Flash 1 und 2 erschienen daraufhin im Jahr 1997, weitere Versionen wurden jeweils im Abstand etwa eines Jahres ausgeliefert. Die vorerst letzte und fünfte Version von Flash existiert seit dem Sommer 2000.

Macromedia hat im Dezember 2000 weltweit mehr als 1200 Mitarbeiter und entwickelt außer Flash weitere Softwareprodukte im Bereich Webauthoring und Grafikerstellung (z.B. Macromedia Dreamweaver, Macromedia Fireworks). Laut einer unabhängigen Studie arbeiten mehr als eine Million professionelle Entwickler mit den Produkten von Macromedia. [Macr2000a]

Aber was ist so herausragend an Macromedias Produkten, was sind die Vorteile von Flash gegenüber herkömmlichen Grafikformaten?

2.1 Vorteile von Flash

Macromedia beschreibt sein Produkt Flash beziehungsweise dessen Dateiformat Shockwave Flash (SWF) folgendermaßen: „Flash ist der Standard für interaktive Vektorgrafiken und Animationen im Web.“ [Wolt1999]

Die Aussage zum erlangten Standard diskutiert der Abschnitt 2.2, doch was zeichnet Vektoren aus und warum ist Flash gerade für das Internet so gut geeignet?

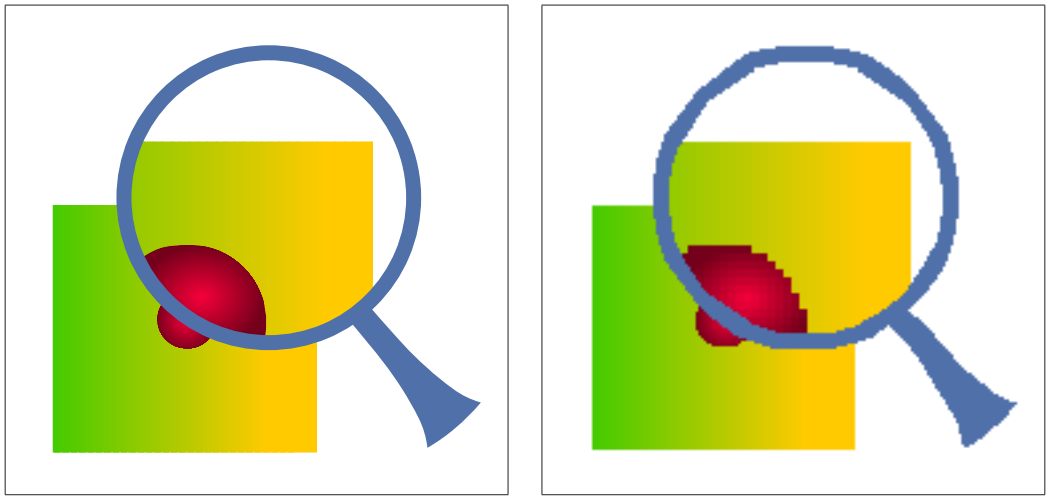
Shockwave Flash ist ein Grafikformat für zweidimensionale Vektorgrafiken, in das aber ebenfalls pixelbasierte Grafiken (sogenannte Bitmaps) eingebunden werden können. Die Verwendung von Vektoren bringt eine Reihe von Vorteilen mit sich. Während Pixelgrafiken aus einer Matrix von Bildpunkten (Pixeln) bestehen und Informationen zur Farbe jedes Pixels enthalten, werden Vektorgrafiken mathematisch durch Vektoren und deren Eigenschaften beschrieben. Für eine Linie wird ein Anfangs- und ein Endpunkt, die Linienfarbe und ihre Stärke benötigt. Eine Kurve besitzt außerdem eine Krümmung, ein Kreis wird durch seinen Mittelpunkt und Durchmesser beschrieben. Die von Vektoren eingeschlossenen Flächen können mit Farben oder Farbverläufen gefüllt werden.

Pixelbasierte Grafiken haben den entscheidenden Nachteil, dass sie nur für eine einzige Auflösung erzeugt wurden, sie sind auflösungs*abhängig*. Vergrößert man den Ausschnitt einer Pixelgrafik, werden auch die einzelnen Pixel größer. Das Bild stellt sich wie ein Mosaik dar. Auch intelligente Anzeigetechniken, die in der Vergrößerungsstufe versuchen, durch Interpolation, d.h. das Einfügen von Pixeln mit Übergangsfarben, die Auflösung der Pixelgrafik zu erhöhen, bringen keine nennenswerten Verbesserungen.

Vektorbasierte Grafiken hingegen können in jeder Vergrößerungsstufe verlustfrei dargestellt werden, sie sind auflösungs*unabhängig*. Ihre Anzeige auf einem Monitor oder ihre Wiedergabe per Drucker ist zwar ebenfalls pixelbasiert, da diese Ausgabegeräte ein Gitter aus Farbpunkten steuern, die Darstellung der Vektorgrafik wird jedoch erst auf Clientseite, also für die entsprechende Auflösung und den ausgewählten Bereich, berechnet. So können auch vergrößerte Ausschnitte von Grafiken die maximale Auflösung des Ausgabegerätes nutzen.

Abbildung 2.1 veranschaulicht den Unterschied zwischen vektor- und pixelbasierten Grafiken an einem Beispiel.

Ein weiterer Vorteil der in Flash verwendeten Vektortechnologie ist, dass Dateien mit Vektorgrafiken in der Regel kleiner als pixelbasierte Grafiken sind. Da nicht die Informationen jedes einzelnen Bildpunktes, sondern nur die Informationen der



(a) Vektorgrafik (Flash, 728 Bytes) (b) Pixelgrafik (PNG, 2413 Bytes)

Abbildung 2.1: Vergleich von vektor- und pixelbasierter Grafik

Linien und Kurven sowie deren Farbe, Strichstärke und Füllung übertragen werden müssen, sind Flash-Dateien außerordentlich klein. Dies kommt einem ernsten Problem des Internets entgegen: dessen begrenzte Bandbreite. Die Bandbreite beschreibt die Menge an Informationen, die pro Sekunde durch ein Netzwerk gelangt. Je geringer die Bandbreite, desto länger dauert es, bis zum Beispiel eine Webseite aus dem Internet geladen ist und angezeigt werden kann.

Für die Darstellung von Flash-Grafiken benötigt man auf jedem System den Flash-Player, den es als Webbrowser-Plugin oder als Standalone-Version gibt. Diese Einschränkung löst letztlich eines der größten Probleme von HTML: Die in Flash erstellten Grafiken werden bei beliebiger Auflösung auf mehreren Plattformen (Windows, Macintosh, Solaris, Linux) identisch dargestellt, da der einheitliche Flash-Player die Darstellung steuert. Sogar die Verwendung beliebiger Schriftarten stellt kein Problem dar, denn Texte wandelt Flash in Schriftkonturen um und fügt sie anschließend wie Grafikobjekte der Flash-Datei hinzu. Die dem Original zu Grunde liegenden Schriftarten müssen deshalb nicht auf dem Clientcomputer existieren.

Das Grafikformat Shockwave Flash ist also auflösungsunabhängig, plattformunabhängig und bandbreitenschonend. Macromedia hat Flash aber nicht nur als Vektorgrafikformat entwickelt, sondern „für interaktive Vektorgrafiken und Animationen“. Was ist mit Flash noch möglich?

Wahrscheinlich wäre es leichter, die Frage danach zu beantworten, was Flash nicht kann. Anfangs wurden mit Flash lediglich interaktive Buttons und ähnliche Navigationselemente erstellt, da die langsamen Internet-Verbindungen zur Zeit der

ersten Versionen noch nicht dazu geeignet waren, vollflächige Animationen fließend zu übertragen. Inzwischen setzen sich auf Grund der günstigen Preisentwicklung schnellere Modems, ISDN und DSL immer mehr durch. Seitdem erstellen zahlreiche Internetagenturen komplette Websites, Produktpräsentationen und Werbespots in Flash. Vektor- und Bitmaptransparenz, komplexe Navigationen, problemlose Einbindung von Sound im kompimierten MP3-Format und der Einsatz beliebiger Schriftarten werden für Effekte aller Art genutzt: Überblendeffekte, Bewegung und sogar Morphing sind möglich geworden. Man spricht deshalb auch von „Flash-Filmen“.

Für Interaktionen mit dem Benutzer stehen Texteingabefelder und eine Skriptsprache namens ActionScript zur Verfügung. Aktionen können an bestimmte Bilder einer Filmsequenz oder an Buttons gekoppelt werden und den Ablauf der Animation steuern, neue Filme beziehungsweise Webadressen laden oder Eigenschaften der momentanen Grafikobjekte verändern.

Alle diese Elemente einer Animation liegen bei Flash in einer einzigen Datei, die zudem streamfähig ist. Das heißt, dass der Benutzer die ersten Bilder der mit Flash erzeugten Animationen bereits verfolgen kann, während die restlichen Bilder noch im Hintergrund geladen werden.

Eine Animation entsteht entweder durch Morphing, das ist das Überblenden eines Grafikobjekts in ein anderes, wobei der Computer die Zwischenschritte berechnet, was die Dateigröße weiter verringert, oder durch zeitbasierte Veränderungen. Die Steuerung der Animationsphasen erfolgt über die in Multimedia-Software üblichen Keyframes - dabei beschreiben Schlüsselbilder die Animationsphasen auf der Zeitleiste. Bewegung entsteht durch Interpolation zwischen den Keyframes. Flash Interessante Perspektiven eröffnet die mit Flash 3 eingeführte variable Transparenzfunktion, die auch Ein- und Ausblendungen ermöglicht.

Macromedia Flash - „der Standard für interaktive Vektorgrafiken und Animationen im Web“? Nach allen Vorteilen, die Flash offensichtlich zu bieten hat, gibt es auch ein paar Einschränkungen.

Es ist sicherlich möglich, mit Flash ansprechende und animierte Webseiten zu erstellen, doch die Gestaltung eines HTML-Dokuments mit eingebundenen Pixelgrafiken ist weniger aufwändig und auch kostengünstiger. Außerdem ist die zu transferierende Datenmenge bei Flash-Dokumenten immer noch um ein Vielfaches höher als die für ein reines HTML-Dokument, wenn nur textbasierte Informationen übertragen werden sollen.

Folgender Vergleich zeigt die möglichen Mehrkosten einer mit Flash gestalteten

Webpräsenz auf: HTML-Editoren stehen häufig kostenlos zur Verfügung (z.B. Netscape Composer). Das Autorensystem Flash zum Entwickeln von Flash-Animationen kostet dagegen mehrere hundert Mark. Dynamisch erzeugte Webseiten lassen sich z.B. mit einem Apache Webserver und der Skriptsprache PHP auf einem Linux-Rechner komplett als Freeware-Lösung realisieren, das Serverprodukt Macromedia Generator ermöglicht die dynamische Erzeugung von Flash-Filmen wie Balkendiagrammen, Tortengrafiken und News-Tickern, kostet aber mehrere tausend Mark.

Des Weiteren ist bei Flash zu bedenken, dass es bei alten und leistungsschwachen Rechnern zu ruckelnden Animationen kommt. Die Darstellung eines Flash-Filmes, also die Umwandlung der mathematischen Beschreibung der Grafikobjekte in Bildschirmpunkte, benötigt wesentlich mehr Rechenleistung als die Interpretation einer HTML-Datei. Bei der heute für Computer typischen Ausstattung ist diese Rechenleistung jedoch im Überfluss vorhanden und wird während des "Surfens" sowieso nicht benötigt.

2.2 Internet-Standard Flash

Die Verwendung von Macromedia Flash eröffnet neue Möglichkeiten des Webdesigns. Shockwave Flash ist aber noch ein relativ neues Dateiformat und daher nicht so verbreitet wie die herkömmlichen Grafikformate JPG oder GIF. Die wenigsten Bildverarbeitungsprogramme können bisher mit der Dateiendung SWF etwas anfangen. Ohne die Sicherheit des Entwicklers, dass zumindest ein Großteil der Endbenutzer die Flash-Animationen betrachten kann, wäre es fatal, dieses Format im WorldWideWeb zu verwenden. Das Angebot des Internets ist so vielfältig und groß, dass eine nicht darstellbare Webseite vom Benutzer einfach ignoriert wird. Sollte erst noch der Download eines Plugins nötig sein, um den Inhalt der Seite vollständig zu Gesicht zu bekommen, sinkt die Akzeptanz eventuell soweit, dass die Webseite verlassen wird.

Macromedia gibt an, dass im Dezember 2000 96,4% der weltweiten Internetbenutzer das Flash-Plugin auf ihrem Computer installiert hatten. Die unabhängige Marktforschungsfirma NPD Online kam bei einer Befragung von 2.634 Personen zu diesem Ergebnis. [Macr2000b]

Tabelle 2.1 zeigt deutlich, dass die 96,4% Verfügbarkeit, die Macromedia angibt, jedoch nur für die USA und nur für Flash-Filme gilt, die abwärtskompatibel zur Version 2 sind. Daher ist diese Aussage sehr kritisch zu betrachten, denn die einzelnen Flash-Versionen bauen zwar aufeinander auf, unterscheiden sich aber teilweise

Weltweite Verfügbarkeit nach Version	Flash 2	Flash 3	Flash 4	Flash 5
USA	96,4%	94,7%	86,1%	32,1%
Kanada	96,5%	95,2%	91,0%	35,9%
Europa	96,8%	94,9%	89,8%	45,3%
Asien	95,5%	93,7%	86,5%	41,4%
Südamerika	95,5%	92,5%	89,6%	43,3%

Tabelle 2.1: Verfügbarkeit des Macromedia Flash-Players (Dezember 2000)

erheblich. Die dritte Version führt z.B. Vektortransparenz, Morphing und Actions ein. Flash 4 erweitert die Skriptsprache ActionScript, bringt Eingabetextfelder zur Erstellung von Formularen mit und kann das inzwischen sehr populäre Audioformat MP3 verarbeiten. Die Neuerungen von Flash 5 betreffen schließlich vor allem die Bedienfreundlichkeit von Flash. Die grafische Entwicklungsumgebung ähnelt nun dem Erscheinungsbild anderer Macromedia-Produkte und die ActionScript-Syntax wurde an das bekannte JavaScript angepasst und um einen Debugger erweitert.

Deshalb sollte man sicherheitshalber vom Flash-Player der Version 4 als Grundvoraussetzung ausgehen, um (zur Zeit noch) alle Flash-Filme betrachten zu können. Für diesen Fall ermittelte die genannte Studie im Durchschnitt noch 89% Verfügbarkeit bei den Benutzern. Dies entspricht immer noch 287 Millionen Menschen, beziffert man die Anzahl der Online-User mit 323 Millionen [IDC2000].

Ein Vorteil des Flash-Plugins, der sicher zu seiner Verbreitung beiträgt, ist seine momentane Größe von 220 Kilobytes (Version 5 für Windows). Eine Datei, die mit einem Standardmodem (33,6 KBit/s) innerhalb einer Minute auf den eigenen Rechner kopiert werden kann. Doch ist daraus abzuleiten, dass das Flash-Plugin 287 Millionen mal aus dem Internet heruntergeladen wurde? Sicher nicht, da dies - wie bereits in der Einleitung erwähnt - ein Vorgang ist, der vom Endbenutzer gern vermieden wird, auch wenn das Plugin selbst natürlich kostenlos zu beziehen ist.

Die Verbreitung des Shockwave Flash-Plugins basiert vielmehr auf einer geschickten Strategie von Macromedia: Durch Kooperation mit Firmen wie Microsoft, Netscape und Apple wird es inzwischen mit jedem neuen Windows- und Macintosh-Betriebssystem ebenso wie mit jedem neuen Webbrowser für diese Systeme (Internet Explorer, Netscape, AOL) standardmäßig mitgeliefert. Seit 1999 existieren außerdem Versionen für Linux, Solaris und IRIX. Das Flash-Plugin steht damit heute auf neuen Rechnern automatisch zur Verfügung und selbst Unix-Systeme werden unterstützt.

Eine plattformunabhängige Java-Version des Plugins existiert zusätzlich für Syste-

me die über einen Java-fähigen Webbrowser verfügen, sie wurde jedoch seit Flash 3 nicht weiterentwickelt.

Um positiv auf die Verbreitung und die Akzeptanz des Flash-Dateiformates einzuwirken und sich damit gegen ehemalige Konkurrenten wie Adobes PGML zu schützen, veröffentlichte Macromedia bereits im Juni 1998 im Rahmen des „Flash Open File Format“-Programms das Dateiformat der Flash-Filme. Macromedia verkündete offiziell, Flash damit als offenen Internet-Standard etablieren zu wollen.

Ein erstes Ergebnis der sogenannten „SWF file format specification“ war Oliver Debons Flash-Plugin für Netscape unter Linux [Debo2000], das vor dem offiziellen Plugin von Macromedia zum Download bereit stand.

Die noch unvollständige und teilweise fehlerhafte Dokumentation des Dateiformates löste Macromedia Anfang 2000 durch das SWF Software Development Kit (kurz: SWF-SDK) ab, das kostenlos aus dem Internet zu beziehen ist [Macr2000c]. Darin beschreibt die vollständig überarbeitete Spezifikation sehr ausführlich das Dateiformat und eigene SWF-Dateien lassen sich mit einer Sammlung von C++-Klassen erzeugen. Die Idee von Macromedia ist offensichtlich: Mit Hilfe des SWF-SDK lassen sich Flash-Filme über eine Programmierschnittstelle erzeugen, was Dritthersteller ermutigen soll, Unterstützung für das SWF-Dateiformat in ihre Programme zu integrieren. Adobe hat dies bereits für seine Produkte Illustrator und LiveMotion getan, Corel integrierte eine Exportschnittstelle für SWF-Dateien in CorelDRAW.

Aber nicht nur die Verbreitung des Flash-Dateiformates wird von Macromedia forciert. Zeitgleich mit dem SWF-SDK zur Erzeugung von Flash-Filmen veröffentlichte Macromedia die Quellen des Flash-Players. Softwarehersteller können sich von Macromedia lizensieren lassen und so ihre Produkte mit Hilfe der Flash-Player-Quellen um die Fähigkeit der Wiedergabe von Flash-Filmen erweitern.

Sehr interessant für die vorliegende Diplomarbeit ist das zuerst genannte SWF-SDK, mit dem sich ohne Einsatz des Flash Studios Flash-Filme erzeugen lassen. Mit Hilfe dieses Software Development Kits lässt sich ein Mechanismus entwickeln, der aus Wettervorhersagedaten beziehungsweise beliebigen zeit- und raumbasierten Daten automatisch einen animierten Flash-Film generiert.

Das SWF-SDK stellt eine Programmierschnittstelle zur Verfügung, die den internen Aufbau einer SWF-Datei nahezu versteckt und stattdessen anschauliche Objekte zur Kapselung der Information verwendet. Ein detailliertes Verständnis des Dateiformates bleibt dem Benutzer damit erspart. Der folgende Abschnitt 2.3 gibt trotzdem einen kurzen Überblick über seine Idee und Struktur. Abschnitt 2.4 ab Seite 16 widmet sich anschließend dem SWF-SDK.

2.3 SWF - das Dateiformat

Das Dateiformat der Flash-Filme wurde von Macromedia mit dem Ziel entworfen, Vektorgrafiken und Animationen über das Internet zu übertragen. Deshalb wurde es als effizientes Übertragungsformat und nicht als Austauschformat zwischen verschiedenen Grafikeditoren entwickelt. Macromedia gibt an, dass das SWF-Dateiformat die folgenden Gesichtspunkte erfüllt [Macr2000c]:

- **Bildschirmdarstellung**
  Das Format wurde hauptsächlich zur Darstellung auf einem Bildschirm entworfen und unterstützt deshalb Anti-Aliasing[3], schnelles Rendering[4] in ein Bitmapformat mit beliebiger Farbtiefe[5], Animation und interaktive Schaltknöpfe.

- **Erweiterbarkeit**
  Das Format ist tagbasiert, d.h. die internen Daten werden durch *Tags* (Marken) in Blöcke bestimmter Typen eingeteilt. Indem der Flash-Player unbekannte Tags und damit den entsprechenden Datenblock ignoriert, kann das Format um neue Möglichkeiten erweitert werden, während die Kompatibiliät zu älteren Flash-Playern erhalten bleibt.

- **Netzwerkübertragung**
  Die Dateien können über ein Netzwerk mit geringer Bandbreite übertragen werden, da sie komprimiert und streamfähig sind. SWF ist ein binäres Dateiformat und damit nicht mit einem Texteditor lesbar wie z.B. HTML. Techniken wie Bit-Packing und Strukturen mit optionalen Feldern werden benutzt, um die Dateigröße zu minimieren.

- **Einfachheit**
  Das Format ist einfach gehalten, damit der Flash-Player von geringer Größe und leicht auf andere Systeme zu portieren ist. Der Flash-Player greift außerdem auf nur wenige Betriebssystemfunktionen zu.

[3] Kantenglättung zur Verminderung des Treppeneffekts, der durch die Pixeldarstellung bei schrägen und gekrümmten Linien entsteht. Durch Interpolation, d.h. eine farbliche Angleichung benachbarter Bildpunkte, werden die „Treppenstufen“ ausgeglichen.

[4] Englische Bezeichnung für „Übersetzung“ oder „Übertragung“. Im CAD- und Grafikbereich versteht man unter Rendering die optische Aufwertung eines zwei- oder dreidimensionalen CAD-Modells mittels computerunterstützter Prozesse / Algorithmen. Dazu können beliebige Lichtquellen positioniert sowie Farben bzw. Texturen und andere Effekte zugeordnet werden. Im Zusammenhang mit Flash bezeichnet Rendering die Umsetzung der mathematischen Beschreibung der Vektorgrafik in eine Pixeldarstellung für bestimmte Auflösungen und Farbtiefen.

[5] Informationsmenge, mit der die Farbe eines Bildpunktes beschrieben wird. 24 Bit Farbtiefe verteilen sich auf 8 Bit je Farbkanal (rot, grün, blau = RGB) und entsprechen „TrueColor“ mit 16,7 Millionen möglichen Farbwerten.

- **Datenunabhängigkeit**
  Die Dateien können unabhängig von externen Datenquellen wie z.B. Schriftendefinitionen dargestellt werden.

- **Skalierbarkeit**
  Unterschiedliche Computer besitzen unterschiedliche Bildschirmauflösungen und Farbtiefen. Für die Darstellung der SWF-Dateien reicht eine einfache Hardware-Ausstattung aus, der Vorteil leistungsstärkerer Hardware wird jedoch genutzt, sofern diese vorhanden ist.

- **Geschwindigkeit**
  Das Format unterstützt sehr schnelles Rendering bei hoher Qualität.

Neben den aufgeführten Ideen des Dateiformates ist vor allem die tagbasierte Strukturierung der Daten interessant für die spätere Erzeugung von Flash-Filmen mit dem SWF-SDK. Deshalb soll der interne Dateiaufbau noch genauer betrachtet werden:

Alle SWF-Dateien besitzen einen Dateikopf, der mit der Signatur „SWF“ gefolgt von der Versionsnummer des Dateiformats beginnt. Es folgen die Dateigröße in Bytes und die Breite und Höhe des Flash-Filmes. Weiter erhält man Auskunft darüber, wie viele Animationsbilder (Frames) die Datei enthält und mit welcher Rate sie abgespielt werden sollen. Dem Dateikopf folgen beliebig viele Datenblöcke und ein abschließender Endblock, wie in Abbildung 2.2 skizziert. Die Datenblöcke genügen alle einem einheitlichen Format, so dass ein Programm, das eine SWF-Datei einliest und verarbeitet, Blöcke überspringen kann, die es nicht versteht (→ Erweiterbarkeit).

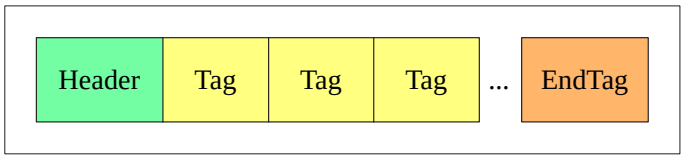


Abbildung 2.2: SWF-Dateistruktur

Die „SWF File Format Documentation“ von Macromedia [Macr2000c] erläutert das Format der unterschiedlichen Datenblöcke. Dort werden für Datenblöcke die Begriffe *Tag* und *Record* verwendet. Tags sind spezielle Records, die neue Datenblöcke einleiten und den Blocktyp festlegen. Records enthalten Attribute und eventuell wieder Records oder Verweise auf andere Tags (Datenblöcke). Ein Beispiel für die Definition eines allgemeinen Grafikobjekts zeigt Quellcode 2.1.

```
DEFINESHAPE tag - leitet Definition des Grafikobjekts ein
tag id - Typ des Datenblocks
character id - Name des Grafikobjekts
RECT record - Größe des Grafikobjekts
    ... - Koordinaten
SHAPEWITHSTYLE record - Defintion des Grafikobjekts
    FILLSTYLE record - Füllstil
        ... - Definition des Füllstils
    LINESTYLE record - Linienstil
        ... - Definition des Linienstils
    STRAIGHTEDGE record - gerade Linie
        ... - Koordinaten von Anfangs- und Endpunkt
    CURVEDEDGE record - gekrümmte Line
        ... - Koordinaten und Bezierpunkte
    ... - weitere Linien
    ENDSHAPE record - Endpunkt
```

Quellcode 2.1: Definition eines SWF-Grafikobjekts

Es existieren zwei Kategorien von SWF-Tags beziehungsweise Typen von Datenblöcken: Definitionsblöcke *(definition tags)* und Kontrollblöcke *(control tags*). Definitionsblöcke beschreiben den Inhalt des Flash-Filmes. Sie definieren Grafikobjekte (Rechtecke, Kreise, Texte, ...), eingebettete Pixelgrafiken und Sound. Während ihrer Definition bekommen diese Elemente eine eindeutige Kennung zugewiesen, die in einer Referenzliste gespeichert wird. Kontrollblöcke werden dazu verwendet, den Film zu steuern und die Elemente des Films zu verändern. So kann z.B. ein Grafikobjekt platziert, Sound gestartet oder zu einem anderen Animationsbild gesprungen werden. Die Elemente des Films werden dabei über die Referenzliste angesprochen, können also auch mehrfach im Film verwendet werden.

Bis zur Darstellung eines Grafikobjekts auf dem Bildschirm laufen die folgenden Schritte ab:

1. ein Definitionsblock definiert das Grafikobjekt
2. das Grafikobjekt bekommt eine eindeutige Kennung zugewiesen
3. die Kennung wird in der Referenzliste gespeichert
4. ein Kontrollblock referenziert das Grafikobjekt über seine Kennung und stellt es auf dem Bildschirm dar

Um die Streamfähigkeit von Flash-Filmen zu garantieren, darf natürlich ein Kontrollblock stets nur auf Definitionsblöcke verweisen, die vor ihm in der Datei stehen.

Mit der Abbildung der Tags und Records in die C++-Klassen des SWF-SDK und dessen Anwendung beschäftigt sich der folgende Abschnitt.

2.4 SWF-SDK

Das „Macromedia Flash File Format (SWF) Software Development Kit (SDK)“ - so der volle Titel - wurde bereits unter dem Aspekt „Internet-Standard Flash“ kurz erwähnt. Es steht kostenlos zum Download auf Macromedias Website bereit und enthält neben der Dokumentation des SWF-Dateiformats in C++ geschriebenen Quellcode, der das Erzeugen von Flash-Dateien über eine Programmierschnittstelle ermöglicht.

Die mitgelieferten Klassen sind mit der Einschränkung frei zu verwenden, dass sie lediglich zur Entwicklung von SWF-exportierenden Programmen genutzt werden [Macr2000c]. Macromedia verlangt, dass die unter Verwendung des SDK entwickelten Produkte Flash-Filme im korrekten Format speichern. Erzeugte SWF-Dateien müssen erstens fehlerfrei mit einem aktuellen Flash-Player abgespielt werden können und zweitens fehlerfrei in die grafische Entwicklungsumgebung Flash importiert werden können. Macromedia übernimmt keine Garantie für die Korrektheit der bereitgestellten Software und behält sich vor, das SDK nach Bedarf zu aktualisieren.

Das zur Zeit verfügbare SWF-SDK stellt über eine Bibliothek von C++-Klassen die komplette Funktionalität bereit, die benötigt wird, um Flash 3 SWF-Dateien zu erzeugen. Die Klassen der Bibliothek können in zwei Bereiche unterteilt werden: den *Low Level Manager* und den *High Level Manager*.

Der Low Level Manager hat das Ziel, die Struktur einer SWF-Datei so genau wie möglich abzubilden. Für jedes Defintions- und Kontroll-Tag existiert eine Klasse, die von der Basisklasse `FObj` (für Flash Object) abstammt. Alle übrigen Records sind ebenfalls als Klassen realisiert. Den eigentlichen Flash-Film erstellt die Klasse `FObjCollecton`, die die `FObj`s sammelt und schließlich in eine Datei schreibt. Tabelle 2.2 zeigt die Gruppen, in die sich die Klassen des Low Level Managers einteilen lassen.

`FObjCollection`	- sammelt Tags (`FObj`s) und schreibt die Datei
`FObj`	- Basisklasse aller Tagklassen
`Control Tag Classes`	- Kontrolltags (beginnen mit `FCT`)
`Definition Tag Classes`	- Definitonstags (beginnen mit `FDT`)
`Primitive Data Classes`	- weitere Records wie Füllstil, Farbe, Linie, ...
`Helper Classes`	- Hilfsklassen für Sound und Pixelgrafiken

Tabelle 2.2: Klassen des Low Level Managers

Der High Level Manager abstrahiert die zugrundeliegende Dateistruktur aus Tags und Records und bildet eine benutzerfreundliche Schnittstelle zum Erzeugen von

Flash-Filmen. High Level Klassen basieren auf den Klassen des Low Level Managers und beginnen mit dem Präfix `HF` (für High Level Flash). Es existieren Klassen zum Erzeugen von Filmen (`HFMovie`), Schlüsselbildern (`HFFrame`), Grafikobjekten (`HFRectangle`, `HFButton`, ...), Text (`HFText`), Bitmaps (`HFBitmap`) und Sound (`HFSound`). So lässt sich zum Beispiel ein Kreis mit Hilfe der Klasse `HFCircle` aus Mittelpunkt und Radius erstellen und muss nicht aus einzelnen gekrümmten Linien (Low Level Klasse `FShapeRecEdgeCurved`) zusammengesetzt werden. Im Gegensatz dazu sind jedoch einige Methoden, insbesondere die des ActionsScripts, nicht vom High Level Manager implementiert. Die tatsächlich vorhandenen HF-Klassen zeigt Tabelle 2.3.

`HFMovie`	- Flash-Film
`HFFrame`	- Schlüsselbild
`HFObject`	- Basisklasse aller Film-Elemente
`HFButton`	- Schaltknopf
`HFFont`	- Schriftart
`HFShape`	- Basisklasse aller Grafikobjekte
`HFEditText`	- Editierbares Textfeld
`HFOval`	- Ellipse
`HFCircle`	- Kreis
`HFPolygon`	- Polygon
`HFRectangle`	- Rechteck
`HFBitmap`	- Rechteck mit Pixelgrafik gefüllt
`HFText`	- Text
`HFSound`	- Sound
`HFAction`	- ActionScript-Anweisung

Tabelle 2.3: Klassen des High Level Managers

Die Verwendung der Klassen des High Level Managers soll an einem Beispiel (siehe Quellcode 2.2 auf Seite 18) verdeutlicht werden. Es werden drei gleich große Quadrate (`HFRectangle`) mit unterschiedlichem Füllstil erzeugt, die sich teilweise überlappen. Das dritte Quadrat dreht sich in 12 Schritten um die eigene Achse, es reicht aber aus, eine Viertel Drehung zu imlementieren, da das Quadrat bereits dann wieder seine ursprüngliche Form annimmt.

Die allererste Zeile, die Include-Anweisung, bindet die Headerdatei `HF3SDK.h` des SWF-SDK ein, die benötigt wird, um die Klassen des High Level Managers verwenden zu können. Sie weist außerdem auf die Version hin, in der das SWF-SDK momentan vorliegt.[6]

[6] Dennoch gibt Macromedia auf seinen Webseiten an, dass sich mit dem SWF-SDK Flash-Filme der Version 4 generieren lassen. Diese Aussage ist falsch, weil die in Flash 4 eingeführten ActionScript-Anweisungen noch nicht unterstützt werden.

```
#include <HF3SDK.h>

int main() {

    // erzeuge einen neuen Flash-Film (300*300 Pixel)
    HFMovie movie = HFMovie();
    movie.SetSize( 6000, 6000 );
    movie.SetFrameRate( 3 );

    // drei Rechtecke (FPIXEL = 20)
    HFRectangle* rect1 =
        new HFRectangle( 0*FPIXEL,0*FPIXEL, 100*FPIXEL,100*FPIXEL );
    HFRectangle* rect2 =
        new HFRectangle( 50*FPIXEL,50*FPIXEL, 150*FPIXEL,150*FPIXEL );
    HFRectangle* rect3 =
        new HFRectangle( 100*FPIXEL,100*FPIXEL, 200*FPIXEL,200*FPIXEL );

    // setze die Füllfarben
    rect1->SetSolidFill( Red_RGBA );
    rect2->SetLinearFill( Blue_RGBA, Black_RGBA );
    rect3->SetRadialFill( Yellow_RGBA, Violet_RGBA );

    // platziere Rechtecke übereinander
    rect1->SetDepth( 1 ); rect2->SetDepth( 2 ); rect3->SetDepth( 3 );

    // füge Rechtecke zu erstem Schlüsselbild hinzu
    movie.Frame( 0 )->AddObject( rect1 );
    movie.Frame( 0 )->AddObject( rect2 );
    movie.Frame( 0 )->AddObject( rect3 );

    // rect3 dreht sich um den eigenen Mittelpunkt
    movie.Frame( 1 )->RemoveObject( rect3 );
    rect3->Rotate( FloatToFixed( 30.0 ) );
    movie.Frame( 1 )->AddObject( rect3 );

    movie.Frame( 2 )->RemoveObject( rect3 );
    rect3->Rotate( FloatToFixed( 60.0 ) );
    movie.Frame( 2 )->AddObject( rect3 );

    // Animation beginnt von vorn

    // schreibe die SWF-Datei und beende das Programm
    movie.WriteMovie("HFRectangleExample.swf");
    delete rect1; delete rect2; delete rect3;
    return( 0 );
}
```

Quellcode 2.2: High Level Manager Beispiel

Größenangaben werden in Flash stets in Twips abgespeichert, eine künstliche Einheit, die 1/20 Pixel entspricht. Das heisst, dass ein 1000*1000 Twips großes Quadrat in seiner Ursprungsgröße 50*50 Pixel auf dem Bildschirm einnimmt. (Natürlich kann es auf jede beliebige Größe skaliert werden, schließlich ist es ein Vektorgrafikobjekt.) Die im SDK definierte Konstante FPIXEL spiegelt den Faktor 20 wider. Für die Drehung und das Skalieren von Objekten werden ebenfalls flash-interne Größen verlangt. Der Zoomfaktor oder Rotationsgrad kann aber mit dem Makro `FloatToFixed` in den Argumenttyp umwandelt werden.

Die Elemente des Flash-Films können über den Aufruf der Methode `SetDepth(int)` in verschiedene Ebenen positioniert werden. Höhere Ebenen liegen über niedrigeren und verdecken die darunterliegenden beziehungsweise lassen diese je nach Transparenz ihrer Füll- und Linienfarbe durchscheinen. Die Ebene, in der ein Grafikobjekt platziert ist, wird benötigt, um dieses Grafikobjekt während der Animation wieder vom Bildschirm verschwinden zu lassen. Flash referenziert die Elemente des Films über ihre „character id“ (siehe Seite 15) und eine Ebenenangabe. Mehrere Instanzen eines Grafikobjekts dürfen deshalb nicht auf derselben Ebene existieren. Der High Level Manager kümmert sich um die Einhaltung dieser Regel [7].

Kompiliert und startet man den Quellcode des Beispiels, wird der beschriebene Flash-Film erzeugt. Seine Dateigröße beträgt lediglich 266 Bytes. Abbildung 2.3 zeigt eine Momentaufnahme des zweiten Schlüsselbildes, das dritte Quadrat ist bereits um 30 Grad rotiert.

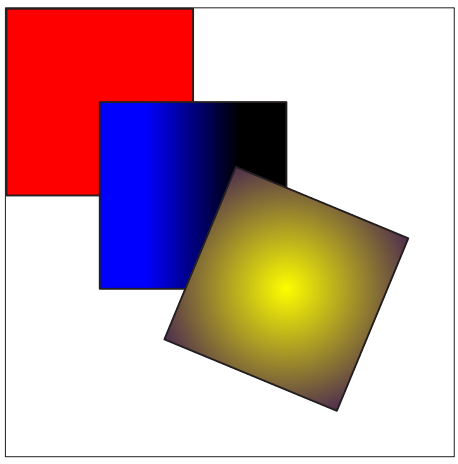


Abbildung 2.3: Screenshot des Beispiel-Films

[7] Ein Fehler im SWF-SDK: Mehrere Instanzen eines Objekts werden vom High Level Manager auf unterschiedlichen Ebenen platziert, löschen lässt sich jedoch nur die zuletzt hinzugefügte Instanz. Eine eigenständige Korrektur der Klasse `HFObject` war nötig, um den Fehler zu beheben. Macromedia reagierte auf eine Mitteilung dieses Fehlers nur mit einem Verweis auf die OpenSWF-Newsgroup von www.flashkit.com.

2.5 Ausblick

Zusammenfassend lässt sich feststellen, dass SWF-Dateien sowohl mit der grafischen Entwicklungsumgebung Macromedia Flash als auch mit dem Flash File Format SDK erzeugt werden können. Die grafische Entwicklungsumgebung kostet mehrere hundert Mark, ist aber wesentlich einfacher zu bedienen als das kostenlose SWF-SDK und setzt keinerlei Programmierkenntnisse voraus - der Benutzer „malt einfach drauf los“, kann seine Ergebnisse mitverfolgen und sofort testen. Das Software Development Kit ermöglicht das Erzeugen von Flash-Filmen über eine Programmierschnittstelle, die abgesehen von Programmierfehlern auch Fehler im Layout nicht sofort kenntlich macht. Das Testen eines Filmes erfordert stets das Kompilieren der Quelle und das Öffnen des erzeugten Films mit einem Flash-Player. Der Entwicklungszyklus verlängert sich immens.

Dennoch eröffnet das SWF-SDK neue interessante Möglichkeiten: Die automatische Generierung von Flash-Filmen mit ähnlichem Inhalt, die z.B. Daten desselben Formats visualisieren. Genau hier setzt die vorliegende Diplomarbeit an. Wettervorhersagedaten und Landkarten sollen jeden Tag nach bestimmten Vorgaben in eine Animation umgesetzt werden. Die Aufgabe ist es also, einen Mechanismus zu entwickeln, der diesen Visualisierungsvorgang durchführt. Einen ersten Entwurf eines solchen Generators[8] skizziert Abbildung 2.4.

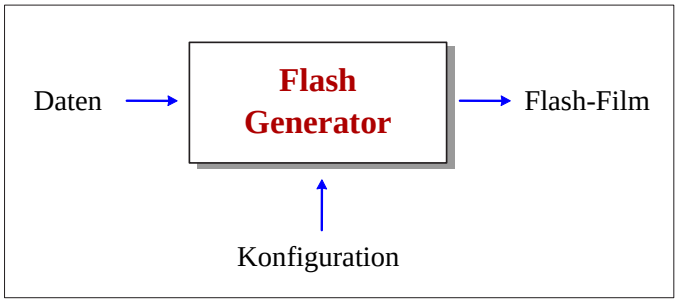


Abbildung 2.4: Erster Entwurf eines Flash-Generators

Ein Nachteil bleibt weiterhin, dass für jede minimale Layoutänderung des erzeugten Flash-Films die Programmquellen des Generators angepasst und neu übersetzt werden müssen. Ein zusätzliches Konfigurationsmodul wäre wünschenswert, wie schon in Abbildung 2.4 angedeutet. Einstellungen an z.B. Farben, Füll- und Linienstilen lassen sich so zur Laufzeit vornehmen. Eine Auswahl erfolgt am einfachsten über Auswahlleisten und Schaltknöpfe einer grafischen Oberfläche. Da das SDK in C++ geschrieben ist, folgt daraus die Programmierung einer Konfigurationsoberfläche in C++?

[8] hiermit ist nicht das Softwareprodukt Macromedia Generator gemeint.

Exkurs: Sprachkonflikt - C++ versus Java

Das SWF-SDK besteht aus Klassen, die ohne jegliche Bildschirmeingabe und -ausgabe arbeiten. Es kann auf jeder Plattform mit dem Gnu C++-Compiler übersetzt werden. Auch der Flash-Player ist inzwischen für verschiedene Plattfomen implementiert, Flash-Filme können plattformübergreifend betrachtet werden (siehe Abschnitt 2.2). Die angestrebte Konfigurationsoberfläche des Flash-Generators lässt sich jedoch nur für mehrere Systeme gleichzeitig entwickeln, wenn Java als Programmiersprache verwendet wird. Java bietet die Möglichkeit, plattformunabhängige Applikationen mit grafischer Oberfläche zu entwickeln.

Die Idee, die C++-Klassen des SWF-SDK mit eigenen Java-Klassen zu kombinieren, lässt sich über eine Schnittstelle namens *Java Native Interface* (JNI) realisieren. Mit der JNI-Technik wird die Schnittstelle zwischen C++-Bibliothek und Java einmalig über die Methodenköpfe definiert und im Folgenden lassen sich C++-Methoden aus Java heraus aufrufen.

Hier kommt es also zum Entscheidungskonflikt zwischen C++ und Java als zu verwendende Programmiersprache für die Implementation eigener Klassen, da beide oben genannten Varianten prinzipiell durchführbar sind. Für C++ spricht die bessere Performance und dass das SWF-SDK in C++ geschrieben ist. Bei einer einheitlichen C++-Lösung kann das SWF-SDK direkt verwendet werden, die Implementation ist jedoch plattformabhängig. Java bedeutet im Gegensatz dazu einen erheblichen Mehraufwand an Programmierarbeit, da die Schnittstelle zum SWF-SDK mittels JNI erst geschaffen werden muss.

Das Argument der Plattformunabhängigkeit überwiegt jedoch und lässt die Entscheidung zugunsten von Java ausfallen, denn das folgende Kapitel über das Datenformat XML wird zeigen, dass XML ebenso wie Java unabhängig von der verwendeten Plattform einsetzbar ist. Java und XML sind wie füreinander geschaffen. Die Wahl von Java erfordert den erwähnten zusätzlichen Programmieraufwand, ermöglicht aber eine elegantere Lösung. Z.B. entfallen auf diese Weise sämtliche Zeichenketten-Konvertierungen vom erweiterten Unicode-Zeichensatz, der in C++ für XML-Daten verwendet wird, in den ASCII-Zeichensatz des C++-Programmcodes und umgekehrt.

3 eXtensible Markup Language

Ein Teil der Aufgabenstellung der vorliegenden Diplomarbeit ist die „Visualisierung von raum- und zeitbezogenen Daten mit Macromedia Flash“. Damit ist Flash als das Ausgabeformat des Visualisierungsprozesses vorgegeben, nicht jedoch das oder die Formate der Eingabedaten.

Eine breite Unterstützung möglicher Eingabeformate kann im Rahmen dieser Arbeit nicht realisiert werden. Ebensowenig darf eine Beschränkung auf das Datenformat der beispielhaft verwendeten Wettervorhersagedaten erfolgen. Die Idee ist vielmehr, eine Schnittstelle zwischen Eingabedaten und Visualisierungsmodul zu definieren, dessen Format sehr genau die Fähigkeiten von Macromedia Flash widerspiegelt. Im Gegensatz zum bitcodierten SWF-Dateiformat soll dieses Zwischenformat jedoch leicht zu lesen und zu bearbeiten sein, damit beliebige Eingabedaten relativ einfach dorthin konvertiert werden können. Das angestrebte Schnittstellenformat muss die Flash-Dateistruktur bestehend aus Tags und Records abbilden und ebenfalls erweiterbar sein. Ein textbasiertes Format bietet sich an, das einen Wortschatz definiert, mit dessen Hilfe die Struktur eines Flash-Filmes beschrieben werden kann. Der Flash-Generator aus Abbildung 2.4 liest eine solche Textdatei, die notfalls mit einem Texteditor erstellt werden kann, und erzeugt ohne weitere Programmierung aus dem Inhalt den beschriebenen Flash-Film. Da der Flash-Generator auf dem SWF Software Development Kit basiert (siehe Abschnitt 2.4), erweist sich eine direkte Abbildung der SDK-Klassen durch das Schnittstellenformat als noch eleganter: Die vom Flash-Generator eingelesene Information kann direkt in Objekte des SWF-SDK umgesetzt werden.

Abbildung 3.1 zeigt einen erweiterten Entwurf des Visualisierungsprozesses.

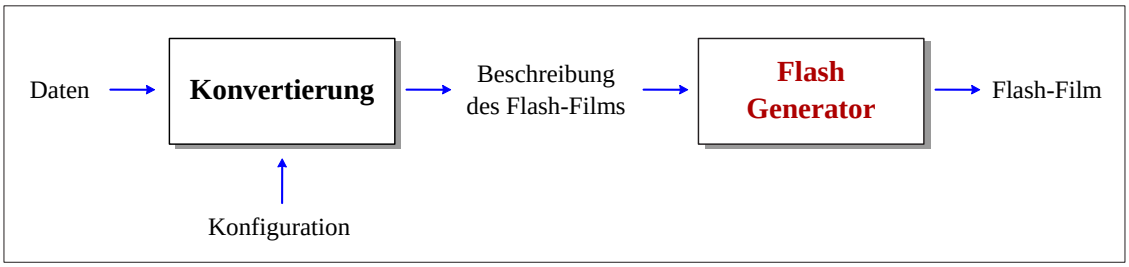


Abbildung 3.1: Erweiterter Entwurf des Visualisierungsprozesses

Ein erster Ansatz, ein Zwischenformat nach den oben genannten Anforderungen zu entwickeln, ließ ein Schnittstellenformat mit eigener Syntax entstehen. Lese- und Schreibmodule für dieses Format hätten ebenso wie Klassen zur Manipulation der Daten implementiert werden müssen. Ein Standard, der gewisse Grundregeln

vorschreibt, wie strukturierte Daten in einer Textdatei abzulegen sind, ist XML. Was XML bedeutet, welche Ideen mit diesem Standard verfolgt werden und warum die Implementation eigener Schreib- und Lesemodule nicht mehr erforderlich ist, beschreiben die folgenden Abschnitte.

3.1 Was ist XML?

XML ist eine Abkürzung und steht für *eXtensible Markup Language*. Eine Kurzdefinition lautet: „XML ist eine textbasierte Meta-Auszeichnungssprache, die die Beschreibung, den Austausch, die Darstellung und die Manipulation von strukturierten Daten erlaubt, so dass diese - vor allem über das Internet - von einer Vielzahl von Anwendungen genutzt werden können.“ [ZAIT2000]

Eine Auszeichnungssprache formatiert Texte mit Hilfe von Auszeichnungen (oder Markierungen, engl. Markup), die den Inhalt des Dokuments in einzelne Elemente strukturieren. Diese Auszeichnungen sind selbst im Textformat gehalten und heißen im allgemeinen Sprachgebrauch *Tags*. Sie bestehen aus einer Start- und einer Endmarke und bilden zusammen mit dem jeweils markierten Inhalt die Elemente des Dokuments. Die inhaltliche Bedeutung oder visuelle Formatierung eines Elements bestimmt der Name des Tags. Komplexere Dokumentstrukturen entstehen durch die Schachtelung von Elementen.

„Extensible“ heißt XML, da es sich im Gegensatz zu HTML nicht um ein festes Format im Sinne einer bestimmten Auszeichnungssprache, sondern um eine Metasprache handelt. Eine Metasprache ist eine Sprache zur Beschreibung von Sprachen, d.h. sie stellt Vorschriften bereit, um eine beliebige Anzahl konkreter Auszeichnungssprachen für die verschiedensten Arten von Dokumenten zu definieren. XML liefert ein Konzept, wie Daten zu strukturieren sind, ohne dass vorher festgelegt wird, welche Elemente zur Verfügung stehen. Eine mittels XML definierte Sprache heißt auch *XML-Anwendung* oder *XML-Ausprägung*.

Mit XML können Struktur und Inhalt von Dokumenten so präzise beschrieben werden, dass es letztlich nicht mehr notwendig ist, die zum Verständnis und der Weiterverarbeitung von Daten notwendigen Informationen fest in die Anwendungen zu integrieren.

3.2 XML - die Entwicklung

Das HTML-Dilemma

Mit der Hypertext Markup Language (HTML) existiert eine Auszeichnungssprache, deren Einfachheit den Siegeszug des WorldWideWeb zu Beginn und Mitte der Neunziger Jahre ermöglichte. 1989 entwickelte Tim Berners-Lee am Forschungszentrum der Europäischen Organisation für Kernforschung, CERN, diese Hypertextauszeichnungssprache mit dem primären Ziel, den Austausch wissenschaftlicher Dokumente mit der Möglichkeit des Verweisens auf entfernte Quellen zu unterstützen. HTML ist damit vornehmlich eine Präsentationsbeschreibungssprache.

Der Vorteil von HTML liegt in seiner Einfachheit: Die Sprache besteht in der Version 3.2 aus ca. 70 Tags und über 50 Attributen [Mach1997]. Doch diese Einfachheit hat auch ihren Preis. Durch den kommerziellen Erfolg des WorldWideWeb und wachsende Möglichkeiten und Anforderungen an den Datenaustausch über das Internet wurde deutlich, dass sich HTML für anspruchsvollere Anwendungen disqualifiziert. Die ursprüngliche Aufgabe der strukturierten Darstellung von Daten trat immer mehr in den Hintergrund. HTML wurde zum Medium für virtuelle Kaufhäuser, Banken und andere graphisch aufwendige Webseiten. So nutzt inzwischen eine Vielzahl von Anwendungen wie etwa Datenbanken mit HTML-Frontend das WorldWideWeb als Oberfläche. Allerdings geht durch die Verwendung von HTML die reiche, innere Struktur der Daten verloren. Komplexere Strukturen wie das Relationenschema einer Datenbank oder Objekthierarchien sind nicht abbildbar. Weiterhin ist es mit HTML nicht möglich, Daten semantisch auszuzeichnen, d.h. den Dokumenten Informationen über ihren Inhalt, jenseits der reinen Darstellung, mitzugeben. „Bei der Umsetzung in Web-Dokumente findet immer ein Informationsverlust statt. Bei einer über das Web abfragbaren relationalen Datenbank etwa verschwindet die auf dem Server vorhandene Strukturierung der Nutzdaten in Felder auf der Client-Seite in einem Meer von Tags. Eine Nutzung der Daten beim Betrachter, die über das Ausschneiden von Text mittels Cut & Paste hinausgeht, ist nicht mehr möglich.“ [Mach1997]

Die häufig als „HTML-Dilemma“ bezeichneten Defizite von HTML lassen sich wie folgt zusammenfassen [Bosa1997]:

1. **Erweiterbarkeit**
   HTML erlaubt weder das Setzen eigener Tags noch das Spezifizieren individueller Attribute zur semantischen Auszeichnung von Daten. HTML ist damit ein reines Präsentationsformat. Ein in HTML codiertes Dokument enthält nur

Informationen, wie Inhalte darzustellen sind; weitergehende Informationen, d.h. Metainformationen, über die Semantik des Inhalts sind nicht abbildbar.

2. **Struktur**
   In HTML können keine Datenstrukturen jenseits von Formatinformationen beschrieben werden. Der Zusammenhang der Daten untereinander ist nicht darstellbar.
3. **Validierung**
   HTML fehlen Sprachspezifikationen, die Anwendungen, die HTML-codierte Daten verarbeiten, eine Überprüfung der strukturellen Validität der Daten erlauben, also der Korrektheit der Struktur des Markup in XML-Dokumenten.

SGML

Mit SGML (Standard Generalized Markup Language) existiert seit über zehn Jahren ein internationaler Standard (ISO 8879) für die Definition, Identifikation und Benutzung der Struktur und des Inhalts von Dokumenten. Als Metasprache stellt SGML Vorschriften bereit, um Auszeichnungssprachen formal zu definieren. SGML dient als Basis für die verschiedensten Auszeichnungssprachen auf diversen Medien. Es enthält dazu Sprachmittel für unterschiedlichste Zwecke und ist damit eine äußerst flexible Architektur, mit der Dokumente für beliebige Medien aufbereitet werden können, ohne die Struktur der Daten zu verlieren. So ist HTML eine Anwendung (genauer: ein *Dokumenttyp*) von SGML. Problematisch ist jedoch die Komplexität von SGML, die die Entwicklung von SGML-Anwendungen teuer und kompliziert machte und bisher einer weiten Verbreitung von SGML entgegenstand [Bosa1997]. Während HTML also aufgrund der fehlenden Erweiterbarkeit für komplexere Anwendungen ungeeignet ist, erweist sich SGML wegen seiner hohen Komplexität als im Internet nur begrenzt einsetzbar.

Hier setzt die eXtensible Markup Language (XML) an. Mit dem im Februar 1998 vom WorldWideWeb Consortium (W3C[9]) als *Recommendation* (Empfehlung) verabschiedeten Sprachkonzept wird versucht, eine für das Internet geeignete Meta-

[9] Das W3C ist ein 1994 gegründetes internationales Industriekonsortium, das sich die Entwicklung der Standards für das Web zur Aufgabe gemacht hat. Es steht unter gemeinsamer Leitung des Massachusetts Institute of Technology Laboratory for Computer Science (MIT/LCS) in den USA, des europäischen Institut National de Recherche en Informatique et en Automatique (INRIA) und der Keio University in Japan. Die Finanzierung des W3C erfolgt durch die Mitgliedsorganisationen (z.B. Apple Computer, Inc., AT&T, British Telecommunications Laboratories, IBM Corporation, Intel Corporation, Microsoft Corporation, Netscape Communications, Object Management Group, SAP AG, Sun Microsystems, u.v.m.). Das W3C ist nach eigenen Angaben „...funded by member organizations, and is vendor neutral, working with the global community to produce specifications and reference software that is made freely available throughout the world.“ [W3C2000a]

sprache zu etablieren, die die Funktionalität von SGML für das WorldWideWeb bietet, allerdings ohne dessen hinderliche Komplexität [W3C2000b].

XML ist eine Teilmenge von SGML und wie bei SGML handelt es sich um eine Metasprache. Um die Komplexität zu reduzieren, wurden alle für das Internet als überflüssig angesehenen SGML-Eigenschaften sowie eine Vielzahl als zu kompliziert erachteter und zu selten genutzter Features nicht in XML übernommen. Trotz der Reduzierungen ist XML aufwärtskompatibel zu SGML und wird daher gelegentlich auch „SGML lite“ genannt [Mach1997]. Die seit vielen Jahren bewährten grundsätzlichen Ideen von SGML blieben jedoch erhalten, so dass viele Entwickler auf XML trotz seiner erst kurzen Geschichte vertrauen.

3.3 XML - die Sprache

XML basiert genau wie SGML auf der Idee des strukturierten Auszeichnens von Daten. Da HTML eine SGML-Anwendung ist, unterscheiden sich XML-Dokumente auf den ersten Blick nicht wesentlich von HTML-Dokumenten. Auch XML-Dokumente bestehen aus durch Tags ausgezeichneten Inhalten. Während die Anzahl und Benennung der Tags für HTML aber vorgegeben ist, können für XML-Dokumente beliebig viele und frei („semantisch“) benannte Tags verwendet werden. Somit besteht der wesentliche Unterschied darin, dass bei XML die Auszeichner Informationen über den Inhalt enthalten können und die Verschachtelung der Tags ineinander die Struktur der Daten abbilden kann.

XML unterscheidet sich in den folgenden drei Punkten grundsätzlich von HTML [Bosa1997]:

- Tags und Attribute können individuellen Anforderungen entsprechend definiert und benannt werden.
- Dokumentenstrukturen können in beliebiger Komplexität abgebildet werden.
- XML-Dokumente können - müssen aber nicht - eine formale Beschreibung ihrer Grammatik enthalten.

Quellcode 3.1 enthält beispielhaft ein vollständiges XML-Dokument, das einen Artikel mit Titel, Autor und Änderungsdatum repräsentiert. Schon auf den ersten Blick sind einige wesentliche Merkmale von XML erkennbar: Die Datei liegt im Klartext vor und besitzt eine Struktur. Eine Tatsache, die nicht sofort auffällt ist, dass XML zwischen Groß- und Kleinschreibung unterscheidet.

```
<?xml version="1.0"?>
<!DOCTYPE article SYSTEM "article.dtd">
<article changed="1997/03/10">
    <title>XML, Java and the future of the Web</title>
    <author>Jon Bosak</author>
    <chapter number="1">
        <title>Introduction</title>
        <paragraph>
            The extraordinary growth of ...
        </paragraph>
        <!-- ... -->
    </chapter>
    <newpage/>
    <!-- ... -->
</article>
```

Quellcode 3.1: Ein erstes XML-Beispiel

Der Prolog

XML-Dokumente beginnen mit einem Prolog, der allerdings auch leer sein kann, da alle seine Elemente optional sind. Normalerweise befindet sich dort wenigstens die XML-Deklaration mit der Version des XML-Standards, nach der das Dokument erstellt wurde. Allgemein werden *processing instructions* wie diese, also Anweisungen für Programme, die XML-Dokumente verarbeiten, zwischen `<?` und `?>` gestellt:

```
<?xml version="1.0"?>
```

Außerdem kann im Prolog eine Dokumenttyp-Definition referenziert werden:

```
<!DOCTYPE article SYSTEM "article.dtd">
```

XML-Dokumente dürfen an fast allen Stellen Kommentare enthalten. Nur innerhalb von Deklarationen, Tags und anderen Kommentaren sind sie nicht erlaubt, sie können also nicht geschachtelt werden. Ein Kommentar beginnt mit der Zeichenfolge `<!--` und endelt mit `-->`.

Elemente und Attribute

Elemente gliedern ein XML-Dokument logisch bzw. geben ihm seine Struktur. Sie werden von den Auszeichnungen (Tags) des XML-Dokuments gebildet. Jedes Element beginnt mit einem Start-Tag und endet mit dem gleichnamigen Ende-Tag. Der eingeschlossene Inhalt ist der Inhalt des Elements, der Zeichendaten, wiederum Elemente oder eine Mischung von Zeichendaten und Elementen (*mixed content*) enthalten kann. Durch die Schachtelung von Elementen entsteht die Struktur des XML-Dokuments. Daneben kann es auch leere Elemente geben, die durch

ein einzelnes „empty element tag“ abgekürzt werden können. Beispielsweise ist `<emptyTag></emptyTag>` äquivalent zu `<emptyTag/>`.

Neben dem eigentlichen Inhalt können Elemente auch Attribute enthalten. Die Attribute werden im Start-Tag mit angegeben und in der Form `name="wert"` angegeben. Im Beispiel besitzt das Element `article` das Attribut `changed` und spezifiziert das letzte Änderungsdatum des Artikels.

Attribute können nicht wie Elemente geschachtelt werden, sie enthalten immer nur „flachen“ Text. Diesen flachen Text könnte man natürlich auch als weiteres Sub-Element angeben. In der Tat ist es nicht immer eindeutig zu entscheiden, ob gewisse Werte als Attribute oder besser als eigenständige Elemente in XML abgebildet werden.

Mit den bisher vorgestellten Möglichkeiten ist XML nicht viel mehr als „strukturiertes ASCII“. Angesichts des Ziels, jede nur mögliche Art von Dokument repräsentieren zu können, ist das nur natürlich.

Trennung von Inhalt, Struktur und Layout

Das Grundkonzept von XML ist die konsequente Trennung von Inhalt, Struktur und Layout von Dokumenten (siehe Abbildung 3.2). Bisher wurde nur über das eigentliche XML-Dokument (den Inhalt) gesprochen. Die genaue Struktur der Daten legt eine *Dokumenttyp-Definition* fest, die Präsentation des Inhalts erfolgt mit Hilfe eines *Stylesheets*. Die folgenden Abschnitte beschreiben dieses Konzept, das XML seine Mächtigkeit verleiht.

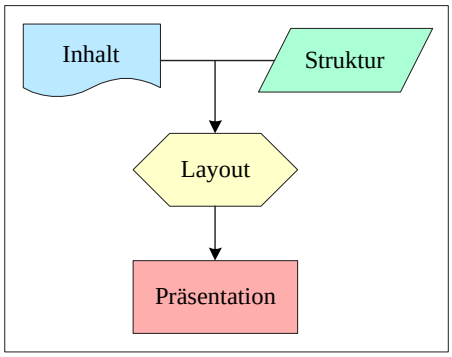


Abbildung 3.2: Das Sprachkonzept von XML

Document Type Definition (DTD)

Dokumenttyp-Definitionen beinhalten die Grammatik, nach der ein XML-Dokument aufgebaut werden kann. D.h. sie geben Auskunft darüber, welche logischen

Elemente (bzw. Tags) ein bestimmter Dokumenttyp enthalten darf, welche vorhanden sein müssen und wie sie kombiniert werden dürfen.

Ein Programm (meistens ein *XML-Parser*) erhält durch die Existenz einer Dokumenttyp-Definition die Möglichkeit, die eingelesenen Dokumente auf die Einhaltung ihrer DTD zu überprüfen. Dokumente, die ihre DTD einhalten, werden als „gültig“ (*valid*) andernfalls als „ungültig“ bezeichnet. Gibt ein Dokument keine DTD an, so kann es weder gültig noch ungültig sein. Der Parser kann es dann nur nach den Grundregeln von XML prüfen und man spricht von einem „wohlgeformten“ (*well formed*) Dokument, sofern es diese Regeln erfüllt.

Die grundlegenden Sprachspezifikationen, die für alle XML-Dokumente gelten müssen, sind im wesentlichen die folgenden Punkte:

- Jedes geöffnete Tag muss explizit geschlossen werden.
- Tags ohne Inhalt (wie `<IMG>` in HTML) müssen in XML entweder explizit geschlossen werden oder mit `/>` enden. Beispielsweise wird aus `<BR>` entweder `<BR/>` oder `<BR></BR>`.
- Die Schachtelung der Elemente ist korrekt und es existiert genau ein Wurzelelement, das alle anderen Elemente umschließt.
- Attributwerte müssen in Anführungszeichen gesetzt werden.
- Das Markup muss wie bei SGML streng hierarchisch gegliedert sein.
- Weiterhin dürfen keine Markup-Zeichen (`<` oder `&`) im Text vorkommen (diese können als `&lt;` oder `&amp;` geschrieben werden)
- Am Anfang des Dokuments sollte der Hinweis auf die XML-Version erfolgen: `<?xml version="1.0"?>`.

Mit Hilfe einer DTD ist es Programmen nun möglich, ein XML-Dokument auf strukturelle Fehler zu überprüfen oder eine neue Instanz dieses Dokumenttyps zu bilden. Das Quellcode-Beispiel 3.1 bindet die zugrundeliegende DTD über die Anweisung `<!DOCTYPE article SYSTEM "article.dtd">` aus der externen Datei `article.dtd` ein. Es ist aber auch möglich, die DTD direkt im XML-Dokument zu definieren. Dies ist allerdings höchstens für Entwicklungszwecke sinnvoll, da so kein anderes Dokument die DTD mitbenutzen kann. Über eine Kombination der beiden Möglichkeiten lässt sich jedoch eine externe DTD intern erweitern.

Quellcode 3.2 listet den Inhalt der DTD des ersten XML-Beispiels auf.

Mit der Anweisung `<!ELEMENT name inhalt>` werden Elementtypen deklariert. Als `inhalt` kann entweder eines der Schlüsselwörter `EMPTY` (kein Inhalt, „empty element tag“) oder `ANY` (beliebiger Inhalt) oder ein sogenanntes *content model* (Inhaltsmodell) angegeben werden. Inhaltsmodelle erinnern in ihrem Aufbau an reguläre

```
<!ELEMENT article   (title,author,(chapter+,newpage?)*)>
<!ATTLIST article   changed  CDATA  #REQUIRED>
<!ELEMENT title     (#PCDATA)>
<!ELEMENT author    (#PCDATA)>
<!ELEMENT chapter   (title,paragraph+)>
<!ATTLIST chapter   number   CDATA  "">
<!ELEMENT newpage   EMPTY>
<!ELEMENT paragraph (#PCDATA)>
```

Quellcode 3.2: DTD des ersten XML-Beispiels

Ausdrücke. Durch Klammern gruppiert und durch Kommata getrennt werden die Elemente angegeben, die nacheinander als Inhalt des Elements auftreten (`A,B,C,D`). Darf an beliebiger Stelle Text stehen, so verwendet man `#PCDATA`[10]. Es ist möglich, Inhalte der Elemente als optional (`A?`), als alternativ (`A|B`), mit mindestens einem Vorkommen (`A+`) oder beliebig vielen (`A*`) Wiederholungen vorzugeben.

Zu jedem Element kann eine Attributliste deklariert werden. Zu jedem Attribut muss ein Name und ein Typ angegeben werden. Der Attributtyp für einen einfachen Textwert lautet `CDATA`. Der Attributtyp *Enumerated* bezeichnet eine Liste möglicher Werte für das Attribut, die durch senkrechte Striche getrennt werden. Das Verweisen auf Elemente eines Dokuments ist über die Attributtypen `ID` und `IDREF` möglich.

Zu jedem Attribut muss schliesslich noch ein Defaultwert angegeben werden. Alternativ existieren die Schlüsselworte `#REQUIRED` (Wert für das Attribut muss im Dokument immer angeben werden), `#IMPLIED` (der Parser setzt einen sinnvollen Wert ein) und `#FIXED` (fest vorgegebener Wert). Dazu ein Beispiel:

```
<!ATTLIST element optional     CDATA   #IMPLIED
                 mandatory    (a|b|c) #REQUIRED
                 unchangeable CDATA   #FIXED    "123"
                 default      (0|1)   "0">
```

Ein Nachteil von DTDs: Wohlbekannte Datentypen wie Integer oder Float stehen zur genaueren Definition von Attributen nicht zur Verfügung, was eine erhebliche Einschränkung beim Einsatz von XML-Dokumenten zur Datenspeicherung und zum Datenaustausch sein kann. Außerdem genügen DTDs selbst nicht der XML-Syntax. *XML-Schema* ist ein noch in der Entwicklung befindlicher Ansatz, der diese Nachteile beheben soll.

[10] PCDATA steht für *parsed character data*. Im Gegensatz zu PCDATA wird CDATA nicht mehr vom Parser analysiert. Dadurch werden innerhalb von CDATA keine Markup-Zeichen wie spitze Klammern erkannt.

Stylesheets

Die Darstellung eines XML-Dokuments erfolgt mit Hilfe einer Formatvorlage, eines *Stylesheets*. In diesem Stylesheet wird das Layout des Dokuments festgelegt. XML zeichnet sich ja gerade durch die unendliche Menge möglicher Tags aus, so dass in einer Applikation (z.B. einem Webbrowser) unmöglich ein Layout für die Tags festgelegt sein kann, wie dies beispielsweise für HTML der Fall ist. Durch Stylesheets können sowohl die Autoren als auch die Nutzer von Webdokumenten deren Präsentation beeinflussen, ohne dabei auf eine geräte- oder anwendungsunabhängige Weiterverarbeitbarkeit verzichten zu müssen [W3C2000c]. So kann die Verwendung unterschiedlicher Stylesheets zur Anpassung eines Dokuments für verschiedene Zwecke, wie beispielsweise Ausdruck und Bildschirmanzeige, genutzt werden. Auf Stylesheets wird durch entsprechende processing instructions verwiesen, z.B.:

```
<?xml-stylesheet type="text/xsl" href="mystylesheet.xsl"?>
```

Das W3C entwickelt derzeit mit der *eXtensible Stylesheet Language* (XSL) eine eigene Stylesheet Sprache für XML [W3C2000d]. Daneben wird seit 1996 die Entwicklung der *Cascading Stylesheets* (CSS) weitergeführt, einer Stylesheet Sprache, die sowohl mit XML als auch HTML verwendet werden kann. XSL kann im Gegensatz zu CSS Dokumente transformieren. Z.B. ist es mit XSL möglich, die Reihenfolge von Elementen zu verändern.

XSL gliedert sich in eine bereits standardisierte Transformationssprache *XSL Transformations* (XSLT) und eine Formatierungssprache, die *Formatting Objects* (XSL-FO). Beide Sprachen sind XML-Anwendungen, d.h. wieder in XML definiert. Mit den Elementen der Transformationssprache XSLT können Regeln definiert werden, die angeben, wie ein XML-Dokument in ein anderes überführt werden soll. Abschnitt 3.6 beschäftigt sich ausführlicher mit dieser Technik.

Die zweite Hälfte der eXtensible Stylesheet Language ist die Formatierungssprache XSL-FO. Sie stellt ein Vokabular zur Verfügung, mit dem eine Formatierung von Elementen medienunabhängig beschrieben werden kann. Die Semantik der Sprachelemente, die Formatting Objects, ist durch die Spezifikation genau definiert. Dies umfasst beispielsweise die Angabe von Abständen und Schriftarten oder die Festlegung des Seitenlayouts. Im Allgemeinen verwendet ein Stylesheet die XSL-Transformationssprache (XSLT), um ein XML-Dokument in ein neues XML-Dokument umzuwandeln, welches die XSL-Formatierungsobjekte und so Vorgaben hinsichtlich der beabsichigten Präsentation enthält. XSL-FO besitzt zur Zeit noch den Status *W3C working draft* [W3C2000e].

Document Object Model (DOM)

Ein XML-Dokument ist eine Textdatei, die neben dem Inhalt (den reinen Daten) auch dessen Struktur, oder allgemeiner Metadaten, enthält. Betrachtet man ein XML-Dokument genauer, kann man die innere Struktur als *Baum* auffassen. Eine Baumstruktur beginnt mit einer Wurzel, die selbst keine Vorgänger besitzt. Aus der Wurzel entspringen beliebig viele Äste, die sich wiederum gabeln können bis zu den Blättern. Zur leichteren Betrachtung fasst man Wurzel, Gabelungen und Blätter als *Knoten* (*Nodes*) auf, die Äste sind die Verbindungen zwischen den Knoten.

Das Auffassen der logischen Dokumentstruktur als Baum trifft keinerlei Aussage darüber, wie ein XML-Dokument von einem Programm tatsächlich in konkreten Datenstrukturen repräsentiert wird. Wichtig ist aber im Zusammenhang mit anderen Anwendungen, die XML-Dokumente verarbeiten, dass eine Schnittstelle zur XML-Datenstruktur angeboten wird, welche die logische Abarbeitung des Dokuments als Baum ermöglicht. Eine Vereinbarung über eine solche Baumschnittstelle zu XML-Dokumenten wird *Document Object Model* (DOM) genannt.

Das Document Object Model des W3C wurde allgemein als ein standardisiertes Datenzugriffsmodell entwickelt. Es ist eine plattform- und sprachunabhängige Schnittstelle, die es Programmen und Skriptsprachen erlaubt, dynamisch auf den Inhalt und die Struktur eines XML- oder HTML-Dokuments zuzugreifen und sie zu verändern [W3C2000f].

Wie schon der Name sagt, ist das DOM ein Objektmodell, das die Baumstruktur des Dokuments in einer Objekthierarchie abbildet. Die Objekte selbst sind keine statischen Datenstrukturen, sondern bieten genau definierte Schnittstellen an, die das Auslesen und Verändern des Objektinhalts und der Objekthierarchie ermöglichen. Die DOM-Spezifikation des W3C hat die Aufgabe, diese Schnittstellen (*Interfaces*) für die unterschiedlichen Objekttypen plattformunabhängig zu definieren.

Bis auf wenige Ausnahmen stammen alle Interfaces des DOM vom Node-Interface ab. Das Node-Interface repräsentiert einen Baumknoten und bietet Methoden zur Navigation in der Objekthierarchie. Weitere Interfaces, die die Bestandteile eines Dokuments (Element, Attribut, Text) charakterisieren, sind direkt vom Node-Interface abgeleitet. Für jeden Objekttyp ist genau festgelegt, welche anderen Objekttypen die untergeordneten Bestandteile bilden dürfen. Hierdurch ist der Aufbau des Objektmodells bestimmten Regeln unterworfen, die garantieren, dass nur wohlgeformte XML-Dokumente dargestellt werden können.

Die aus Quellcode 3.1 resultierende Objekthierarchie stellt Abbildung 3.3 in einem sogenannten Strukturmodell dar. Element-Objekte, die die innerern Knoten der

Hierarchie bilden, sind orange dargestellt, Attribut-Objekte gelb und Text-Objekte, die die Blätter des Baums bilden, blau.

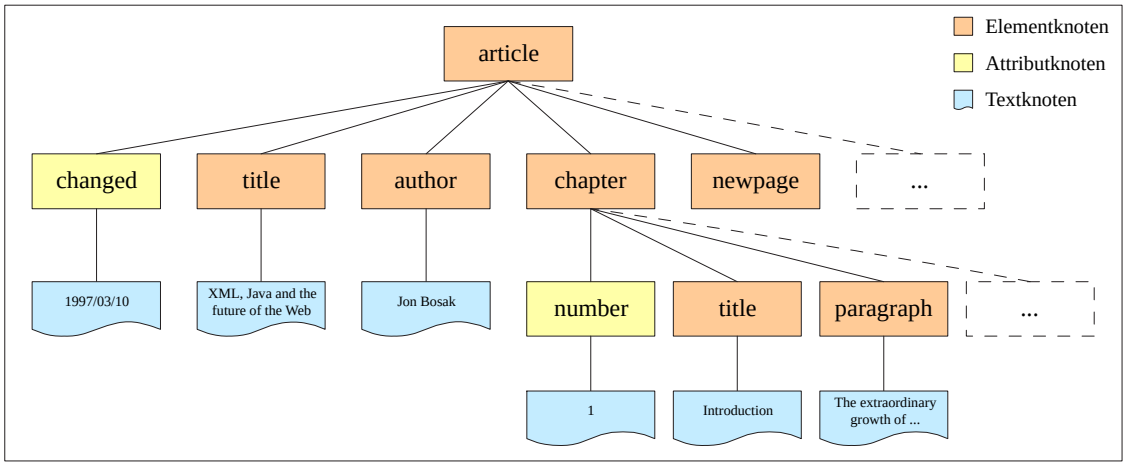


Abbildung 3.3: Baumstruktur des ersten XML-Beispiels

Am 13. November 2000 hat das W3C bereits eine Erweiterung des DOM, die *Level 2 Specification*, als *Recommendation* - die höchste Stufe eines W3C-Standards - vorgelegt [W3C2000f]. Sie enthält z.B. zusätzliche Schnittstellen für Stylesheets und zum leichteren Traversieren des Objektbaums.

Weitere Techniken

Wie bereits dargestellt, ist XML weit mehr als eine Auszeichnungssprache. Neben der eigentlichen Spezifikation XML 1.0, die definiert, was Tags und Attribute sind, existiert eine wachsende Zahl zusätzlicher Technologien: Eine erweiterte Form der Hyperlinks (*XLink*) existiert für XML, Verweise auf bestimmte Punkte eines XML-Dokuments sind von ausserhalb möglich (*XPointer* basierend auf *XPath*), Namensräume unterstützen die Koexistenz unterschiedlicher XML-Dokumente (*XML Namespaces*) und genaue Typdefinitionen in XML-Dateien (*XML Schema*) befinden sich in der Entwicklungsphase. Diese XML-Techniken sind jedoch für den Inhalt dieser Arbeit nicht relevant und werden deshalb nicht genauer erläutert. Einen guten Überblick gibt [Beck2000].

3.4 Beurteilung von XML

Vor- und Nachteile von XML

XML ist vom W3C mit dem Ziel entwickelt worden, ein offenes Datenformat für über das Web nutzbare Dokumente zu bieten. Dabei, so zwei der Designziele, solle

es sich im Internet auf einfache Weise nutzen lassen und ein breites Spektrum von Anwendungen unterstützen [W3C2000b]. Bislang fehlte eine Möglichkeit, Daten, sei es auf Webseiten oder für unterschiedliche Anwendungen, allgemeinverständlich zu beschreiben. Es fehlte ein universelles Datenformat. Sollte sich die Erfolgsgeschichte von XML weiter fortsetzen, was allgemein erwartet wird, könnte sich die Sprache zum zukünftigen Standard der Datenbeschreibung im Internet entwickeln.

Da XML noch sehr jung ist, befinden sich viele Implementierungen noch in der Entwicklungsphase. Wie bei jeder neuen Technologie besteht auch bei XML Unsicherheit, wie und wofür die Sprache sich wird durchsetzen können. Alle größeren Firmen haben jedoch inzwischen XML-Unterstützung oder XML-Schnittstellen für ihre Produkte angekündigt und leistungsfähige XML-Parser existieren kostenlos von Apache, IBM, Oracle und Sun.

Dass XML ein textbasiertes Format ist, birgt den Nachteil, dass XML-Dateien fast immer mehr Speicherplatz als vergleichbare Binärdateien beanspruchen. Doch Festplattenplatz ist heute nicht mehr teuer. Außerdem können kostenlose Komprimierungsprogramme eingesetzt werden und moderne Kommunikationsprotokolle beherrschen Datenkompression „on the fly“. Der große Vorteil eines textbasierten Formats ist, dass es mit einem einfachen Texteditor bearbeitet werden kann - auch wenn XML dafür eigentlich nicht gedacht ist.

Ein häufig unterschätztes Argument für XML ist die Unterstützung internationaler Zeichensätze. Das Internet und die weltweite Kommunikation überwinden Länder- und Sprachgrenzen. Daher unterstützt XML in der Version 1.0 den erweiterten Unicode-Zeichensatz. So ist XML genau wie Java unabhängig von der verwendeten Plattform einsetzbar. Der berühmte Satz des XML-Mitentwicklers Jon Bosak „XML gives Java something to do“ [Bosa1997] resultiert aus der Fähigkeit von XML, nicht nur Textdaten, sondern „grundsätzlich beliebige, textuell codierbare (also theoretisch alle) Formen von Daten zu speichern. (...) XML könnte also neben der universellen, systemunabhängigen Sprache Java das universelle Datenformat des Web werden.“ [BeMi1998].

Anwendungsfelder für XML

Vor dem Hintergrund der oben beschriebenen allgemeinen Vorteile einerseits und der Beschränktheit von HTML andererseits lassen sich vier besonders geeignete Anwendungsfelder für XML identifizieren:

- **XML als standardisiertes Datenaustauschformat**
  XML kann als Datenaustauschformat zwischen unterschiedlichen Datenbeständen dienen.

- **Verlagerung der Rechenleistung vom Server zum Client**
  Durch die Möglichkeit, alle Daten, die zu einem Dokument gehören, in XML zu codieren, können alle zum Verarbeiten des Dokuments notwendigen Informationen übertragen werden und beim Client entsprechend ausgeführt werden.
- **Variable Darstellung von Informationen**
  Auf XML-Dokumente können in Abhängigkeit vom jeweiligen Anwendungszweck unterschiedliche Sichten angenommen werden. Dokumente können unterschiedlich angezeigt und ausgedruckt werden.
- **Intelligentes Suchen nach Informationen in XML-Datenbeständen**
  Intelligente Softwareagenten können dank der zusätzlich vorhandenen Metainformationen die Bedeutung von in XML codierten Informationen verstehen. So wird z.B. eine bessere Indizierung von Webseiten (allgemeiner von Produkten) möglich, um das bekannte Problem der Fehltreffer in Suchmaschinen zu lösen.

Mit der konkreten Verwendung der XML-Technologie im Rahmen dieser Diplomarbeit beschäftigen sich nun die folgenden Abschnitte. Ein XML-Parser wird benötigt, um ein auf XML basierendes Datenformat verarbeiten zu können. Abschnitt 3.5 stellt daher die Verwendung des XML-Parsers der Apache Software Foundation vor. Ein XML-Dokument eines fremden Dokumenttyps lässt sich mit den Regeln der *eXtensible Stylesheet Language Transformations* (XSLT) in das gewünschte Format umwandeln. Wie die nötigen Transformationsregeln zu formulieren sind, erläutert Abschnitt 3.6. Und einen Ansatz, wie theoretisch eine beliebige XML-Quelle zu einem Flash-Film wird, liefert Abschnitt 3.7 zusammen mit der Vorstellung der verwendeten Software.

3.5 XML-Parser

XML besteht aus einigen einfachen Regeln. Die innere Struktur eines XML-Dokuments ist sehr klar und einfach zu verstehen. Aus der simplen Struktur der Dokumente und der Tatsache, dass die Auszeichungen durch die Klammerung (`<...>`) leicht vom Inhalt zu trennen sind, ergibt sich ein großer Vorteil für die Softwareindustrie.

Zum einen ist die manuelle und maschinelle Erstellung von XML-Dokumenten mit geringem Aufwand zu programmieren. Dokumente können zur Not noch mit einem simplen Texteditor geschrieben werden. Zum anderen hat die einfache Struktur Bedeutung für die Herstellung von Software, die XML-Dokumente verarbeiten kann.

Grundlage für die Verarbeitung von strukturierten semantischen Datenströmen sind sogenannte *Parser*, die die einzelnen Elemente einer Sprache aus dem Quelltext isolieren, auf syntaktische Korrektheit prüfen und Aktionen auf den Elementen durchführen können. Vereinfacht betrachtet trennt ein XML-Parser die Auszeichnung der Elemente vom Inhalt und stellt die Daten über eine Schnittstelle weiteren Anwendungen zur Verfügung. Die Verarbeitung des Inhalts auf Grund der erkannten Auszeichnungen wird im Allgemeinen erst auf einer höheren Softwareebene durch sogenannte *Prozessoren* (z.B. bei Stylesheets durch einen XSL-Prozessor) durchgeführt.

Da es in XML nur wenige unterschiedliche Elementtypen gibt, können sowohl Parser als auch Prozessoren in ihrer Grundstruktur einfach, damit robust und kostengünstig, realisiert werden. Zudem verarbeitet ein XML-Parser beliebige auf XML basierende Sprachen (von Neuerungen wie z.B. XML-Schema abgesehen). Bedingt durch die Tatsache, dass sich XML-Parser relativ leicht schreiben lassen und vielfältig einsetzbar sind, existieren heute zahlreiche frei verfügbare XML-Parser für unterschiedliche Sprachen wie Java, C++, Perl und Python.

Für Programmierschnittstellen (*Applicaton Programming Interface*, kurz API) von XML-Parsern existieren zwei Standards: DOM und SAX. Ein Parser, der die DOM-API implementiert, bietet eine Funktion, die das XML-Dokument komplett einliest und im Speicher eine Repräsentation des Dokuments aufbaut. Über die bereits erwähnten Interfaces des DOM (vgl. Seite 32) lässt sich der Inhalt der Baumstruktur lesen und verändern.

Da DOM-Parser immer das komplette Dokument einlesen und im Speicher halten, sind sie für sehr große Daten eher ungeeignet. Andererseits ermöglicht das DOM, Dokumente zu manipulieren oder neu zu erzeugen, indem man das Dokument im Speicher bearbeitet und anschließend abspeichert.

Die zweite Programmierschnittstelle ist das SAX, die *Simple API for XML*. SAX ist ein De-facto-Standard, der nicht vom W3C sondern im Rahmen der XML-DEV Mailingliste entwickelt wurde [Megg2000]. Das XML-Dokument wird dabei sequentiell abgearbeitet und der Parser informiert über Ereignisse, die *parsing events*. D.h. SAX arbeitet ereignis-gesteuert und man übergibt dem Parser vor dem Start eine Reihe von Callback-Funktionen, die bei jedem Ereignis aufgerufen werden. Zu den parsing events gehören z.B. das Erkennen von Start- oder Ende-Tags oder das Eintreten gewisser Fehlerzustände.

SAX ist schnell und braucht wenig Speicher. Allerdings kann man mit SAX Dokumente nur einlesen, sie können weder manipuliert noch gespeichert werden, da kein

Abbild des Dokuments im Speicher erstellt wird.

Allgemein legt die XML 1.0 Spezifikation fest, dass ein Parser keine Korrektur von Fehlern im XML-Dokument vornehmen darf. Im Gegensatz zu einem HTML-Browser, den z.B. fehlende Ende-Tags oder falsch geschachtelte Tags nicht von der Darstellung der Webseite abhalten, muss ein XML-Parser die Bearbeitung des Dokuments sofort mit einer Fehlermeldung beenden, sofern es nicht wohlgeformt (vgl. Seite 28) ist.

Ein sehr leistungsfähiger XML-Parser, der im Rahmen dieser Arbeit verwendet wird, ist der *Apache Xerces-J* [Apac2000b]. Er ist in Java implementiert und unterstützt in der Version 1.2.1 sowohl die SAX- als auch die DOM-Schnittstelle (jeweils Level 1 und 2). Eine Überprüfung der XML-Dokumente auf Gültigkeit kann wahlweise erfolgen, d.h. der Parser ist ein *validierender* Parser. Xerces-J entstand ursprünglich aus dem *IBM XML4J* XML-Parser und wird inzwischen von der *Apache Software Foundation*[11] weiterentwickelt. Die Quellen des Parsers sind frei verfügbar und dürfen im Rahmen der *Apache Software License* [Apac2000c] verwendet, verändert und weitergegeben werden.

Die Verwendung eines DOM-Parsers am Beispiel des Apache Xerces-J soll im Folgenden verdeutlicht werden. Nach dem Import der Parser-Klasse `org.apache.xerces.parsers.DOMParser` und dem Document-Interface `org.w3c.dom.Document`, das die Wurzel des Dokumentbaums repräsentiert, wird ein XML-Dokument mit wenigen Zeilen Quellcode in ein DOM überführt:

```
DOMParser parser = new DOMParser();
parser.parse( xmldoc_filename );
Document doc = parser.getDocument();
```

Angefangen von der Dokument-Wurzel `doc` kann nun der Baum traversiert und jedes Element ausgelesen, manipuliert und die Baumstruktur verändert werden.

Standardmäßig überprüft der Xerces-J Dokumente nur auf ihre Wohlgeformtheit. Möchte man die strengere Gültigkeitsprüfung durchführen, muss das entsprechende Feature vor dem Parsen gesetzt werden:

```
parser.setFeature("http://xml.org/sax/features/validation", true );
```

[11] Die *Apache Software Foundation* (ASF) ist eine Open-Source Vereinigung, die aus freiwilligen Mitgliedern besteht. Ursprünglich wurde sie unter dem Namen *Apache Group* mit ihrem kostenlosen Webserver bekannt - dem meistverwendeten Webserver im Internet. Heute arbeitet die ASF im Rahmen des *Apache XML Project* zusätzlich an verschiedenen Implementationen zum Thema XML. [Apac2000a]

Meldungen zu Fehlern, die während des Parsens auftreten, werden normalerweise nicht ausgegeben. Dazu bedarf es eines *ErrorHandlers*, der dem Parser folgendermaßen mitgeteilt wird:

```
parser.setErrorHandler( myErrorHandler );
```

Das Interface `org.xml.sax.ErrorHandler` vereinbart die Methodenköpfe, die ein ErrorHandler implementieren muss. Sie lauten:

```
void warning( SAXParseException e )    // warning
void error( SAXParseException e )      // recoverable error
void fatalError( SAXParseException e ) // non-recoverable
```

Dass der ErrorHandler auch für einen DOM-Parser aus dem Paket `org.xml.sax` stammt, ist dadurch bedingt, dass die meisten DOM-Parser auf einem SAX-Parser basieren. Sie lassen den SAX-Parser das Dokument sequentiell verarbeiten und bauen daraus - gesteuert durch die Ereignisse, die der SAX-Parser auslöst - das DOM auf. Die Methoden des ErrorHandlers ruft also eigentlich ein versteckter SAX-Parser auf.

Quellcode 3.3 auf Seite 39 zeigt die genannten Befehle im Zusammenhang. Das Beispiel bekommt den Namen einer XML-Datei als Kommandozeilenparameter und erzeugt das entsprechende DOM des Dokuments mit Hilfe des Xerces-J.

3.6 XSLT als Übersetzer

Das XML-Beispiel auf Seite 27 enthält neben dem Inhalt auch die Struktur eines Artikels. Die verwendeten Auszeichnungen beschreiben die Bedeutung der einzelnen Textpassagen (Titel, Autor, Abschnitte, ...). Die zugrundeliegende Sprache, eine XML-Anwendung, ist jedoch frei erfunden und exakt an die eigenen Bedürfnisse angepasst. Kein Texteditor oder Webbrowser kann mit ihrer Semantik etwas anfangen - sie verstehen die Bedeutung der Tags nicht.

Die Spezifikation der *eXtensible Stylesheet Language Transformations* (XSLT) definiert Syntax und Semantik einer Sprache, mit der sich XML-Dokumente in eine andere Sprache übersetzen lassen. Dabei handelt es sich bei den Transformationen, die XSLT vornimmt, um die Wandlung eines Dokumentquellbaums (*source tree*) in einen Ergebnisbaum (*result tree*). Das Ausgabeformat des Transformationsprozesses kann ein - gewöhnlich auf einer anderen DTD basierendes - XML-Dokument, ein HTML-Dokument oder jedes beliebige Unicode-Textformat sein (siehe Abbildung 3.4).

```
import java.io.IOException;
import org.apache.xerces.parsers.DOMParser;
import org.w3c.dom.Document;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

/**
 * Eigener ErrorHandler zur Fehlerausgabe
 */

class MyErrorHandler implements ErrorHandler {

  public void warning( SAXParseException e ) {
    System.err.println("XML PARSER WARNING:\n" + e.getMessage());
  }

  public void error( SAXParseException e ) {
    System.err.println("XML PARSER ERROR:\n" + e.getMessage());
  }

  public void fatalError( SAXParseException e )  {
    System.err.println("XML PARSER FATAL ERROR:\n" + e.getMessage());
  }
}


/**
 * Beispiel zur Verwendung der API des Xerces-DOMParsers
 */

public class DOMParserExample {

  public static void main(String[] args) throws IOException,
                                                 SAXException {
      DOMParser parser = new DOMParser();

      parser.setErrorHandler( new MyErrorHandler() );
      parser.setFeature("http://xml.org/sax/features/validation",true);
      parser.parse( args[0] );

      Document doc = parser.getDocument();

      // Verarbeitung des Dokuments
  }

}
```

Quellcode 3.3: Verwendung eines DOM-Parsers am Beispiel des Xerces-J

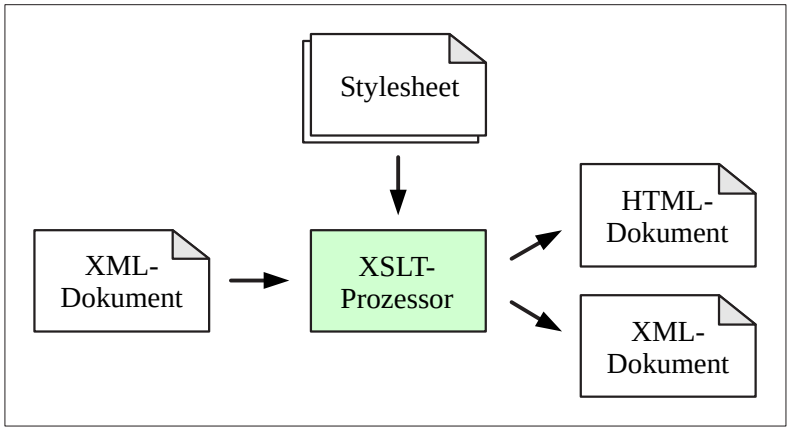


Abbildung 3.4: Verwendung eines XSLT-Prozessors

Um das Beispiel von Seite 27 in ein entsprechendes HTML-Dokument umzuwandeln, benötigt man z.B. das Stylesheet aus Quellcode 3.4 auf Seite 41. Es ist selbst ein wohlgeformtes XML-Dokument, das sich aus Elementen des XSLT *Namespace* mit dem Präfix „xsl“ und anderen (Ergebnis-) Elementen zusammensetzt.

Das Wurzelelement `xsl:stylesheet` legt zu Beginn fest, dass der XSL-Namensraum der von XSLT in der Version 1 ist. Innerhalb dieses Wurzelelements können die folgenden *Top-Level-Elemente* vorkommen [Behm1999]:

```
<xsl:import href="..."/>
<xsl:include href="..."/>
<xsl:strip-space elements="..."/>
<xsl:preserve-space elements="..."/>
<xsl:output method="..."/>
<xsl:key name="..." match="..." use="..."/>
<xsl:locale name="...">...</xsl:locale>
<xsl:attribute-set name="...">...</xsl:attribute-set>
<xsl:variable name="...">...</xsl:variable>
<xsl:param name="...">...</xsl:param>
<xsl:template name="...">...</xsl:template>
<xsl:template match="...">...</xsl:template>
```

Die zentrale Rolle für den Umwandlungsprozess spielen die zuletzt genannten *Template*-Regeln. Jedes Template-Element ist dabei entweder benannt oder enthält im `match`-Attribut ein Pattern (Muster), das spezifiziert, welche Inhalte des Ausgangsdokuments durch neue ersetzt werden. Die Selektion von Elementen des Ausgangsdokuments erfolgt dabei anhand der Knoten des entsprechenden Strukturbaums über die Schnittstellen des Document Object Models. Daher enthalten die Pattern

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

  <xsl:output method="html"/>

  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE>
          <xsl:apply-templates select="article" mode="head"/>
        </TITLE>
      </HEAD>
      <BODY>
        <xsl:apply-templates select="article" mode="body"/>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="article" mode="head">
     <xsl:value-of select="author"/>
     <xsl:text>: </xsl:text>
     <xsl:value-of select="title"/>
  </xsl:template>

  <xsl:template match="article" mode="body">
    <CENTER>
      <H1><xsl:value-of select="title"/></H1>
      <H3>by <xsl:value-of select="author"/></H3>
    </CENTER>
    <xsl:apply-templates select="chapter"/>
    <HR/>
    Last changed: <xsl:value-of select="@changed"/>
  </xsl:template>

  <xsl:template match="chapter">
    <P>
      <H3><xsl:value-of select="title"/></H3>
      <xsl:apply-templates select="paragraph"/>
    </P>
  </xsl:template>

  <xsl:template match="paragraph">
    <xsl:value-of select="."/><BR/>
  </xsl:template>

</xsl:stylesheet>
```

Quellcode 3.4: Stylesheet für das erste XML-Beispiel

Verweise auf Baumknoten in Form von Pfadangaben entlang der Baumstruktur, deren Syntax die *XPath*-Spezifikation definiert. Tabelle 3.1 zeigt mögliche Muster zur Selektion von Elementen.

XPath-Angabe	Bedeutung
`/`	das Wurzelelement
`*`	jedes Element
`@*`	jedes Attribut
`@class`	jedes Attribut `class`
`para`	jedes Element `para`
`para|indent`	jedes Element `para` oder `indent`
`list/elem`	jedes Element `elem` als direktes Kind von `list`
`article//para`	jedes Element `para` unterhalb von `article`
`../@class`	Attribut `class` des übergeordneten Elements
`text()`	jedes Textelement
`node()`	jeder Knoten; keine Attribute, nicht die Wurzel
`id("a1")`	Element mit der `id` „a1“
`para[1]`	jedes Element `para`, das das erste Kind seines übergeordneten Elements ist
`*[position()=1 and self::para]`	dito
`para[last()=1]`	Element `para`, das das einzige Kindelement des übergeordneten Elements ist
`list/elem[position()>1]`	jedes Element `elem` als Kind einer `list`, das nicht deren ersten Kindelement ist

Tabelle 3.1: Beispiele für XPath-Angaben

Die Transformation eines XML-Dokuments anhand der Anweisungen eines Stylesheets übernimmt ein *XSLT-Prozessor*. Ein solcher Prozessor vergleicht die gegebenen Muster mit den Knoten des Quellbaums und generiert den Ergebnisbaum, indem er an den zutreffenden Stellen die entsprechenden templates (Schablonen) anwendet. Dabei werden nur Elemente berücksichtigt, für die Anweisungen im Stylesheet existieren oder eine Standardregel zutrifft (und nicht überdefiniert wurde). Die Standardregeln definieren eine rekursive Abarbeitung aller Elemente beginnend mit der Wurzel, bei der Textelemente und Attributwerte (sofern das Attribut selektiert wird) ausgegeben bzw. processing instructions und Kommentare übergangen werden. Im Einzelnen lauten die Regeln:

1. Rekursion durch den kompletten Baum:

```
<xsl:template match="*|/">
  <xsl:apply-templates/>
</xsl:template>
```

2. Ausgabe von Textelementen und Attributwerten:

```
<xsl:template match="text()|@*">
  <xsl:value-of select="."/>
</xsl:template>
```

3. Keine Verarbeitung von processing instructions und Kommentaren:

```
<xsl:template match="processing-instruction()|comment()"/>
```

Kommt es zu Konflikten zwischen verschiedenen Template-Regeln, die dasselbe Element betreffen, hat eine genauer spezifizierte Regel (z.B. `article/para` im Vergleich zu `para`) die höhere Priorität. Stammen die Regeln aus unterschiedlichen Stylesheets, wird diejenige ausgeführt, die zuletzt eingelesen wurde.

XSLT bietet weit mehr Möglichkeiten, als am Beispiel gezeigt werden konnte. Die XPath-Spezifikation bietet weitere Funktionen zur Selektion von Knoten und es gibt die Möglichkeit eigene Funktionen zu integrieren. Das Erzeugen von Knoten, die Numerierung (`xsl:number`-Tag) und die bedingte Verarbeitung (`xsl:if` und `xsl:choose`) sind mit XSLT ebenso möglich wie Sortieren und die Übergabe von Parametern an Template-Regeln [Behm1999].

XSLT-Prozessoren

XSLT-Prozessoren sind für die meisten Plattformen frei aus dem Internet zu beziehen. Auch die Apache Software Foundation hat unter dem Namen *Xalan* einen XSLT-Prozessor entwickelt. Xalan-Java 1.2.1 arbeitet standardmäßig mit dem Apache Xerces-J zusammen, kann aber mit jedem anderen XML-Parser eingesetzt werden, der DOM Level 2 und SAX Level 1 konform ist.

Die Verwendung des Xalan-Java XSLT-Prozessors soll an einem Beispiel verdeutlicht werden.

Das Stylesheet von Seite 41 enthielt bereits alle Anweisungen, die nötig sind, um das XML-Beispiel von Seite 27 zur Präsentation in einem Webbrowser in eine HTML-Datei umzuwandeln. Die Implementation einer einfachen Transformationsklasse, die die Xalan-API verwendet, zeigt nun Quellcode 3.5. Startet man die Klasse mit dem XML-Dokument und dem Stylesheet als Kommandozeilenparameter, erhält man als Ausgabe Quellcode 3.6. Die Darstellung der HTML-Ausgabe im Webbrowser zeigt abschließend Abbildung 3.5.

```
import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.XSLTProcessor;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;

/**
 * Beispielklasse zur Demonstration der API des Xalan-XSLT-Prozessors.
 * Aufruf: java SimpleTransform document.xml stylesheet.xsl
 */

public class SimpleTransform {

  public static void main(String[] args) throws
                                        java.io.IOException,
                                        java.net.MalformedURLException,
                                        org.xml.sax.SAXException {
    // erzeuge einen XSLT-Prozessor
    XSLTProcessor processor = XSLTProcessorFactory.getProcessor();

    // transformiere XML-Dokument args1 anhand der XSLT-Regeln aus
    // args2 und schreibe das Ergebnis auf die Standardausgabe
    processor.process( new XSLTInputSource(args[0]),
                       new XSLTInputSource(args[1]),
                       new XSLTResultTarget(System.out) );
  }
}
```

Quellcode 3.5: Verwendung eines XSLT-Prozessors am Beispiel des Xalan-Java

```
<HTML>
    <HEAD>
        <TITLE>Jon Bosak: XML, Java and the future of the Web</TITLE>
    </HEAD>
    <BODY>
        <CENTER>
            <H1>XML, Java and the future of the Web</H1>
            <H3>by Jon Bosak</H3>
        </CENTER>
        <P>
            <H3>Introduction</H3>
            The extraordinary growth of ...<BR>
        </P>
        <HR>
        Last changed: 1997/03/10
    </BODY>
</HTML>
```

Quellcode 3.6: Ausgabe des XSLT-Prozessors für das erste XML-Beispiel

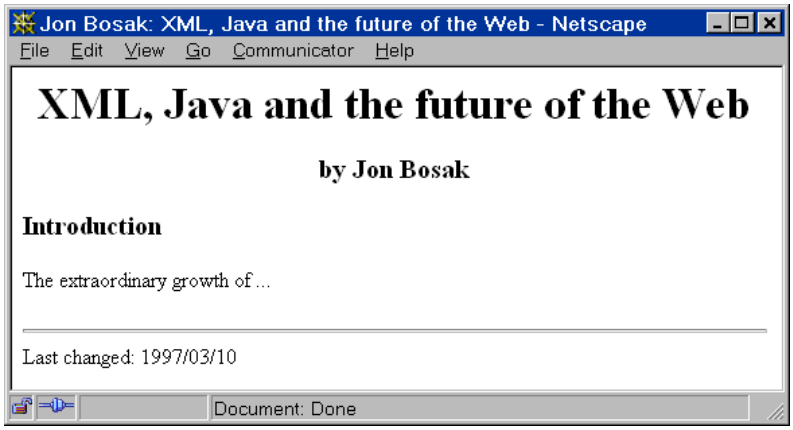


Abbildung 3.5: Darstellung des transformierten XML-Beispiels im Webbrowser

3.7 XML und SWF

Ein Blick zurück an den Anfang des Kapitels: Das Ziel war die Entwicklung eines textbasierten Dateiformats, das als Schnittstelle zwischen Eingabedaten und Visualisierungsmodul, d.h. der Erzeugung der Flash-Filme, existieren sollte. XML ist ein Standard, der alle Mittel bereitstellt, um ein solches Format exakt zu spezifizieren. Frei verfügbare Software ist in der Lage, beliebige auf XML basierende Sprachen zu verarbeiten, und kann verwendet werden, um Dokumente einzulesen, ihre Struktur und ihren Inhalt zu manipulieren und wieder auszugeben. Die Transformation von einer XML-Sprache in eine andere bzw. in ein Nicht-XML-Format ist ebenfalls standardisiert und wird von Softwarepaketen unterstützt. So ist es also prinzipiell möglich, beliebige auf XML basierende Dateiformate mit den XML-Techniken in das eigene Schnittstellenformat zu konvertierern und für den Visualisierungsprozess zur Verfügung zu stellen.

Einen auf XML abgestimmten Entwurf des Visualisierungsprozesses zeigt Abbildung 3.6 (vgl. Abbildung 3.1 auf Seite 22).

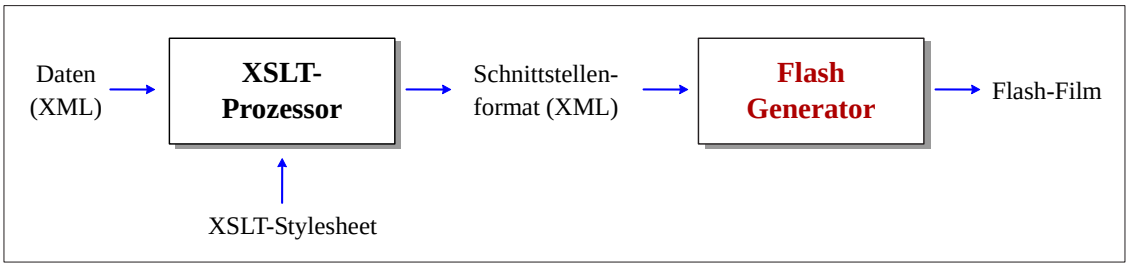


Abbildung 3.6: Auf XML abgestimmter Entwurf des Visualisierungsprozesses

Das fehlende Modul im vorliegenden Entwurf ist nun noch der Flash-Generator. Wie schon in Abschnitt 2.5 beschrieben, erfordert er recht aufwendige Implementie-

rungsarbeit, um in der Lage zu sein, textbasierte Daten mit Hilfe des Macromedia SWF-SDK in das bitcodierte Flash-Dateiformat zu überführen.

An diesem Punkt der Entwurfsphase stellte sich durch Recherche im Internet heraus, dass bereits eine Applikation existiert, die exakt die genannten eigenen Ideen umsetzt: *Saxess Wave*, eine in Java geschriebene Applikation der Firma Saxess Software Design aus Köln [Saxe2000].

Saxess Wave

Im ersten Schritt konvertiert Saxess Wave XML-Dokumente mit Hilfe von XSLT in ein ebenfalls auf XML basierendes Format namens *SWFML* (Shockwave Flash Markup Language), das das binäre Flash-Dateiformat abbildet. Als zweiten und entscheidenden Schritt erzeugt Saxess Wave aus den SWFML-Daten einen Flash-Film. Interessant ist dabei, dass Saxess Wave ohne das Macromedia SWF-SDK arbeitet, sondern die Erzeugung des binären Flash-Dateiformates komplett in eigenen Java-Klassen abwickelt.

Zusätzlich bietet Saxess Wave unter Zuhilfenahme des Apache FOP-Renderers an, XSL-Stylesheets, die Anweisungen der XSL Formatting Objects (XSL-FO) enthalten, nach SWFML zu konvertieren (zu rendern). Die Möglichkeit, über eine Schnittstelle aus den medienunabhängigen Formatierungsangaben der FO-Elemente Flash-Filme zu erzeugen, soll aber hier nicht weiter berücksichtigt werden, da XSL-FO noch nicht den Status eines Standards erreicht hat.

Den Datenfluss von Saxess Wave veranschaulicht Abbildung 3.7.

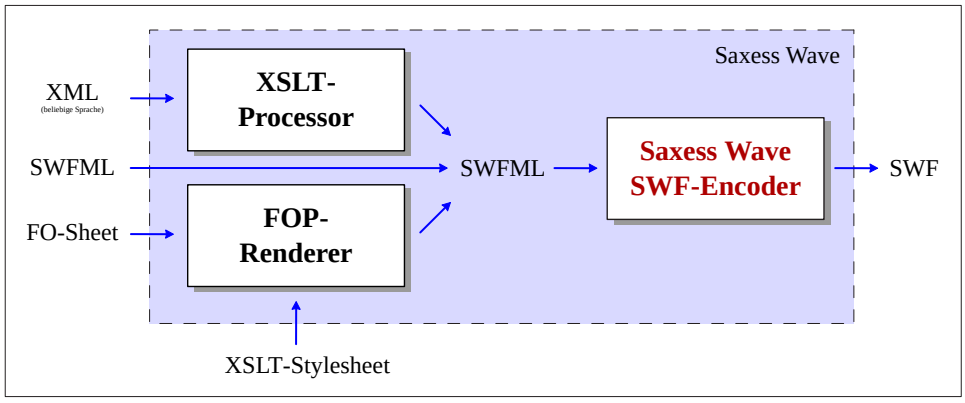


Abbildung 3.7: Datenfluss von Saxess Wave

Saxess Wave - der Flash-Generator? Bei einer genaueren Untersuchung wurde deutlich, dass Saxess Wave für einen ersten Ansatz als Visualisierungsmodul gut geeignet

ist, einige bedeutende Fähigkeiten von Flash (z.B. ActionScript) jedoch in der vorliegenden Version 0.5 nicht unterstützt. Saxess Wave 1.0 befindet sich laut Internet-Homepage in der Entwicklung und soll die volle Flash-Unterstützung enthalten. Es wird jedoch auch angekündigt, dass die Software dann nicht mehr als kostenlose Evaluationsversion sondern als Shareware zur Verfügung stehen wird [Saxe2000].

Trotz seiner Einschränkungen soll die Verwendung von Saxess Wave 0.5 vorgestellt werden, da seine Fähigkeiten zur Erzeugung von einfachen nicht-interaktiven Flash-Filmen ausreichen. Im Rahmen dieser Diplomarbeit wird es als Flash-Generator verwendet. Eine nachfolgende Diplomarbeit meines Kommilitonen Ralf Kunze, ebenfalls bei Herrn Prof. Dr. Vornberger, hat die Entwicklung eines umfangreicheren Flash-Generators als Aufgabenstellung, der auf Basis des SWF-SDK den kompletten Funktionsumfang (Interaktivität, Bitmapgrafiken u.a.) bieten wird.

SWFML

Das in XML formulierte Schnittstellenformat SWFML von Saxess Wave deckt die grundlegenden SWF-Tags ab. Außerdem formuliert es in der zugrunde liegenden DTD einige komplexere Grafikformen, wie z.B. Kreis, Rechteck und Polygon, die auch vom High Level Manager des SWF-SDK (vgl. Abschnitt 2.4) angeboten werden. Generell entspricht der Umfang der mit Saxess Wave zur Verfügung stehenden Grafik-Elemente eher der Flash Version 3, die Handhabung von Texten und Schriften ist jedoch vorbildlich. Das originale Macromedia SWF-SDK stellt für das Einbinden von Texten in Flash-Filme lediglich eine Klasse bereit, mit deren Hilfe die Umrisse der Buchstaben angegeben werden können - Unterstützung zum Auslesen der Umrisse aus lokal vorliegenden Schriftdateien existiert nicht. Saxess Wave hat den Vorteil, dass es in Java implementiert ist und die Java2D-API für diese Aufgabe nutzen kann. So ist bei Saxess Wave in einer SWFML-Datei lediglich die Schriftart, die Schriftgröße und der Text anzugeben und Java2D erstellt die Repräsentation des Textes durch Vektorgrafikobjekte.

Negativ zu erwähnen ist, dass Saxess Wave noch kaum ActionScript, keine Einbindung von Bitmap-Grafiken, keine Verlauffüllungen und keinen Sound unterstützt. So bereitet der Beispiel-Flash-Film, der in Abschnitt 2.4 mit dem High Level Manager des SWF-SDK erzeugt wurde, Saxess Wave bis auf die erwähnten Verlauffüllungen keine Schwierigkeiten und kann dank der SWFML-Schnittstelle ohne Programmierung erstellt werden. Quellcode 3.7 enthält die Formulierung des Beispiels in SWFML.

Das Wurzelelement einer SWFML-Datei heißt `SWF` und definiert globale Eigenschaften des Flash-Films: Breite (`w`) und Höhe (`h`) des Films, Hintergrundfarbe (`color`)

und Framerate (`rate`), d.h. Anzahl der angezeigten Bilder pro Sekunde. Farbangaben erfolgen in hexadezimaler Notation in der Form `"AARRGGBB"` oder dezimal in der Form `"(a,r,g,b)"`. Koordinaten bzw. Breiten- und Höhenangaben erfolgen nicht wie beim SWF-SDK in *Twips*, der Größeneinheit von Flash, die 1/20 Pixel entspricht, sondern in regulären Pixeln. Dadurch ist mit Saxess Wave eine Genauigkeit wie beim SWF-SDK nicht zu erreichen, ein weiterer Nachteil insbesondere beim Zeichnen von feinen Linien.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE SWF SYSTEM "swf.dtd">

<SWF w="300" h="300" color="FFFFFFFF" rate="3">
    <rect ID="rect1" x="0" y="0" w="100" h="100"
                     color="FFFF0000" lc="FF000000" lw="1"/>
    <rect ID="rect2" x="50" y="50" w="100" h="100"
                     color="FF0000FF" lc="FF000000" lw="1"/>
    <rect ID="rect3" x="100" y="100" w="100" h="100"
                     color="FFFFFF00" lc="FF000000" lw="1"/>

    <PlaceObject ID="rect1" depth="1"/>
    <PlaceObject ID="rect2" depth="2"/>
    <PlaceObject ID="rect3" depth="3"/>
    <ShowFrame/>

    <RemoveObject ID="rect3" depth="3"/>
    <PlaceObject ID="rect3" depth="3">
        <Matrix angle="30.0" tx="95.1" ty="-54.9"/>
    </PlaceObject>
    <ShowFrame/>

    <RemoveObject ID="rect3" depth="3"/>
    <PlaceObject ID="rect3" depth="3">
        <Matrix angle="60.0" tx="204.9" ty="-54.9"/>
    </PlaceObject>
    <ShowFrame/>
</SWF>
```

Quellcode 3.7: SWFML-Quelle des Beispiel-Films

Alle möglichen Kindelemente des Wurzelelements `SWF` sollen nicht diskutiert werden, denn sie können der SWFML-DTD entnommen werden. Im Beispiel erfolgt zuerst durch `rect`-Elemente die Definition der drei Rechtecke mit ihrer Größe, Füllfarbe (`color`), Linienfarbe (`lc`) und Liniendicke (`lw`). Jedes Element bekommt eine eindeutige ID zugewiesen, die für die Referenzierung mit `PlaceObject` und

`RemoveObject` notwendig ist. `PlaceObject` kann ein `Matrix`-Tag enthalten, das eine Verschiebung, Drehung oder Verzerrung der aktuellen Instanz eines bereits definierten Grafikobjekts erlaubt. Dabei erfolgt eine Drehung nicht wie beim SWF-SDK um den Mittelpunkt des Rechtecks, sondern immer um den Ursprung des Koordinatensystems, also um die linke obere Ecke des Films. Daraus resultiert die ebenfalls im `Matrix`-Tag angegebene Verschiebung des Grafikobjekts an seinen angestammten Platz. Mit einem `ShowFrame` wird der bis dahin definierte Inhalt eines Frames auf dem Bildschirm angezeigt.

Die „Übersetzung“ der SWFML-Quelle in einen Flash-Film erfolgt mit dem Kommando:

```
java com.saxess.visweb.swfio.Driver in.swfml out.swf
```

Vergleicht man abschließend die Dateigrößen der beiden fertigen Flash-Filme für dieses noch recht simple Beispiel (beide ohne Verlauffüllungen), so ist die Größe und die innere Tag-Struktur der Dateien identisch. Saxess Wave scheint auch ohne die Verwendung des offiziellen Macromedia SWF-SDK platzsparende Flash-Filme zu produzieren.

4 Daten und Transformationen

Wie bereits in Abschnitt 3.7 angedeutet, ist es mit Hilfe der in XML definierten Transformationssprache XSLT und einem Flash-Generator, der textbasierte SWFML-Dateien in bitcodierte Flash-Filme überführt, möglich, beliebige XML-Dateien ohne Programmierung oder die Verwendung einer grafischen Entwicklungsumgebung als Flash-Film zu visualisieren. Die eigentliche Präsentation der Daten spezifizieren XSLT-Stylesheets, deren Erstellung natürlich zeitaufwendig bleibt. Dennoch entfallen sämtliche Compilerläufe, die bei der direkten Verwendung des SWF-SDK für jede Layoutänderung notwendig wären. Außerdem steht leistungsfähige Software zum Lesen, Schreiben und zur Syntaxprüfung der verwendeten Daten kostenlos zur Verfügung.

Der genannte Entwurf geht davon aus, dass die Eingabedaten bereits in einem XML-Format vorliegen - das trifft jedoch (noch) sehr selten zu. Im Fall dieser Diplomarbeit sollen Wettervorhersagedaten grafisch dargestellt werden, die komprimiert in einem binären Dateiformat namens *GRIB* vorliegen, das zudem nur *Rasterdaten* enthält. Rasterdaten geben die Werte einer skalaren Größe an den Kreuzungspunkten eines regelmäßigen Gitternetzes an. Bei Wettervorhersagedaten kann dies z.B. die Temperatur oder der Luftdruck an der betreffenden Stelle sein. Die Kartendaten, die die Präsentation der Wetterdaten ergänzen sollen, verhalten sich ähnlich: Sie liegen als Vektorgrafiken vor, stehen jedoch nur im binären *Shapefile*-Format des Geoinformationssystems ARC/INFO zur Verfügung.

Die erste Aufgabe wird deshalb die Überführung der binären Wetter- und Kartendaten in eine XML-Repräsentation sein. Erst dann kann der angestrebte Mechanismus aus XML-Techniken und Flash-Generator (siehe Abbildung 3.6) eingesetzt werden.

Um die wahren Vorteile des Vektorgrafikformats Flash (vgl. Abschnitt 2.1) nutzen zu können, ist es unerlässlich, die Wettervorhersagedaten vom Rasterformat in Vektorgrafikobjekte zu konvertieren. Dieser Vorgang wird *Vektorisierung* genannt und fasst Punkte mit gleichem Skalarwert zu sogenannten *Isolinien* zusammen.

Weiterhin soll geklärt werden, welche Schritte erfolgen müssen, um raum- und zeitbezogene Daten für die Visualisierung vorzubereiten. Den zeitlichen Ablauf unterstützt Flash durch framebasierte Animationen. Die räumliche Komponente kann jedoch nur dargestellt werden, sofern die Koordinaten der Datenwerte in Pixel-Angaben (bzw. in Twips) umgerechnet werden. Die benötigten Operationen der 2D-Computergrafik sind *elementare Transformationen* und *Clipping*. Da außerdem weder die vorliegenden Karten- noch die Wetterprognosedaten auf der für Deutsch-

land typischen *Lambert-Projektion* basieren, ist voher noch eine Änderung der Projektion erforderlich.

Den Abschluss dieses Kapitels bilden einige Gedanken zur Gestaltung der Ausgabe, d.h. zum Layout der Flash-Filme.

4.1 Eingabedaten

Ein Entwurf zur Visualisierung von raum- und zeitbezogenen Daten kann noch so allgemeingültig formuliert sein, eine Unterstützung beliebiger Dateiformate ist nicht realisierbar - daher die Verwendung von XML. Das hat zur Folge, dass Nicht-XML-Formate im ersten Schritt in eine XML-Sprache zu konvertieren sind - ein Vorgang, der individuell für jedes Eingabedatenformat erfolgen muss. Beispielhaft werden im Rahmen dieser Diplomarbeit zwei binäre Dateiformate verwendet.

4.1.1 Landkarten

Die Grundlage jeder Wettervorhersage bilden Umrisse von Ländern, Flussverläufe, Seen und Städte mit ihren Namen. Die Ortsangabe „Osnabrück“ ist vertrauter und kann leichter eingeordnet werden, als eine Koordinatenangabe der Form „52,27° nördlicher Breite, 8,07° östlicher Länge“. Sowohl Landkarten als auch Wettervorhersagedaten definieren Ortsangaben über ein Koordinatensystem, das aber für den Betrachter uninteressant ist, denn er orientiert sich an den bekannten Elementen der Landkarte: „Wie warm wird es morgen in Osnabrück?“

Dank des Systemwissenschaftlichen Instituts der Universität besteht Zugriff auf verschiedene Dateien, die die benötigten geografischen Gegebenheiten für Deutschland und Europa in der gewünschten Auflösung enthalten. Umrisse von Ländern und Seen sind als Polygone kodiert, Flüsse sind als Linienzüge abgelegt und Städte als einzelne Punkte. Das zugrundeliegende Koordinatensystem ist durchweg das geografische, das den Erdball in 360 Längengrade und 180 Breitengrade einteilt. östl. Länge

Die Dateien werden normalerweise mit dem Geoinformationssystem ArcView[12] bearbeitet und sind sogenannte *Shapefiles*. Das Shapefile-Format besteht aus drei Dateien [ESRI1998]: Einer Hauptdatei (.shp), einer Indexdatei (.shx) und einer dBASE-Datei (.dbf). Die Hauptdatei enthält in Datensätzen variabler Länge die Beschreibung von Vektorgrafikobjekten. Unterstützt werden Punkte, Linien und

[12] Teil der Software ARC/INFO von ESRI, siehe www.esri.com

Polygone. Die Einträge der Indexdatei verweisen auf die Struktur der Hauptdatei. Die dritte Datei, die dBASE-Datenbankdatei, enthält Datensätze mit Attributen zu den Grafikobjekten der Hauptdatei. Zwischen Attribut-Datensätzen und den Geometrie-Datensätzen besteht eine 1:1-Beziehung und Grafikobjekte und Attribute werden in beiden Dateien in derselben Reihenfolge definiert, wie in Tabelle 4.1 skizziert.

ID	city.shp	city.dbf
1	type=1 (point) x=9.9748 y=53.5358	label=“Hamburg“ category=1
2	type=1 (point) x=10.1312 y=54.2962	label=“Kiel“ category=2
3	type=1 (point) x=8.0727 y=52.2697	label=“Osnabrück“ category=4
4	...	...

Tabelle 4.1: Beispiel für eine Shapefile mit Städten

Das Auslesen der Information aus den Shapefile-Dateien und die Kombination der geometrischen Formen mit ihren jeweiligen Attributen erleichert die kostenlose C-Bibliothek *shapefile* [Warm2000]. Sie stellt zwei Programmierschnittstellen (APIs) zur Verfügung, die die Erstellung einfacher C-Programme zum Lesen und Schreiben von ESRI Shapefiles unterstützen: Die *SHP-API* und die *DBF-API*. Ein Konverter, der Shapefiles in eine XML-Repräsentation übersetzt, lässt sich so recht einfach programmieren. Eine Plattformabhängigkeit durch dieses C-Programm ist kaum gegeben, da eine Erstellung von Landkartendaten im XML-Format nur ein einziges Mal erfolgen muss (im Gegensatz zur Konvertierung der binären Wettervorhersagedaten).

4.1.2 Wetterdaten

Die Wettervorhersage für Deutschland berechnet seit 45 Jahren der *Deutsche Wetterdienst* (DWD), eine staatliche Behörde [DWD2000]. Erst seit kurzer Zeit gibt es auch einige private Anbieter, die ihre Analysen aber wie der DWD ebenfalls nicht gebührenfrei abgeben. Kostenlos können aktuelle Vorhersagen z.B. aus den USA von einem Internetrechner des National Weather Service [NWS2000] heruntergeladen werden. Jedoch umfassen diese Wetterprognosedaten in einem Raster von 360*180 Punkten (entsprechend der Längen- und Breitengrade) die gesamte Erde, was für Deutschland etwa einem Rasterabstand von horizontal 70km und vertikal 110km entspricht.

Das *Lokal-Modell* des DWD berechnet die Wettervorhersage der nächsten zwei Tage für das Gebiet Mittel- und Westeuropa. Das Ergebnis umfasst Prognosewerte im Stundenabstand für ein hochauflösendes Raster von nur 7km Schrittweite

mit 325*325 Prognosewerten je Stunde je Ausprägung (Temperatur, Niederschlag, Luftdruck, ...) [DWD1999]. Zu Testzwecken wurde für diese Diplomarbeit die Wettervorhersage für den 26.12.1999 angeschafft, ein Tag, an dem ein Orkantief das Wetter in Deutschland sehr wechselhaft gestaltete und der damit sehr unterschiedliche Wetterlagen widerspiegelt. Bei den Daten handelt es sich um vier verschiedene Datensätze für jede volle Stunde des Tages, die die prognostizierten Werte für Temperatur, Niederschlag, Bewölkung und Luftdruck enthalten. Jeder Datensatz ist in einer separaten Datei im GRIB-Format abgelegt.

GRIB (Gridded Binary) ist ein universelles, bitorientiertes Dateiformat für Rasterdaten, das Wetterdienste zur Speicherung und zum Austausch ihrer Daten nutzen. Es wurde von der World Meteorological Organization (WMO) als offener internationaler Standard definiert [WMO1998]. Die aktuelle Version ist seit 1990 die GRIB Edition 1. Der Vorteil des GRIB-Dateiformats ist, dass die Dateien im Durchschnitt um mehr als 50% kleiner als reguläre binäre Dateien sind.

Eine GRIB-Datei kann aus einem oder mehreren Datensätzen (Records) bestehen, die jeweils die Werte einer einzelnen Ausprägung bzw. eines einzelnen Parameters für die Punkte eines rechteckigen Rasters enthalten. Ein Record gliedert sich in sechs logische Sektionen (Sections), von denen zwei optional sind:

- **Indicator Section (IS)**
  Startet einen neuen Record und gibt Auskunft über die Recordlänge und die verwendete GRIB Edition.
- **Product Definition Section (PDS)**
  Enthält Angaben zur Herkunft der Daten, zum Typ des Parameters und zum Ort und Zeitpunkt der Messung oder der Prognose (z.B. Temperatur in 2m Höhe über dem Boden). Informiert über die Existenz von GDS und BMS.
- **Grid Description Section (GDS) - optional**
  Definiert die Lage und Größe des Rasters, zu dem die Daten gehören. Optional, da verschiedene vordefinerte Raster bereits über eine Einstellung in der PDS angeben werden können.
- **Bit Map Section (BMS) - optional**
  Die Bits dieses Bitfeldes geben für alle Rasterpunkte Auskunft über die Präsenz eines Parameterwertes in der BDS. Optional, da nicht benötigt, falls alle Werte vorhanden oder z.B. Lückenfüller in der BDS verwendet werden.

- **Binary Data Section (BDS)**
  Enhält die gepackten Wetterdaten, einen Skalierungsfaktor und einen Referenzwert.

- **'7777' (ASCII Characters)**
  Kennzeichnet das Recordende.

Die detaillierte Spezifikation des GRIB-Formats ist vom WMO aus dem Internet zu beziehen [WMO1998].

Verschiedene Decodierer ermöglichen das Lesen von GRIB-Dateien. Die umfangreichste Unterstützung von Parametertabellen und Rasterdefinitionen bietet das C-Programm *wgrib [NCEP2000]*, ein Kommandozeilen-Programm, das das Inhaltsverzeichnis einer GRIB-Datei auflisten und wahlweise die Rasterdaten im ASCII-Format in Dateien speichern kann. Das Inhaltsverzeichnis der GRIB-Datei mit der Niederschlags-Vorhersage des DWD für den 26.12.1999, 15 Uhr lautet z.B.:

```
rec 1:0:date 1999122600 APCP kpds5=61 kpds6=1 kpds7=0 levels=(0,0)
    grid=255 sfc 0-15hr acc: bitmap: 975 undef
APCP=Total precipitation [kg/m^2]
timerange 4 P1 0 P2 15 TimeU 1  nx 325 ny 325
GDS grid 10 num_in_ave 0 missing 0
center 78 subcenter 255 process 132 Table 2
rotated LatLon grid
  lat 3.250000 to -17.000000  lon -12.500000 to 7.750000
  nxny 105625  (325 x 325)  dx 62 dy 63  scan 0  mode 128
  transform: south pole lat -32.500000  lon 10.000000
             rot angle 0.000000
min/max data 0 218  num bits 8  BDS_Ref 0  DecScale 0 BinScale 0
```

Da der C-Quellcode von wgrib verfügbar ist, werden im Rahmen der Realisierung (Abschnitt 5.3.2) die benötigten Passagen des Programms nach Java portiert und so umgeschrieben, dass eine direkte programminterne Weiterverarbeitung der Daten möglich ist .

4.2 Vektorisierung von Rasterdaten

Die Vektorisierung von Rasterdaten ist ein spezielles Gebiet der Computergrafik, das sich mit der Umwandlung von Rasterdaten in Vektordaten beschäftigt. Die häufigste Anwendung ist sicher die Konvertierung von Pixelgrafiken in Vektorgrafiken, für die inzwischen jedes umfangreichere Grafikprogramm ein Modul

bereitstellt (CorelTrace von Corel, ArcScan für ARC/INFO, CAD Overlay für AutoCAD). Je leistungsfähiger die Vektorisierungssoftware, desto besser erkennt sie zusammenhängende Gebiete, Linien und Kreisbögen und erstellt daraus Vektorobjekte wie Polygone, Linien usw. Dieser Vorgang ist meistens sehr rechenintensiv und entsprechende Software kostet bis zu mehreren zehntausend Mark.

Besonders interessant ist der Einsatz von Vektorisierern für den CAD- und den GIS-Bereich, wo z.B. eingescannte topografische Karten oder CAD-Zeichnungen in den vektorbasierten Datenbestand übernommen werden sollen. Doch auch für die hier gestellte Aufgabe - die Visualisierung der Wettervorhersagedaten des DWD mit Macromedia Flash - ist es unverzichtbar, die Rasterdaten aus den GRIB-Dateien in Vektorgrafikobjekte umzuwandeln. Die Vorteile von Vektorgrafiken kamen bereits im Abschnitt 2.1 zur Sprache: Sie sind auflösungsunabhängig, zoomfähig und platzsparend.

GRIB-Dateien enthalten die Werte eines Parameters für ein reguläres rechteckiges Raster bzw. Gitter. Die Parameterwerte können deshalb für die weitere Verarbeitung in ein zweidimensionales Array eingelesen werden. Die Indezes der Rasterzellen wachsen dabei zeilenweise von oben nach unten und spaltenweise von links nach rechts. Die eigentlichen Positionen der Rasterpunkte auf der Erdoberfläche sind vorerst nicht von Interesse.

Jeder Rasterpunkt hat genau vier direkte Nachbarn, sofern er nicht am Rand liegt. Zwischen zwei benachbarten Rasterpunkten, d.h. auf einer *Raster-* oder *Gitterkante*, wird davon ausgegangen, dass sich der Wert des Parameters linear verhält: Der Wert steigt, fällt oder bleibt gleich. Zwischenwerte auf einer Gitterkante können durch *lineare Interpolation* berechnet werden, wie Abbildung 4.1 darstellt.

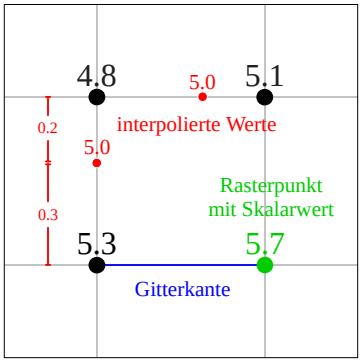


Abbildung 4.1: Lineare Interpolation auf Gitterkanten

Die Aufgabe eines Vektorisierungsalgorithmus ist es, anhand vorgegebener Werte die entsprechenden Isolinien in diesem Raster zu finden. *Isolinien* sind Linienzüge,

auf denen der Parameter einen konstanten Wert besitzt. Man erhält sie, indem benachbarte Punkte mit gleichem Parameterwert - Rasterpunkte oder linear interpolierte Punkte auf Gitterkanten - miteinander verbunden werden.

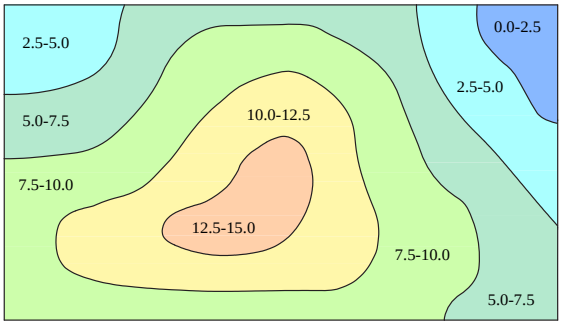


Abbildung 4.2: Temperatur-Isoflächen (in °C)

Isolinien kreuzen sich niemals, wie man sich an einem Beispiel wie in Abbildung 4.2 leicht klar machen kann oder auch von Höhenlinien auf Landkarten kennt. Sie werden eventuell komplett von anderen Isolinien umrundet, schneiden diese jedoch nicht. Die Fläche zwischen zwei Isolinien soll als *Isofläche* bezeichnet werden, sie enthält die Parameterwerte eines bestimmten *Wertebereichs*. Die Isoflächen sind die eigentlichen Elemente, die ein Vektorisierer im vorliegenden Fall als Resultat liefern soll. Repräsentiert als Polygone können die Wettervorhersagedaten eingefärbt und eventuell beschriftet in Flash visualisiert werden: Z.B. Temperatur zwischen 2,5°C und 5,0°C in hellblau.

Neben kostspieligen kommerziellen Applikationen zur Vektorisierung von Rasterdaten gibt es im Internet einige kostenlose Ansätze [Bour1987][Libe2000][Webe2000], die jedoch alle nicht den gewünschten Anforderungen entsprechen. Vielfach waren die Algorithmen auf Pixelgrafiken abgestimmt und verschmolzen lediglich gleichwertige Pixel miteinander. Für direkt benachbarte Bildpunkte liefert dieses Vorgehen zufriedenstellende Ergebnisse, nicht jedoch für ein Raster mit 7km Schrittweite. Erst mittels Interpolation auf den Gitterkanten erhält man die geforderten Isoflächen bzw. Isolinienzüge und nicht nur auf Rasterpunkten basierende Linien oder Flächen. Schwierigkeiten macht vielen Algorithmen vor allem auch das Einfärben von ineinander liegenden Isoflächen.

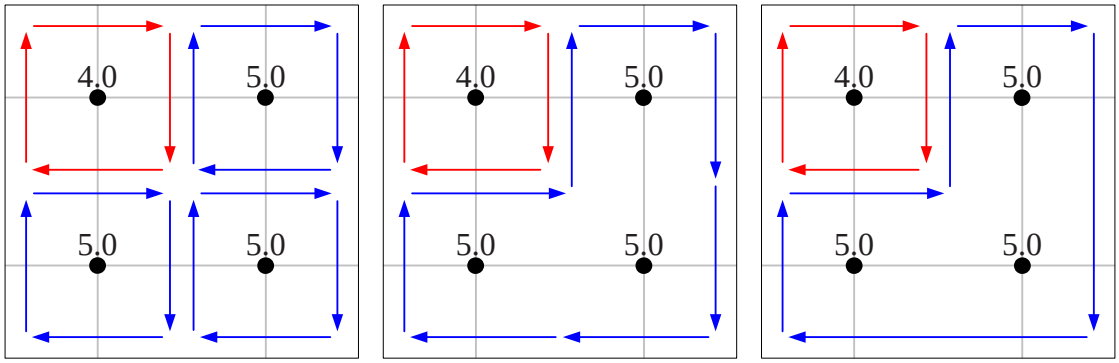
Im Folgenden sollen zwei Algorithmen zur Vektorisierung von Rasterdaten vorgestellt werden, die beide vom vorliegenden Sonderfall eines zweidimensionalen, vollständig besetzten rechteckigen Rasters ausgehen. Ein einfacher Algorithmus wurde bei einem ersten Ansatz verfolgt, musste aber anschließend verworfen werden, da er keine lineare Interpolation verwendet. Der zweite Algorithmus basiert auf einem Artikel der Association for Computing Machinery (ACM), bestimmt die

Isolinienpunkte durch lineare Interpolation, muss aber für die vorliegende Aufgabe noch erweitert werden, um das korrekte Einfärben der Isoflächen zu ermöglichen.

4.2.1 Einfacher Algorithmus

Als „lauwarm“ bezeichnete der unbekannte Autor seinen Algorithmus [Card1999], verfolgt aber einen interessanten Ansatz. Die grundlegende Idee ist die Darstellung jedes Rasterpunktes als vektorielles Rechteck, d.h. jeder Punkt wird im ersten Schritt von vier geradlinigen Vektoren im Uhrzeigersinn umlaufen. Jeder Vektor kennt dabei seinen Vorgänger und Nachfolger (siehe Abbildung 4.3a).

Alle Vektoren (vier je Rasterpunkt) werden dem Parameterwert am Rasterpunkt entsprechend in eine Liste eingetragen, so dass es für jeden möglichen Wert des Parameters eine Liste von Vektoren gibt. Alternativ können die Vektoren auch nach Wertebereichen des Parameters einsortiert werden. In Abbildung 4.3 sind die zum Wert 5.0 gehörigen Vektoren blau und die des Rasterpunktes mit Wert 4.0 rot dargestellt.



(a) Ausgangssituation (b) Eliminierung (c) Optimierung

Abbildung 4.3: Beispiel für einen einfachen Vektorisierungs-Algorithmus

Im nächsten Schritt werden je Vektorenliste alle Vektoren untereinander verglichen. Beschreiben zwei Vektoren dieselbe Strecke in entgegengesetzter Richtung, stehen sie zwischen zwei Rasterpunkten, an denen der Parameterwert identisch ist (bzw. aus demselben Wertebereich stammt). Die umlaufenden Vektoren der beiden Rasterpunkte können miteinander zu einem größeren Rechteck verschmelzen, das anschließend zwei Rasterpunkte umläuft. Dies geschieht, indem die Grenzvektoren deaktiviert werden und die Verzeigerung ihrer Vorgänger bzw. Nachfolger geändert wird.

Durch die Wiederholung dieses Vorgangs verschmelzen alle benachbarten Rasterpunkte mit identischem Wert bzw. desselben Wertebereichs zu einer Fläche, die von vielen einzelnen Vektoren umlaufen wird (siehe Abbildung 4.3b). Dabei werden nur die Nachbarn entsprechend des *4-way-stepping* berücksichtigt.

Bei einer abschließenden Optimierung können aufeinanderfolgende Vektoren, die in dieselbe Richtung zeigen, als ein verlängerter Vektor geschrieben werden (siehe Abbildung 4.3c).

Der Algorithmus erzeugt Polygone, die sich aus Geraden zusammensetzen, die in nur vier Richtungen (oben, links, rechts, unten) verlaufen können. D.h. es entstehen „Klötzchen"-Polygone, die nicht viel besser als Pixelgrafiken wirken. Die fehlende Interpolation auf den Gitterkanten begründet die Ungenauigkeit des Algorithmus und führte dazu, dass er schliesslich verworfen wurde.

4.2.2 Algorithm 531 - Contour Plotting

Der Artikel „Algorithm 531 - Contour Plotting" [SnyW1978] wurde 1978 von William V. Snyder bei der Association for Computing Machinery (ACM) eingereicht. „Contour Plotting" beschreibt allgemein das Zeichnen von Umrisslinien bzw. Isolinien.

Snyder geht bei seinem Artikel von einem zweidimensionalen Array mit skalaren Werten und von vorgegebenen Isowerten aus. Er beschreibt zwei mögliche Vorgehensweisen, wie die gesuchten Isolinien zu finden sind. Prinzipiell gibt es die beiden folgenden Ansätze:

- Isolinien können von einem Startpunkt aus durch das gesamte Raster verfolgt werden, bis sie einen geschlossenen Linienzug bilden oder eine Begrenzung schneiden.
- Der Reihe nach wird innerhalb jeder Gitterzelle (eingerahmt von vier Arrayelementen) nach Isolinien gesucht und alle gefundenen Teilstücke werden direkt ausgegeben.

Snyder entschied sich für den ersten Ansatz, um die Anzahl der Plot-Kommandos zu minimieren. Durch die Wahl des zweiten Ansatzes wäre sein Algorithmus für diese Diplomarbeit uninteressant gewesen, denn nur aus geschlossenen Linienzügen lassen sich später Isoflächen erstellen.

Sein *line-following*-Algorithmus lässt sich in die folgenden Schritte einteilen:

1. Gegeben ist ein zweidimensionales Raster in Form eines Arrays und die Liste der gesuchten Isowerte.

2. Das Raster wird zeilenweise durchlaufen und die Kanten jeder Rasterzelle werden auf Schnittpunkte mit Isolinien untersucht. Ein solcher Schnittpunkt existiert, wenn ein gesuchter Isowert zwischen den Parameterwerten der beiden beteiligten Rasterpunkte liegt.

3. Wurde eine solche Gitterkante gefunden, wird der Schnittpunkt $P_{schnitt}$ der Isolinie mit der Kante durch lineare Interpolation des Parameterwertes w zwischen den Start- und Endpunkten der Kante bestimmt. Für die x-Koordinate lautet die Berechnung (y entsprechend):

$$x_{schnitt} = x_{start} + (x_{end} - x_{start}) \cdot \frac{w_{schnitt} - w_{start}}{w_{end} - w_{start}}$$

   Zusätzlich wird vermerkt, dass der Isowert auf der aktuellen Gitterkante erkannt wurde, damit dieser Schritt nicht wiederholt wird.

4. Das Programm durchsucht nun die an die aktuelle Gitterkante (die Kante, auf der der Schnittpunkt mit einer Isolinie gefunden wurde) angrenzende Gitterzelle nach derselben Eigenschaft, einem Schnittpunkt mit der aktuellen Isolinie. Ist diese Kante gefunden, erfolgt die Bestimmung des Schnittpunkts wie unter 3 und die beiden Schnittpunkte werden mit einer geraden Linie verbunden (siehe Abbildung 4.4).
   Schritt 4 wird wiederholt, bis schließlich keine entsprechenden Gitterkanten mehr gefunden werden, die noch nicht markiert wurden.

5. Die Schritte 2, 3 und 4 werden solange wiederholt, bis keine weitere Gitterkante gefunden wird, deren Schnittpunkte mit allen gesuchten Isolinien nicht bereits vermerkt ist.

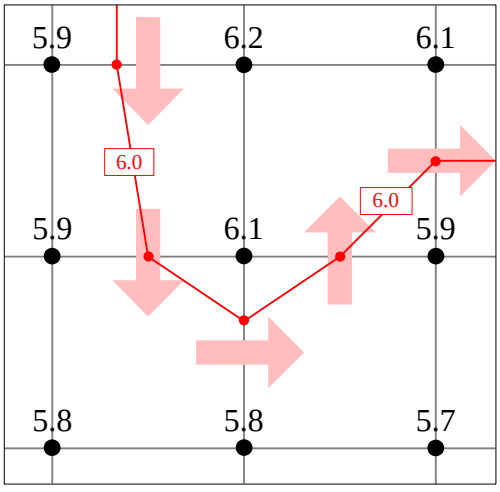


Abbildung 4.4: line-following-Algorithmus

Der Algorithmus liefert durch die Interpolation im Vergleich zum einfachen Algorithmus (Abschnitt 4.2.1) viel bessere Ergebnisse. Zu beantworten ist noch die Frage, wie die Flächen zwischen den Isolinien gefüllt werden können.

Der Algorithmus von Snyder gibt die gefundenen Isolinien direkt auf einen Drucker oder Bildschirm aus und erstellt keine programminterne Repräsentation der Linien. Geschlossene Linienverläufe entstehen nur für Isolinien, die nicht an den Rand des Rasters stoßen. Außerdem erhält man keine Auskunft über die Bereiche zwischen den Isolinien und die daraus resultierende Füllfarbe der Flächen. Das Fazit lautet: Snyders Algorithmus erstellt lediglich Isolinien jedoch keine Isoflächen. Der Algorithmus kann als Bestätigung dienen, dass eine Aufsplittung der Gitterzellen in Dreiecke - so lauteten erste Überlegungen zu diesem Thema - nicht notwendig ist. Jedoch muss ein eigener Algorithmus anhand der genannten Ideen entwickelt werden.

4.2.3 Erweiterung von Snyders Algorithmus

Um aus Rasterdaten Isoflächen zu erzeugen, müssen die Isolinien in programminternen Datenstrukturen abgelegt werden und weiterhin zur Verfügung stehen. Denn zu bedenken ist, dass jede Isolinie zwei Isoflächen voneinander abgrenzt. Jede Isolinie gehört zur Umrisslinie von *zwei* aneinandergrenzenden Isoflächen. Liegt eine Isofläche komplett in einer anderen, so muss in die äußere ein Loch geschnitten werden, so dass ein „Donut“ entsteht. Ein solcher Ring lässt sich mit einem Polygon, d.h. mit einem einzigen Linienzug, nur durch eine unsichtbare Verbindung zwischen äußerer und innerer Begrenzung erreichen (siehe Abbildung 4.5). Flash bietet diese Möglichkeit.

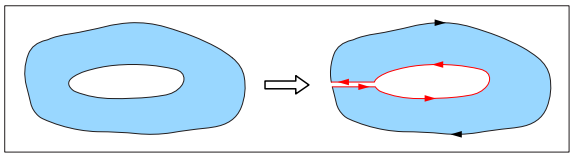


Abbildung 4.5: Der „Donut“-Effekt

Isoflächen ohne Loch, die keinen Kontakt mit dem Rand haben, die also von einer anderen Isofläche umgeben sind, werden von einer geschlossenen Isolinie begrenzt. Isoflächen, die dagegen teilweise vom Rand des Rasters begrenzt werden, besitzen keine Umrisslinie, die nur aus den Punkten eines Isowertes besteht. Ihre äußere Begrenzungslinie setzt sich abwechselnd aus Randstücken und Isolinien zusammen, wobei die Isolinien beliebig zu beiden Isowerten gehören können, die den Wertebereich der Isofläche begrenzen. Ein Beispiel: Die dunkelgrüne Isofläche

aus Abbildung 4.2 auf Seite 56 spiegelt den Wertebereich 5.0-7.5°C wider. Sie wird begrenzt von den Isowerten 5.0°C und 7.5°C. Ihre Begrenzungslinie setzt sich aus drei Randstücken und zwei Isolinien mit dem Wert 5.0°C und einer Isolinie mit dem Wert 7.5°C zusammen.

Ein Algorithmus, der diese Gegebenheiten berücksichtigt und als Ergebnis Isoflächen bzw. Polygone sortiert nach ihrem Wertebereich liefert, lautet:

1. *Vorgaben:* Gegeben sind wie bei Snyders Algorithmus ein zweidimensionales, vollständig besetztes Array mit skalaren Werten (das Raster) und die Liste der gesuchten Isowerte.

2. *Voraussetzung:* Eine Voraussetzung soll sein, dass kein Rasterpunkt selbst den Wert eines gesuchten Isowertes hat, d.h. die Isowerte befinden sich ausschließlich auf den Gitterkanten. Hat ein Rasterpunkt zu Beginn den Wert eines Isowertes, so ist dieser minimal um ein angemessenes δ zu verändern.
   Diese Einschränkung ist notwendig, damit das *line following* (Punkt 6) Isolinien stets anhand ihrer Schnittpunkte mit Gitterkanten verfolgen kann und nicht auf einem Rasterpunkt stoppt, weil von dort keine eindeutige Nachbarzelle existiert.

3. *Vorbereitung:* Die Liste der gesuchten Isowerte wird sortiert und um $-\infty$ und $+\infty$ ergänzt. So können die Bereiche zwischen den genannten Isowerten als die gesuchten Wertebereiche der Isoflächen genutzt werden, denn auch alle Werte unterhalb des kleinsten bzw. oberhalb des größten vorgegeben Isowerts werden $-\infty$ bzw. $+\infty$ mit einbezogen. Für jeden Wertebereich wird eine (noch leere) Liste von Polygonen angelegt, wie in Abbildung 4.6 dargestellt.

$-\infty$	...	-5.0	$\rightarrow$	Liste von Polygonen
-5.0	...	-2.5	$\rightarrow$	Liste von Polygonen
-2.5	...	± 0.0	$\rightarrow$	Liste von Polygonen
± 0.0	...	$+2.5$	$\rightarrow$	Liste von Polygonen
$+2.5$	...	$+5.0$	$\rightarrow$	Liste von Polygonen
$+5.0$	...	$+\infty$	$\rightarrow$	Liste von Polygonen

Abbildung 4.6: Isowertebereiche mit Polygonlisten

4. *Start der Suche:* Ein leeres Polygon wird erzeugt und erhält als Startpunkt die Koordinaten des Rasterpunktes der linken oberen Ecke. Anhand des Wertes dieses ersten Rasterpunktes wird der dazugehörige Wertebereich ermittelt und das aktuelle Polygon in die entsprechende Polygonliste eingefügt.

5. *Suche am Rand:* Ist das aktuelle Polygon bereits geschlossen (der Anfangspunkt entspricht dem Endpunkt und das Polygon besteht aus mehr als einem Punkt), gehe zu Schritt 7.
   Ausgehend vom Endpunkt des Polygons wird im Uhrzeigersinn die nächste Randkante des Rasters auf mögliche Schnittpunkte mit Isolinien untersucht. Dabei ist darauf zu achten, dass es je Kante Schnittpunkte mit mehreren Isolinien geben kann. (1) Existiert kein Isowert auf der aktuellen Kante, wird das aktuelle Polygon um die Koordinaten des Endpunktes der Randkante erweitert. Schritt 5 beginnt von vorne. (2) Existiert auf der aktuellen Kante ein Isowert, wird zusätzlich zum aktuellen Polygon ein neues Polygon mit den Koordinaten des Schnittpunktes als Startpunkt erzeugt. Wie bei Snyder werden die Koordinaten der Isopunkte durch lineare Interpolation ermittelt. Das neue Polygon wird in die entsprechende Polygonliste eingefügt und für die spätere Bearbeitung auf einem Stack abgelegt. Das aktuelle Polygon wird stattdessen vorerst weiterbearbeitet und um die Koordinaten des Isowertes auf der aktuellen Kante erweitert (siehe Abbildung 4.7). Die Suche entlang der Isolinie startet (→ Punkt 6).

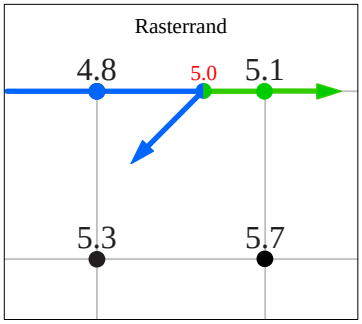


Abbildung 4.7: Suche am Rand

6. *line following:* Die drei restlichen Kanten der aktuellen Gitterzelle werden nach dem aktuellen Isowert abgesucht. Mindestens eine weitere Kante muss einen Schnittpunkt mit der aktuellen Isolinie besitzen (drei weitere Schnittpunkte ist ein Spezialfall → Seite 63), wie man sich anhand von Abbildung 4.4 auf Seite 59 klarmachen kann. Wurde der neue Schnittpunkt noch nicht besucht, kann das aktuelle Polygon um ihn erweitert und der Schnittpunkt als erkannt markiert werden. Die Markierung erfolgt dabei in Abhängigkeit vom aktuellen Isowert und dem zugehörigen Wertebereich (ist der Isowert die obere oder die untere Grenze?). So kann für beide angrenzenden Isoflächen die Isolinie entlanggelaufen werden, ohne auf bereits markierte Kanten zu treffen.
   Sofern die Schnittpunktkante keine Randkante ist, springe in die benachbarte

Zelle und wiederhole Schritt 6. An einer Randkante angelangt folgt Schritt 5. Ist keine Forsetzung der Isolinie möglich, da bereits alle Kanten markiert sind, ist das (innere) Polygon geschlossen, es folgt Schritt 9.

7. *Abarbeitung des Polygon-Stacks:* Ist der Stack leer, sind alle Polygone mit Kontakt zum Rand des Rasters gefunden - gehe zu Schritt 8. Ansonsten wird das oberste Polygon des Stacks zum aktuellen und es folgt Schritt 5.

8. *Suche nach inneren Polygonen*: Um alle Polygone zu finden, die keinen Kontakt mit dem Rand des Rasters haben, werden alle Gitterzellen zeilenweise durchlaufen. Es reicht, die jeweils *untere* Kante einer Zelle auf noch nicht markierte Schnittpunkte mit Isolinien zu untersuchen. Auch hier können je Kante mehrere Schnittpunkte mit verschiedenen Isolinien auftreten. Sobald ein neuer Schnittpunkt gefunden wird, verfolgt Schritt 6 die Isolinie mit einem neuen Polygon. das ebenfalls anhand seines zugehörigen Wertebereichs in die Polygonliste eingestellt wird. Schritt 9 folgt anschließend.
   Sind alle Gitterzellen durchlaufen, folgt Schritt 10.

9. *Ausschneiden der Löcher*: Das neu gefundene innere Polygon schneidet ein Loch in sein umgebenes Polygon. Zu diesem Zweck muss das äußere Polygon um eine Verbindungslinie und die Umrisslinie des inneren Polygons erweitert werden (siehe Abbildung 4.8). Als Einstiegspunkt dieser Erweiterung dient der bei Schritt 8 zuletzt gefundene bereits markierte Schnittpunkt einer Isolinie mit einer Rasterkante (falls in der aktuellen Zeile schon ein solcher Punkt gefunden wurde, sonst der linke Rand). Zwischen diesem Einstiegspunkt und dem aktuellen inneren Polygon kann keine weitere Isolinie existieren.
   Die weitere Abarbeitung der Schleife von Schritt 8 folgt.

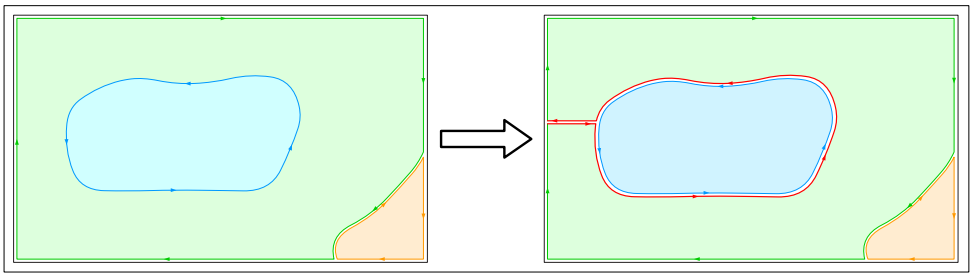


Abbildung 4.8: Ausschneiden der inneren Polygone

10. *Löschen unnötiger Punkte*: Anhand eines geeigneten Algorithmus können alle Polygonpunkte gelöscht werden, die kollinear sind (z.B. entlang der Randpunkte).

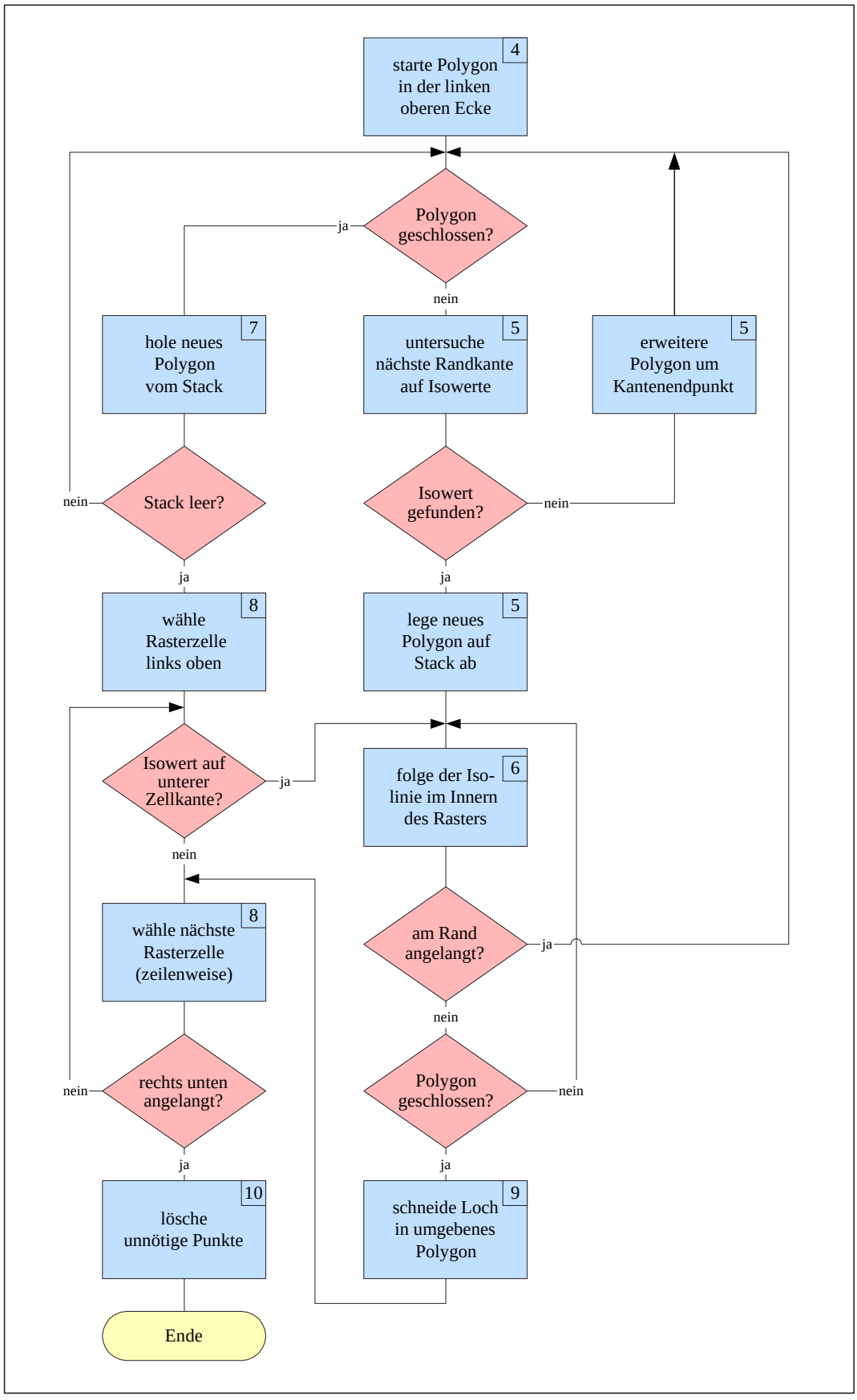


Abbildung 4.9: Erweiterter Snyder-Algorithmus

Der Algorithmus liefert in Listen von Polygonen die Isoflächen getrennt nach ihren Wertebereichen und damit ihren Füllfarben. Snyders ursprünglicher Algorithmus ist kaum wiederzuerkennen. Abbildung 4.9 auf Seite 64 zeigt in einem Flussdiagramm noch einmal die Abfolge der einzelnen Schritte.

Spezialfall: Isolinie schneidet alle vier Kanten einer Gitterzelle

Schneidet eine Isolinie alle vier Kanten einer Gitterzelle, ist der Verlauf der Isolinie nicht eindeutig, da sich derselbe Isowert auf allen vier Kanten befindet. Anhand der Position der Schnittpunkte auf der oberen und der unteren Kante wird in diesem Fall entschieden, welche Verbindungen gewählt werden: Ist der x-Wert auf der oberen Zellkante kleiner als auf der unteren Zellkante, erfolgt die Verbindung unten-rechts und oben-links. Abbildung 4.10 zeigt den Sachverhalt anhand derselben Gitterzelle für zwei unterschiedliche Parameterwerte (z.B. Temperatur) in der linken oberen Ecke.

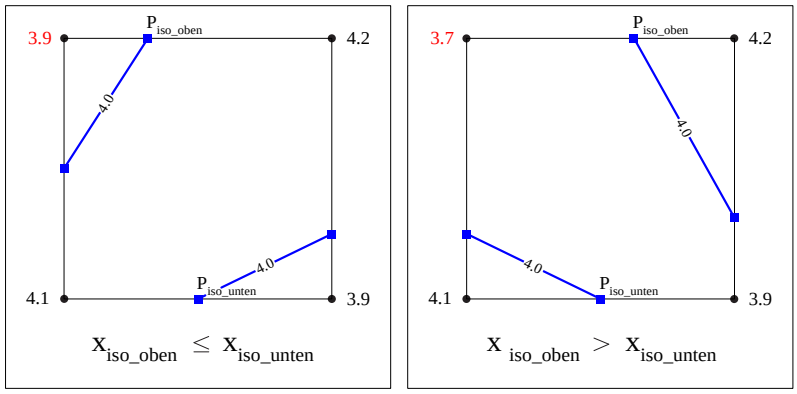


Abbildung 4.10: Spezialfall beim *line following*

4.3 Kartenprojektionen

Da die Erde von sehr komplexer Form ist, gehört es zu den Hauptaufgaben der *Geodäsie* (Vermessungskunde), die Figur unseres Planeten zu bestimmen. Zur Abstraktion und Vereinfachung der Erdoberfläche werden *geodätische Bezugssysteme*, wie z.B. die mathematischen Bezugsflächen *Kugel* und *Rotationsellipsoid*, eingeführt. Um die Position in einem solchen Bezugssystem zu beschreiben, verwendet man *Koordinaten*, die einem *Koordinatensystem* zugeordnet sind [Vose1998].

Auch in den zur Verfügung stehenden Shapefiles und den GRIB-Dateien erfolgen Positionsangaben über zweidimensionale Koordinaten. Denn legt man eine Kugelgestalt der Erde zugrunde, können alle Punkte der Erdoberfläche durch zwei Parameter, die sogenannte *geografische Länge u* und die *geografische Breite v*, eindeutig

(bis auf die Pole, denen keine eindeutige geografische Länge zugeordnet werden kann) festgelegt werden. Eine Parameterdarstellung der Erdkugeloberfläche in diesen beiden Parametern hat die Form

$$x = R\cos u\cos v, \quad y = R\sin u\cos v, \quad z = R\sin v$$

mit R als Radius der Kugel (siehe Abbildung 4.11). Durch $v = 0$ wird der Äquator beschrieben, $v = \pm\frac{\pi}{2}$ erfasst die Pole. Der Nullmeridian und der Äquator sind die Referenzebenen zur Bestimmung der Längen- und Breitengrade. Die geografische Breite beschreibt den Winkel der Verbindungsachse zwischen Punkt und Erdmittelpunkt zur Äquatorebene. Die geografische Länge ist der Winkel zwischen der Nullmeridianebene und einer weiteren Ebene, die senkrecht zur Äquatorebene den zu referenzierenden Punkt und den Erdmittelpunkt schneidet.

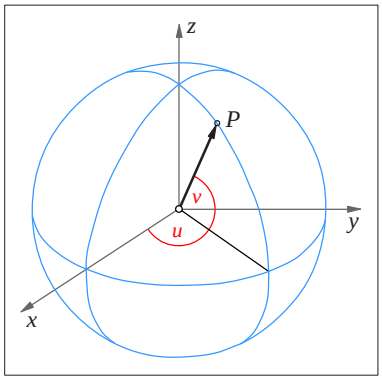
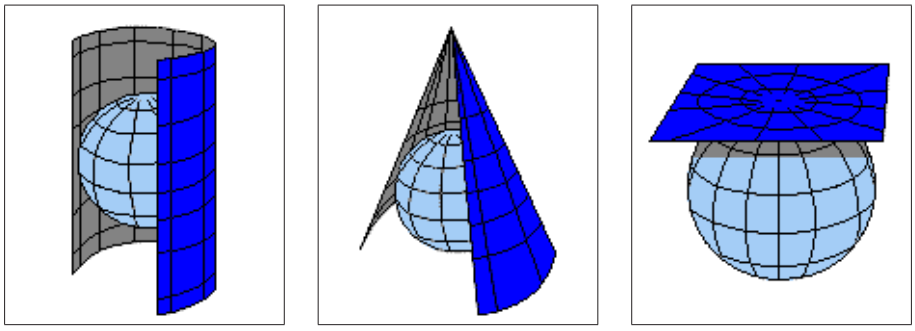


Abbildung 4.11: Parameterdarstellung einer Kugel [Hosc1984]

Werden die geografischen Koordinaten direkt als Abszissen- und Ordinatenwerte eines zweidimensionalen Koordinatensystems dargestellt, kommt es zu starken Verzerrungen in Richtung der Pole. Die *Kartografie* beschäftigt sich mit der nicht trivialen Aufgabe, durch *Kartenprojektionen* - auch *Kartenentwürfe* genannt - die Oberfläche der Erdkugel in eine Ebene abzubilden. Da es unmöglich ist, Karten zu konstruieren, die ein exaktes Abbild der Erdoberfläche darstellen [Hosc1984], werden Projektionsflächen gewählt, die das abzubildene Gebiet möglichst gut approximieren. Dabei werden Tangentialebenen, Kegel und Zylinder oder komplexe mathematische Flächen verwendet, die in eine Ebene abrollbar sind. Eine Abbildung direkt in die Ebene wird *azimutale Abbildung* oder *azimutaler Entwurf* genannt, bei Abbildungen auf einen Zylinder spricht man von einem *Zylinderentwurf*, während Abbildungen auf einen Kegel als *Kegelentwurf* oder *konische Abbildung* bezeichnet werden. Abbildung 4.12 veranschaulicht diese drei Möglichkeiten.

Aus Sicht des Anwenders einer Karte sind die folgenden Eigenschaften der ange-



(a) Zylinder (b) Kegel (c) Ebene

Abbildung 4.12: Beispiele für Projektionsflächen [Faus2000]

wandten Projektion wünschenswert. Eine mathematisch exakte Definition der Begriffe ist entsprechenden Lehrbüchern zur Differentialgeometrie zu entnehmen.

- *Winkeltreue* (auch *Konformität*): Der Winkel zwischen zwei sich schneidenden Linien bleibt bei der Abbildung erhalten. Die Meridiane und Breitenkreise schneiden sich im rechten Winkel.
- *Längentreue, Abstandstreue:* Der Abstand zweier Punkte bleibt bei der Abbildung bis auf einen festen, für alle abgebildeten Gebiete der Kugel gültigen (globalen) Maßstabsfaktor erhalten.
- *Flächentreue* (auch *Äquivalenz*): Flächentreue Projektionen erhalten die Flächengrößen (maßstabsbezogen) und dieselben proportionalen Beziehungen von Flächen wie auf der Erde.

Verzerrungen treten auf, wenn eine der oben genannten Eigenschaften nicht zutrifft. Abbildungen, die keine Verzerrungen hinsichtlich der Strecken, Winkel und Flächen aufweisen, werden *verzerrungsfrei* genannt. Bereits *Euler* (1707-1783) hat gezeigt, dass es unmöglich ist, eine Kugel verzerrungsfrei in eine Ebene abzubilden. Insbesondere existiert keine abstandstreue Abbildung der Kugel in die Ebene. Das schließt nicht aus, dass spezielle Linien in wahrer Länge abgebildet werden. Aus den Überlegungen der Differentialgeometrie folgt weiter, dass sich *Winkeltreue* und *Flächentreue* ausschließen [Sosn1999]. So gibt es jede in Abbildung 4.12 gezeigte Kartenprojektion in einer winkeltreuen, einer längentreuen und einer flächentreuen Variante.

Eine typische Projektion für die Darstellung einer Deutschlandkarte ist der flächentreue azimutale Entwurf von *Lambert* (1728-1777) mit geeigneter Wahl der Ent-

wurfsachse. Es handelt sich um eine flächen- und winkeltreue Projektion, die Perspektive wird jedoch verzerrt. Die Verzerrung ist am Zentrum (dem Berührpunkt) gleich Null , steigt aber mit der Entfernung davon an. Lamberts Entwurf sollte deshalb nicht für mehr als eine Hemisphäre angewendet werden. Abbildung 4.13 stellt die Konstruktion des Entwurfs für den Fall dar, dass die Entwurfsachse der Polachse entspricht.

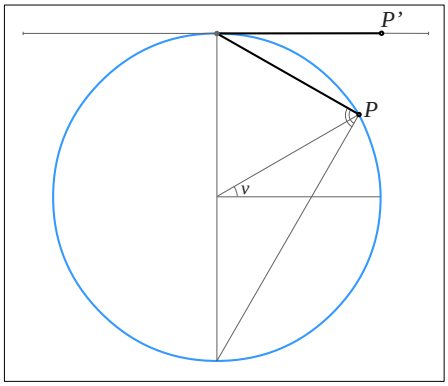


Abbildung 4.13: Flächentreuer azimutaler Entwurf von Lambert [Hosc1984]

Die Entwurfsachse läuft bei einem azimutalen Entwurf durch den Erdmittelpunkt und den Berührpunkt der Tangentialebene an die Kugel. Für eine Deutschlandkarte bietet sich z.B. die geografische Lage von Eisenach in Thüringen als Berührpunkt der Projektionsebene an, da die Stadt ungefähr in der Mitte des Landes liegt und die Verzerrungen so gleichmäßig zum Kartenrand hin zunehmen.

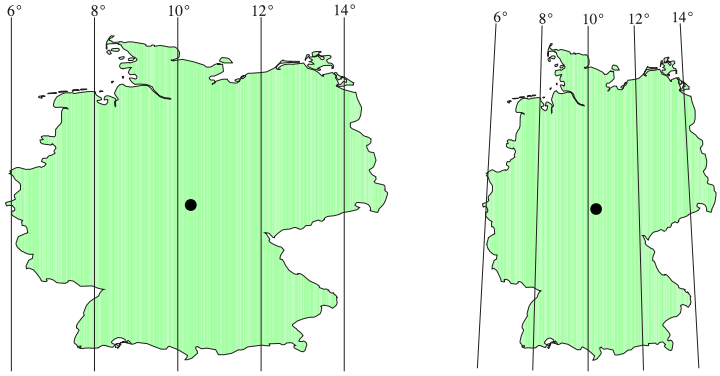
Die Projektion eines grafischen Objekts (z.B. ein Isoflächen-Polygon oder die geografische Lage einer Stadt) wird durch das Projezieren seiner einzelnen Definitionspunkte erreicht. Die Formeln zur Berechnung der Bildkoordinaten des flächentreuen azimutalen Entwurfs von Lambert bei allgemeiner Lage der Entwurfsachse lauten [SnyJ1987]:

$$
\begin{aligned}
x &= k \cdot \cos v \sin(u - u_0) \\
y &= k \cdot [\cos v_0 \sin v - \sin v_0 \cos v \cos(u - u_0)] \\
\text{mit} \quad k &= \sqrt{\frac{2}{1 + \sin v_0 \sin v + \cos v_0 \cos v \cos(u - u_0)}}
\end{aligned}
$$

Wie zu Beginn vereinbart, bezeichen u und v die geografische Länge bzw. Breite. Die Koordinaten des Berührpunkts gibt das Tupel (u_0, v_0) an. Als Bildbereich erhält man Werte zwischen -2 und 2 für beide Koordinaten.

Abbildung 4.14 stellt abschließend für eine einfache Deutschlandkarte die direkte Übertragung der geografischen Koordinaten der Erdoberfläche in die Ebene und die

durch die Lambert-Projektion erlangte Darstellung gegenüber. Der Berührpunkt der Projektionsebene ist markiert.



(a) Geografische Koordinaten (b) Lambert-Projektion

Abbildung 4.14: Deutschland in zwei verschiedenen Projektionen

4.4 Transformationen und Clipping

Zur endgültigen Darstellung der vorliegenden Landkarten- und Wetterdaten müssen weitere Operationen aus dem Bereich der Computergrafik angewendet werden. Für die Umwandlung beliebiger Koordinaten in Pixel des Flash-Films werden die *elementaren zweidimensionalen Transformationen* benötigt, zu denen *Translation*, *Skalierung* und *Rotation* zählen. Mit der Auswahl eines gewünschten Darstellungsbereichs befasst sich das *Clipping*.

4.4.1 Zweidimensionale Transformationen

Elementare Transformationen

Wie schon beim Thema Kartenprojektion angedeutet, wird die Manipulation der Darstellung bei grafischen Objekten allgemein durch mathematische Operationen auf den Definitionspunkten der Objekte beschrieben. Transformationen lassen sich deshalb durch Operationen auf den einzelnen Definitionspunkten beschreiben. Die *elementaren zweidimensionalen Transformationen* lauten nach Fellner [Fell1992]:

Unter *Translation* versteht man eine geradlinige Verschiebung eines Objekts bzw. seiner Definitionspunkte. Die Koordinaten des verschobenen Punktes $P' = (x', y')$

berechnen sich aus der ursprünglichen Position $P = (x, y)$ und einem Translationsvektor $\vec{t} = (t_x, t_y)$ durch

$$x' = x + t_x \quad \text{und} \quad y' = y + t_y$$

Mit *Skalierung* (Scaling, Zooming) wird die Vergrößerung bzw. Verkleinerung von Grafikobjekten bezeichnet. Die transformierten Koordinaten $P' = (x', y')$ des Punktes $P = (x, y)$ werden mit den Skalierungsfaktoren (s_x, s_y) anhand der Formel

$$x' = s_x \cdot x \quad \text{und} \quad y' = s_y \cdot y$$

berechnet. Sind die Werte s_x und s_y gleich, erfolgt eine proportionalitätserhaltende Vergrößerung bzw. Verkleinerung, andernfalls wird das Objekt verzerrt.

Eine Skalierung bewirkt, dass sich der Abstand jedes Punktes vom Ursprung des Koordinatensystems (allgemeiner vom *Fixpunkt*) entsprechend der Skalierungsfaktoren ändert. Die Wahl eines beliebigen Fixpunktes Z wird durch eine Aneinanderkettung von Transformationen erreicht: Translation um $(-Z_x, -Z_y)$, Skalierung mit (s_x, s_y) bezüglich des Ursprungs und anschließende Rücktranslation um (Z_x, Z_y).

Die *Rotation* eines Grafikobjekts wird durch den Rotationswinkel δ festgelegt. Die Rotationsformel für den Punkt $P = (x, y)$ lautet

$$\begin{aligned} x' &= x \cdot \cos\delta - y \cdot \sin\delta \\ y' &= y \cdot \cos\delta + x \cdot \sin\delta \end{aligned}$$

Wie bei der Skalierung kann die Rotation um ein beliebiges Rotationszentrum (R_x, R_y) durch Translation um $(-R_x, -R_y)$, Rotation um den Ursprung und darauffolgender Rücktranslation um (R_x, R_y) realisiert werden.

Allgemein ist darauf zu achten, dass geometrische Objekte durch Transformationen ihren Typ ändern können. Soll zum Beispiel ein Rechteck, das nur durch zwei gegenüberliegende Eckpunkte definiert ist, gedreht werden, so bleibt seine Form im Allgemeinen nur erhalten, wenn es vor der Drehung in ein Polygon mit vier Eckpunkten umgewandelt wird. Die beschriebene Situation tritt im Rahmen der vorliegenden Diplomarbeit nicht ein, da nur Polygone, Linien und Punkte verwendet werden.

Zusammengesetzte Transformationen

Nicht nur durch die Wahl eines beliebigen Fixpunktes bei der Skalierung oder eines beliebigen Zentrums für die Rotation kommt es häufig vor, dass eine Folge von

unterschiedlichen Transformationen auf ein Grafikobjekt angewendet werden soll. Anstatt alle elementaren Transformationen einzeln auf jedem Definitionspunkt auszuführen, bietet die Anwendung von Matrizenmethoden einen effizienteren Ansatz.

Die Einführung von *homogenen Koordinaten* erlaubt die Darstellung der Transformationsgleichungen in einer einheitlichen Matrixform. Jeder Punkt $P = (x, y)$ bekommt die homogene Koordinatenschreibweise $[x_h, y_h, w]$ zugeordnet, wobei w einen Wert ungleich Null besitzt und

$$x_h = x \cdot w \quad \text{und} \quad y_h = y \cdot w$$

gilt.

Die Transformationen werden durch 3×3-Matrizen dargestellt, so dass sich sowohl die Transformation eines Punktes als auch die Verknüpfung von Transformationen als Matrizenmultiplikation realisieren lässt. Die Matrizen werden dabei jeweils von rechts an den homogenen Vektor $[x_h, y_h, w]$ oder die Matrix einer vorhergehenden Transformation multipliziert. Die elementaren Transformationen *Translation*, *Skalierung* und *Rotation* lauten in Matrixschreibweise:

$$\begin{aligned} T(t_x, t_y) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \\ S(s_x, s_y) &= \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ R(\delta) &= \begin{bmatrix} \cos\delta & \sin\delta & 0 \\ -\sin\delta & \cos\delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

Dabei ist zu beachten, dass für die Auswertung einer zusammengesetzten Transformation die Regeln der Matrizenmultiplikation berücksichtigt werden müssen: Es gilt das Assoziativgesetz, jedoch im Allgemeinen nicht das Kommutativgesetz.

Die Transformationsmatrix für eine Rotation um 90° bezüglich des Punktes $(2, 5)$ lautet beispielsweise

$$\begin{aligned} A &= T(-2,-5) \cdot R(\frac{\pi}{2}) \cdot T(2,5) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -2 & -5 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 5 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 7 & -3 & 1 \end{bmatrix} \end{aligned}$$

Für den Punkt $P = (3, 3)$ ergibt sich daraus

$$P' = P \cdot A = [3, 3, 1] \cdot \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 7 & -3 & 1 \end{bmatrix} = [4, 0, 1] = (4, 0)$$

Der Vorteil der Matrixschreibweise ist, dass für eine Folge von Transformationen zuerst die Transformationsmatrizen multipliziert werden, bevor die Koordinaten eines Grafikobjekts modifiziert werden. Dadurch wird die Anzahl der Berechnungsschritte gegenüber der sequentiellen Ausführung der einzelnen Transformationen erheblich reduziert. Der Ansatz hilft außerdem, Rundungsfehler zu vermeiden, da nicht nach jeder einzelnen Transformation ein (eventuell ganzzahliges) Ergebnis berechnet werden muss.

Eine Herleitung von *dreidimensionalen Transformationen* lässt sich durch Hinzunahme einer dritten Koordinate für die z-Richtung relativ einfach aus den genannten zweidimensionalen Transformationen vollziehen. Nur die Beschreibung der 3D-Rotation ist etwas aufwendiger, da ein Grafikobjekt im dreidimensionalen Raum um jede beliebige Achse gedreht werden kann.

Eine Verallgemeinerung der vorgestellten Transformationen stellen die *affinen Abbildungen* dar. Sie werden jedoch im Rahmen dieser Diplomarbeit nicht benötigt und werden deshalb nicht vertieft.

4.4.2 Clipping

Die Wettervorhersagedaten des DWD umfassen nicht nur Deutschland, sondern einen grossen Teil Westeuropas (325*325 Rasterpunkte mit einem Abstand von 7km bedecken eine Fläche von 2275*2275km). Das Beschneiden der Daten auf einen gewünschten rechteckigen Bereich - auch *Fenster* genannt - beschreiben sogenannte *Clipping-Algorithmen*. Diese Algorithmen entfernen Grafikobjekte, die außerhalb des Fensters liegen und beschneiden Objekte, die nur teilweise im sichtbaren Ausschnitt liegen. Dadurch kann die Menge der zu verarbeitenden Daten je nach Fenstergröße eventuell erheblich reduziert werden.

Punkt-Clipping

Für einzelne Punkte ist es einfach zu bestimmen, ob sie innerhalb des erlaubten Bereichs liegen. Das Fensterrechteck ist durch die Eckpunkte (x_{min}, y_{min}) und

(x_{max}, y_{max}) eindeutig bestimmt und ein Punkt P_i liegt innerhalb der Fenstergrenzen, sofern

$$x_{min} \leq x_i \leq x_{max} \quad \text{und} \quad y_{min} \leq y_i \leq y_{max}$$

gilt. Alle Punkte, die diese Bedingung nicht erfüllen, werden entfernt.

Linien-Clipping

Komplizierter ist bereits das Clipping von Linien. Denn eine Linie beschreibt eine Strecke zwischen zwei Punkten, die keinen, einen oder zwei Schnittpunkte mit den Fensterkanten haben kann. Existiert kein Schnittpunkt von Linie und Fensterkante, liegt die Linie komplett innerhalb oder komplett außerhalb des Fensters. Bei einem Schnittpunkt liegt einer der Definitionspunkte der Linie innerhalb, der andere außerhalb des Ausschnitts. Existieren zwei Schnittpunkte, liegen beide Definitionspunkte außerhalb des sichtbaren Bereichs. Die Linie verläuft aber durch das Fenster.

Ein auf *Cohen* und *Sutherland* zurückgehender Algorithmus unterteilt die Ebene anhand der Clipping-Grenzen (den Fensterkanten) in 9 Bereiche (siehe Abbildung 4.15) und ordnet den Definitionspunkten den jeweiligen Bereichscode zu.

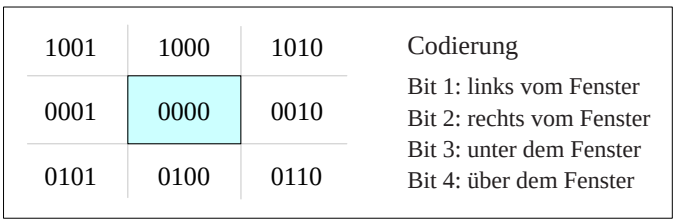


Abbildung 4.15: Bereichscodes des Algorithmus von Cohen und Sutherland

Die Bits der Bereichscodes ermöglichen effiziente Tests auf die Sichtbarkeit von Linien. Gilt die Formel

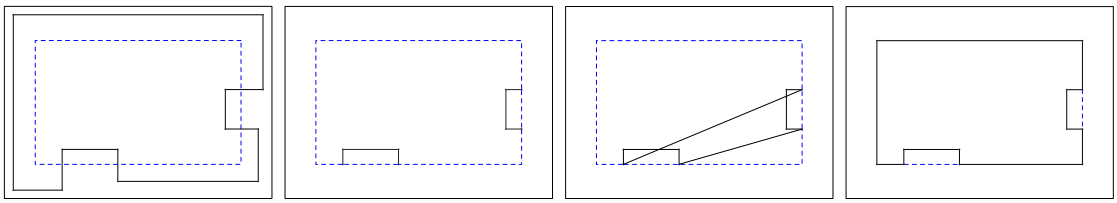
$$code(P_1)\ \&\&\ code(P_2)\ <>\ 0$$

ist die Linie $\overline{P_1 P_2}$ unsichtbar und muss nicht weiter berücksichtigt werden. Andernfalls wird die Linie mit einer Fensterkante geschnitten und der Test wiederholt. Die Iteration bricht ab, sobald beide Endpunkte der Linie innerhalb oder außerhalb des Fensters liegen. Dieser Fall tritt nach spätestens vier Schritten ein. Befindet sich die Linie komplett innerhalb des sichtbaren Bereichs, gilt die Formel

$$code(P_1)\ ||\ code(P_2)\ ==\ 0$$

Polygon-Clipping

Für das Clipping von Polygonen reicht es nicht aus, das Linien-Clipping sequentiell auf die Kanten des Polygons anzuwenden. Das Polygon kann in mehrere unzusammenhängende Teile zerfallen, wie Abbildung 4.16b zeigt.



(a) vorher (b) falsch (c) falsch (d) korrekt

Abbildung 4.16: Clipping eines Polygons

Stattdessen müssen zusätzlich die Ein- und Austrittspunkte verbunden werden, wobei die Ecken des Clipping-Fensters eine Spezialbehandlung erfordern. Die Abbildungen 4.16c veranschaulicht diese Problematik.

Der Polygon-Clipping-Algorithmus von *Sutherland* und *Hodgman* [SuHo1974] clippt daher das gesamte Polygon an den vier Fensterkanten nacheinander, anstatt jede Kante des Polygons dem Clipping-Prozess zu unterwerfen:

```
foreach Clipping-Gerade G do
  foreach Polygonpunkt Pi do
    if ( Pi sichtbar )
      übernimm Pi
    if ( PiPi+1 schneidet G )
      übernimm den Schnittpunkt
```

Der Algorithmus eignet sich auch für das Clipping an beliebigen polygonalen Fenstergrenzen, soll aber im Folgenden nur in seiner einfachen Form für rechteckige Fenster angewendet werden.

Eine ausführlichere Behandlung der beschriebenen Transformationen und Clipping-Algorithmen findet sich in [Fell1992].

4.5 Gestaltung der Ausgabe

Die vorherigen Abschnitte stellen die benötigten Methoden zur Verfügung, um aus den Rasterdaten des Deutschen Wetterdienstes Isoflächen zu generieren und diese

mitsamt der Landkarten aus dem GIS-Bereich in eine Projektion zu bringen, die dem Anwender vertraut ist. Durch verschiedene Transformationen der Grafikobjekte und die abschließende Wahl eines gewünschten Ausschnitts können die vorliegenden Daten nun in einem XML-Format gespeichert werden, das direkt vom geplanten Visualisierungsprozess (vgl. Abbildung 3.6 auf Seite 45) unter Zuhilfenahme eines geeigneten XSLT-Stylesheets verarbeitet werden kann.

Für die Vervollständigung des Datenflusses von GRIB-Dateien und GIS-Daten zu animierten Flash-Filmen fehlt nur noch der Inhalt des XSLT-Stylesheets, d.h. die Umsetzung von Temperaturflächen und -isolinien, Landkartenumrissen, Flüssen und Städten in eine grafische Darstellung. Das Stylesheet enthält - wie bereits der Name sagt - die Gestaltung der Ausgabe.

Von Anfang an wurde mit den aus den Rasterdaten erzeugten Polygonen eine flächendeckende Darstellung der Wetterdaten angestrebt. Denn die stattdessen häufig verwendeten punktuellen Angaben von Temperatur und Niederschlag (z.B. 5°C in Berlin) fordern die Fähigkeiten von Macromedia Flash nicht heraus. Außerdem enthalten die vorliegenden Wetterdaten viel präzisere Werte, die dem Anwender nach Bedarf zur Verfügung stehen sollen, so dass z.B. selbst ein Einwohner eines Dorfes in der Lüneburger Heide die exakte Vorhersage für seine Region abfragen kann.

Flash unterstützt das zusätzliche Einblenden von Informationen durch verlustfreies Zoomen - Ausschnitte können vergrößert betrachtet werden - und die Skriptsprache ActionScript. Durch Interaktion mit dem Benutzer können über ActionScript-Anweisungen die Darstellung von Landkartenmerkmalen geregelt und mehrere Ebenen mit unterschiedlichen Wetterinformationen ein- und ausgeblendet werden. Die Kombination von Ebenen in beliebiger Reihenfolge ist ebenfalls denkbar.

Zusätzlich kann eine detailliertere Darstellung z.B. durch einen *Kachelungs-Mechanismus* erreicht werden.[13] D.h. hinter der Grafik einer Einstiegsseite, die dem Benutzer bereits einen Überblick verschafft, verbergen sich für vorgegebene Ausschnitte präzisere Flash-Filme, die durch Hyperlinks innerhalb der Startgrafik gewählt werden können. So erhält der Internetnutzer für den von ihm gewählten Bereich (für eine Kachel) in einem weiteren Flash-Film alle zur Verfügung stehenden Informationen, muss aber nicht für die gesamte Wettervorhersagekarte die höchte Detailstufe aus dem Internet herunterladen. Kachelungs-Mechanismen reduzieren die Warte- und Onlinezeit des Benutzers. Wichtig ist, dass sich die Kacheln überlappen, so dass keine Stelle der ursprünglichen Karte nur auf einer Kante zu liegen kommt.

[13] Als Vorbild könnte z.B. www.stadtplan.net dienen.

Die meisten der oben genannten Gestaltungsmöglichkeiten sind jedoch Thema der bereits erwähnten Diplomarbeit meines Kommilitonen Ralf Kunze. Eine Benutzeroberfläche zur Steuerung der Sichtbarkeit von Ebenen mit unterschiedlichen Parametern und zur Beeinflussung der Animation wird ebenfalls von ihm entwickelt werden.

Im Rahmen der vorliegenden Arbeit werden Flash-Filme die Daten des DWD - Temperatur, Niederschlag, Luftdruck und Bewölkung - vorerst nur in einer Art Diashow präsentieren. Eine spätere Integration von Kachelung und ActionScript ist jedoch vorgesehen. Ebenso ist beim Entwurf der Gestaltung der einzelnen Parameter darauf zu achten, dass später die Darstellung möglichst beliebiger Kombinationen der Ebenen möglich ist.

Für die Visualisierung von Temperaturwerten bieten sich Flächen mit einer Füllfarbe der Farbskala von lila für sehr kalte Zonen über blau, grün und gelb zu orange und rot für sehr hohe Temperaturen an. So kennt der Anwender die grafische Darstellung von Temperaturbereichen aus Zeitungen und der Tagesschau.

Die Gestaltung der Bewölkung ist ebenfalls leicht zu entwerfen. Da sich bei Flash alle Farben aus einem RGB-Wert und einem zusätzlichen Alphakanal zusammensetzen, der die Transparenz der Farbe, d.h. den Grad der Durchsichtigkeit, festlegt, können Wolken mit zunehmender Dichte durch einen höheren Deckungsgrad visualisiert werden. Wolken werden typischerweise als weiße Flächen und bei besonders starker Bewölkung in Grautönen dargestellt.

Die Menge des Niederschlags repräsentieren auf Klimakarten normalerweise farbige Flächen in den Farben rosa oder hellbraun für keinen Niederschlag über blau zu lila für hohen Jahresniederschlag. Wetterkarten im Fernsehen verwenden stattdessen für die Niederschlagsvorhersage animierte Regentropfen oder Schneeflocken. Für die Kombination mit anderen Parametern kommt alternativ auch eine Füllung der Flächen mit einem Muster in Frage, das je nach Stärke des Niederschlags kleinere Punkte (Tropfen) mit großen Abständen bzw. größere Punkte mit kleineren Abständen verwendet.

Luftdruck wird üblicherweise durch Isobaren[14], d.h. Linienzügen mit der Angabe des jeweiligen Isowertes, beschrieben. Prinzipiell ist auch für alle anderen Parameter eine solche Darstellung denkbar, wenn statt gefüllten Isoflächen nur deren Umrisse ausgegeben werden, für den Luftdruck ist sie jedoch typisch.

[14] Isolinien in Verbindung mit Druck heißen Isobaren.

II Realisierung und Anwendung

5 Realisierung

Betrachtet man die vorhergehenden Themen im Zusammenhang, entsteht ein Datenfluss von den vorliegenden Eingabedaten im GRIB- und Shapefile-Format und ihrer computergrafischen Verarbeitung in Kapitel 4 über das Datenaustauschformat XML, der dazugehörigen Transformationssprache XSLT und dem Flash-Generator Saxess Wave in Kapitel 3 bis hin zum fertigen Flash-Film und seinem Dateiformat SWF in Kapitel 2. Abbildung 3.6 auf Seite 45 zeigte bereits einige der genannten Komponenten als Teil des angestrebten *Visualisierungsprozesses*.

Dieser Prozess geht jedoch davon aus, dass die Eingabedaten erstens in einem XML-Format vorliegen und zweitens Grafikobjekte enthalten, die bereits für eine Visualisierung vorbereitet sind. Dies ist im Allgemeinen aber nicht der Fall, wie schon in der Einleitung von Kapitel 4 angemerkt. Vielmehr ist z.B. bei den Wetterdaten des DWD die Anwendung verschiedener Transformationen - Vektorisierung der Rasterdaten, Änderung der Kartenprojektion, Clipping und andere - notwendig, bevor schließlich Polygone die gewünschten Temperatur- oder Niederschlags-Isoflächen für einen bestimmten Ausschnitt widerspiegeln und zudem Koordinatenangaben enthalten, die direkt als Pixelpositionen innerhalb eines Flash-Films verwendet werden können.

Die XML-Softwareprodukte der Apache Software Foundation, der XML-Parser Xerces-J und der XSLT-Prozessor Xalan-Java, unterstützen das Einlesen von XML-Dateien und die Umwandlung der Daten in die XML-Ausprägung SWFML, aus der letztlich mittels Saxess Wave ein Flash-Film generiert werden kann. Sämtliche Schritte, die aber vorher zur Erzeugung einer XML-Quelldatei mit dem Inhalt des Flash-Films erfolgen müssen, sind in „Handarbeit“ durchzuführen.

Das folgende Kapitel stellt daher die Realisierung eines selbst entwickelten, möglichst flexiblen Mechanismus vor, der die Bearbeitung von in XML codierten Grafikobjekten ermöglicht.

Die Implementation erfolgt, wie in Abschnitt 2.5 festgelegt, in Java. Eigene Klassen sind in Java-Paketen organisiert, die mit der Bezeichnung `de.flashweather` beginnen. Genauere Angaben zu ihrem Funktionsumfang befinden sich in der mit `Javadoc` erzeugten API-Dokumentation auf der beigefügten CD-ROM.

5.1 Datenfluss und Datenformate

5.1.1 Vollständiger Datenfluss

Da sich aufwendigere Transformationen und computergrafische Algorithmen nur auf Grafikobjekte anwenden lassen, die auch als solche im Speicher vorliegen und nicht etwa lediglich in Textform als Teil eines XML-Dokuments, werden Java-Klassen benötigt, die eine Schnittstelle zwischen Java und XML schaffen. Sie werden im Folgenden die *Java-Grafikobjekte* genannt. Diese Klassen sind in der Lage, Teile eines XML-Baums selbstständig zu erfassen und daraus das dazugehörige Grafikobjekt mit seinen Koordinaten und Attributen aufzubauen. Umgekehrt ermöglichen Methoden, die ursprüngliche XML-Struktur eines Grafikobjekts wieder in einen XML-Baum zurückzuschreiben. Die Verwendung der Java-Grafikobjekte ohne den Bezug zu einer XML-Quelle ist ebenfalls möglich.

Jedes Java-Grafikobjekt stellt über eine Programmierschnittstelle (API) Methoden zur Verfügung, die z.B. den Zugriff auf die Definitionspunkte des Objekts erlauben oder einfache Transformationen durchführen. Diese Schnittstelle wird von externen *Transformationsklassen* genutzt, um aufwendigere Transformationen der Grafikobjekte (Clipping, Projektion) durchzuführen. Eine entsprechende Erweiterung des bisherigen Datenflusses zeigt Abbildung 5.1.

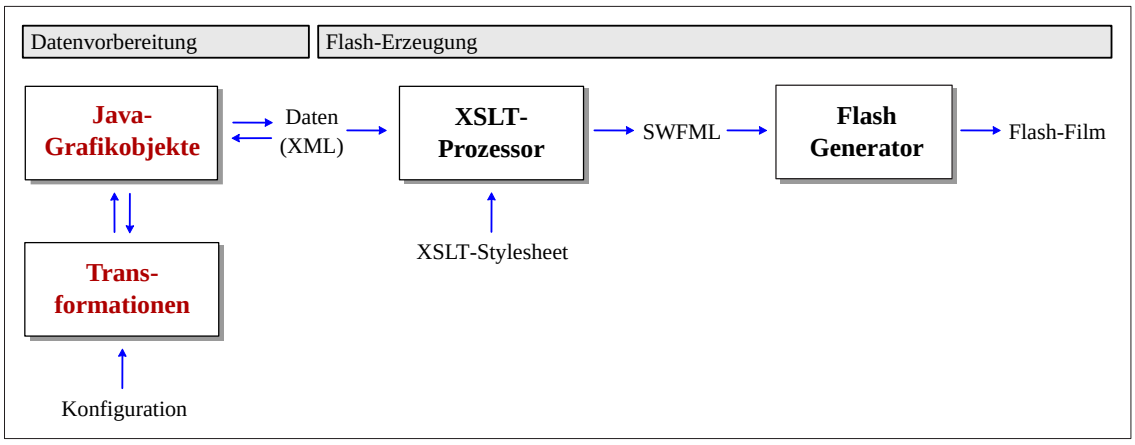


Abbildung 5.1: Vollständiger Datenfluss

Dieser Entwurf ermöglicht, dass ein mehrfacher Wechsel zwischen XML-Repräsentation und Java-Grafikobjekt stattfinden kann. Jeder Lese- und Schreibvorgang ist selbstverständlich zeitaufwendig, kann aber eventuell beabsichtigt sein. Werden z.B. die Daten einer Deutschlandkarte bereits in der gewünschten Projektion für eine spätere Visualisierung bereitgestellt, ist für die Darstellung unterschiedlicher Ausschnitte dieser Karte erneut eine Umwandlung der XML-Daten in Java-Grafikobjekte nötig.

5.1.2 Allgemeines Datenformat

Damit Java-Klassen ein textbasiertes Format wie XML lesen und verarbeiten können, müssen zumindest gewisse Grundregeln festgelegt werden. Fest steht, dass die XML-Dateien mit Hilfe von beliebigen Tags codierte Grafikobjekte und eventuell weitere Informationen enthalten. Eine Java-Klasse muss jedoch die Namen der XML-Tags und -Attribute kennen, um aus ihnen die korrekten Informationen herauslesen und das entsprechende Grafikobjekt konstruieren zu können. Auch die Struktur bzw. Verschachtelung der Tags ineinander ist von Bedeutung. So lautet z.B. die Definition eines zweidimensionalen Polygons in SWFML, dem Eingabeformat von Saxess Wave:

```
<polygon                          Polygon-Tag
    ID="poly1"                    eindeutige Kennung
    color="C0FFB080"              Füllfarbe des Polygons
    lw="1" lc="00FFFFFF">         Stärke und Farbe der Umrandung
    <point x="20" y="20"/>        Eckpunkte mit
    <point x="50" y="20"/>        ... ganzzahligen Koordinaten
    ...                           weitere Eckpunkte
</polygon>                        Ende des Polygon-Tags
```

In Anlehnung an das SWFML-Format, wurde für die Speicherung von Grafikobjekten die folgende XML-Struktur entworfen:

```
<objecttag                        beliebiges Tag des Grafikobjekts
    attr1="value1"                Attribut
    ...                           weitere Attribute
    <infotag attr1=".." ... />    Tag, das keinen Punkt enthält
    <pointtag xattr="#.#"         Punkt mit
              yattr="#.#"/>       ... Gleitkomma-Koordinaten
    ...                           weitere Punkte
</objecttag>                      Ende des Grafikobjekt-Tags
```

Ein äußeres Tag repräsentiert das Grafikobjekt, indem es die Attribute desselben besitzt und dessen Definitionspunkte und weitere Informationen als untergeordnete Tags umschließt. Alle Tags und Attributnamen sollen jedoch im Gegensatz zum SWFML-Polygon frei wählbar sein. Nur so können unterschiedliche Datenformate mit semantisch benannten Tags verwendet werden, wie XML es vorsieht (z.B. `<river>` und `<city>` in einem Format für Landkarten).

Die Einschränkung, dass Grafikobjekte in XML in der oben genannten Datenstruktur definiert sein müssen, ist notwendig, damit sich Java-Klassen realisieren lassen,

die die XML-Daten einlesen und interpretieren können. Die Fähigkeit der Klassen, XML-Formate mit beliebigen Tag- und Attributnamen verarbeiten zu können, kann nicht vollständig umgesetzt werden. Sie kann nur insoweit verwirklicht werden, dass die Klassen für eine beliebige Ausprägung des Formats konfigurierbar sind. Ein Einlesen von in XML codierten Grafikobjekten ist also nur möglich, wenn die vorgeschriebene Struktur der Daten eingehalten wird und die Namen der Tags und Attribute bekannt sind.

Dabei ist die oben gezeigte Datenstruktur für die Speicherung von Grafikobjekten in XML auf die Anforderungen der vorliegenden Arbeit angepasst. Denn die verwendeten Grafikobjekte sind zweidimensionale Polygone, Linien (bestehend aus mehreren Punkten) und Punkte. Spezialfälle wie Kreise oder Rechtecke, die durch einen Radius oder gegenüberliegende Eckpunkte definiert sind, kommen nicht vor.

5.2 Grafikobjekte in Java

5.2.1 Grafikobjekte

Für die Verwaltung zweidimensionaler Raster- und Vektordaten in Java werden zwei unterschiedliche Gruppen von Grafikobjekten benötigt: *Einfache Punkte* und *komplexe Grafikobjekte* wie z.B. Polygone oder Linien, die sich aus den zuerst genannten Punkten zusammensetzen. Beide Gruppen von Grafikobjekten sollen gleichermaßen die grundlegenden Transformationen (Verschiebung, Skalierung, Drehung) beherrschen.

Einfache Punkte enthalten die Angaben zu ihrer Position innerhalb eines Koordinatensystems und speichern die Werte der x- und y-Koordinate. Ein Punkt, der zu einem Raster (engl. *grid*) gehört, verfügt zudem über die Information, welchen skalaren Wert der Parameter (z.B. Temperatur) an der betreffenden Stelle annimmt, und speichert ihn zusätzlich zu den Koordinatenangaben. Eine Speicherung von zusätzlichen Attributen für einfache Punkte ist nicht vorgesehen. Die beiden Punkt-Objekte entsprechen damit XML-Ausdrücken der Form:

```
<point     x="2.5" y="3.7"/>
<gridpoint x="2.5" y="3.7" value="12.4"/>
```

Komplexe Grafikobjekte verwenden diese einfachen Punkt-Objekte für die Speicherung ihrer Definitionspunkte. Ihnen ist jedoch erlaubt, neben den eigentlichen Positions-Informationen weitere Attribute zu beherbergen, die keinen direkten Bezug zur Definition der grafischen Form haben. Komplexe Grafikobjekte entsprechen damit einem Ausdruck wie er im vorherigen Abschnitt als Entwurf für die

Speicherung von Grafikobjekten in XML vorgestellt wurde. Beliebige Attribute des Start-Tags und zusätzliche Sohn-Knoten werden während einer Transformation des Grafikobjekts zwischengespeichert und schließlich bei einer Generierung des entsprechenden XML-Codes wieder mit ausgegeben.

Dazu ein Beispiel: Ein See, dessen Umriss in XML durch ein einleitendes `lake`-Tag und untergeordnete Definitionspunkte beschrieben ist, besitzt einen Namen als Attribut des `lake`-Tags: `<lake name=”Dümmer”>`. Dieser Name ist für die grafische Bearbeitung des Seeumrisses nicht von Bedeutung, wird aber von der Java-Klasse zwischengespeichert, um eine spätere Ausgabe des Sees mit allen seinen Attributen zu ermöglichen. Eventuelle Attribute der Defintionspunkte (außer den Koordinaten) gehen jedoch durch die Bearbeitung verloren. Abbildung 5.2 veranschaulicht das Wechselspiel von Java und XML.

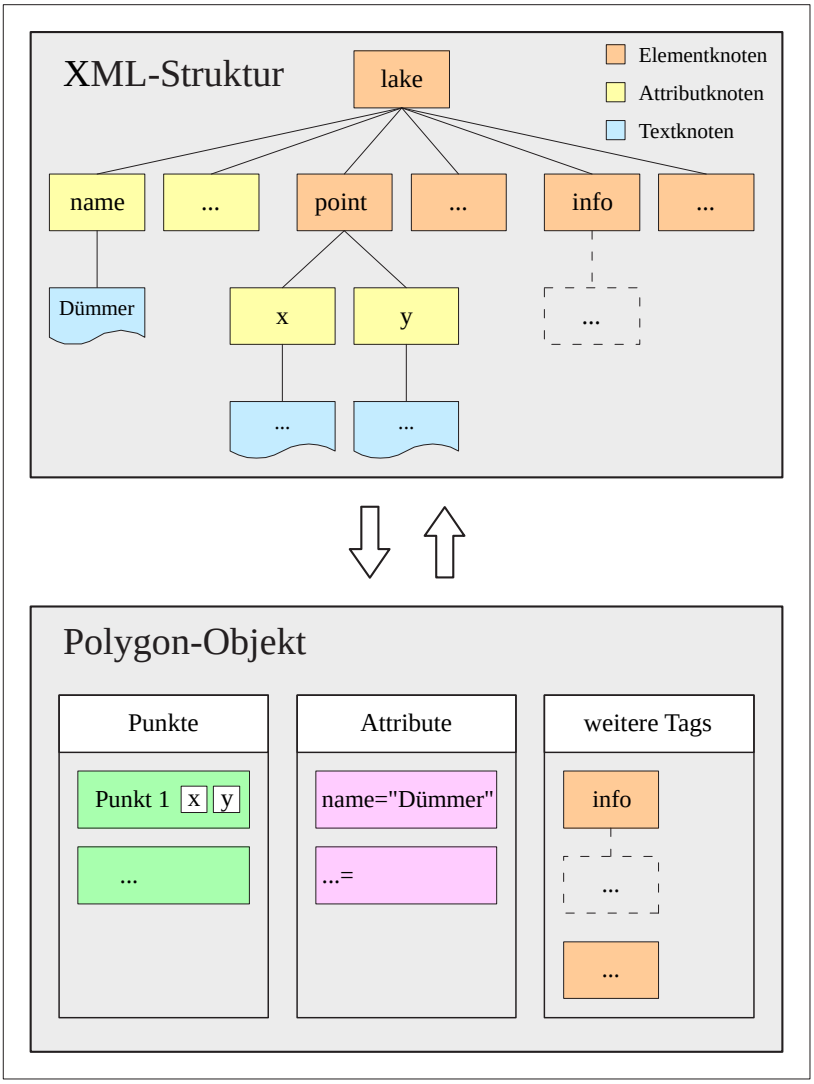


Abbildung 5.2: Grafikobjekte in XML und Java

Die Einschränkung, dass einfache Punkte außer ihrer Position und (bei einem Rasterpunkt) dem Wert des Parameters keine weiteren Attribute besitzen dürfen, muss aus Performancegründen gemacht werden. Ein Raster, das die kompletten Daten einer Stundenprognose des DWD enthält, umfasst z.B. 325 * 325 = 105.625 Punkte. Ein immenser Speicherbedarf wäre vorhersehbar, dürfte jeder dieser Punkte zusätzliche Attribute beherbergen.

5.2.2 Java-Klassenhierarchie

Aus den Überlegungen des vorhergehenden Abschnitts ergibt sich eine Klassenhierarchie für die Implementation der benötigten Java-Grafikobjekte. Abbildung 5.3 stellt sie in einer Baumstruktur dar.

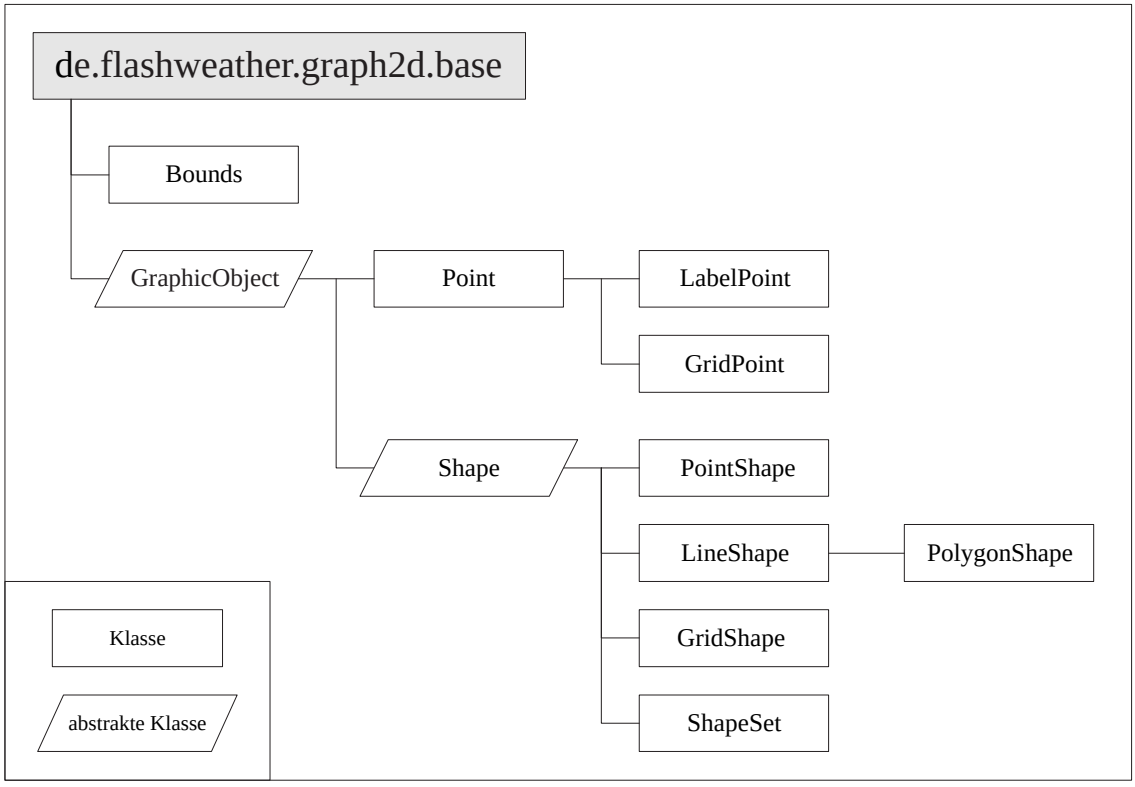


Abbildung 5.3: Klassenhierarchie der Java-Grafikobjekte

Die abstrakte Basisklasse aller Grafikobjekte heißt `GraphicObject`. Einfache Punkte sind `Point`-Objekte oder sie stammen von dieser Klasse ab. Komplexe Grafikobjekte werden von der abstrakten Klasse `Shape` repräsentiert, von der speziellere Typen abstammen.

5.2.3 Einzelne Java-Klassen

Die Grafikobjekt-Klassen des Java-Pakets `de.flashweather.graph2d.base` werden nun im Einzelnen vorgestellt:

- `Bounds`
  stammt als einzige Klasse dieses Pakets nicht von `GraphicObject` ab, da es kein Grafikobjekt sondern die Ausdehnung eines solchen in der Form eines Rechtecks repräsentiert. Über Methoden lassen sich die minimalen und maximalen Koordinaten auslesen (`getXMin`, `getYMin`, `getXMax`, `getYMax`) und weitere Punkte oder `Bounds`-Objekte umschließen (`include`).
- `GraphicObject`
  ist die abstrakte Basisklasse aller Grafikobjekte. Sowohl einfache Punkt-Objekte (`Point`) als auch komplexe Grafikobjekte (`Shape`) stammen von ihr ab und implementieren die abstrakten Methoden, die elementare Transformationen (`translate`, `scale`, `rotate`, `transform`) und die Ausgabe einer XML-Repräsentation (`serialize`) des Java-Objekts zur Verfügung stellen.
- `Point`
  ist ein 2D-Punkt und das einfachste Grafikobjekt. Objekte dieser Klasse verfügen über Methoden zum Lesen (`getX`, `getY`) und Setzen (`setCoordinates`) der Koordinaten, zum Vergleichen des Punkts mit einem anderen (`distanceTo`, `equals`) und Operationen, die Projektionen (`getSurfacePoint`) und Clipping (`isInsideWindow`) unterstützen.
- `LabelPoint`
  ist ein `Point`, der die Position eines Labels, d.h. die Bezeichnung eines Grafikobjekts, markiert.
- `GridPoint`
  ist eine Erweiterung der Klasse `Point` und repräsentiert einen Rasterpunkt. Die Klasse verfügt daher über zusätzliche Methoden zum Lesen (`getValue`) und Setzen (`setValue`) des zum Rasterpunkt gehörenden Skalarwertes.
- `Shape`
  stammt von `GraphicObject` ab und ist die abstrakte Basisklasse aller komplexen Grafikobjekte. Die einzigen abstrakten Methoden sind `cloneHull`, die das Grafikobjekt klont, ohne die Definitionspunkte ebenfalls zu klonen, und `getAllPoints`, die alle Definitions- und Labelpunkte liefert. Weitere Methoden für den Zugriff auf die Definitionspunkte eines komplexen Grafikobjekts stehen - je nach Typ - in den Subklassen zur Verfügung. `Shape` selbst implementiert Methoden zum Verwalten von weiteren Attributen (`get-/set-`

`Attribute`, `read-/writeAttributes`) und zusätzlichen Sohnknoten innerhalb der XML-Struktur (`addInfoNode`, `getInfoNode`), außerdem zum Lesen (`getBounds`) und Setzen (`setBounds`, `setHeight`, `setWidth`) der Ausdehnung des Grafikobjekts.

- `PointShape`
  ist das einfachste komplexe Grafikobjekt und beinhaltet einen einzelnen Punkt (z.B. eine Stadt). Neben den abstrakten Methoden der Superklassen `Shape` und `GraphicObject` implementiert die Klasse die Methoden zum Lesen (`getPoint`) und Setzen (`setPoint`) dieses `Point`-Objekts.

- `LineShape`
  stammt von `Shape` ab und gehört daher zu den komplexen Grafikobjekten. Die Klasse repräsentiert einen Linienzug, der sich über mehrere Punkte erstrecken kann. Sie implementiert Methoden zum Lesen (`getPoint`, `getPoints`, `getFirstPoint`, `getLastPoint`), Setzen (`addPoint`, `insertPointAt`, `setPoints`) und Löschen (`removePoint`) der Definitionspunkte und zusätzlich Methoden zum Verwalten von Punkten, an deren Position die Linie mit Labels versehen werden soll (`addLabelPoint`, `getLabelPoint`, `setLabelPoints`). Die Anzahl der Linienpunkte kann ebenso ermittelt werden (`getLength`) wie die Existenz eines bestimmten Punkts auf der Linie (`searchPoint`).

- `PolygonShape`
  stammt von `LineShape` ab, da ein Polygon einen geschlossenen Linienzug darstellt. Zusätzlich implementiert dieses komplexe Grafikobjekt Methoden zum Lesen (`getFillIndex`) und Setzen (`setFillIndex`) des Füllindex der Polygonfläche - ein Wert der später in eine Farbe übersetzt werden kann.

- `GridShape`
  repräsentiert ein Raster und implementiert wie alle komplexen Grafikobjekte die abstrakt definierten `Shape`- und `GraphicObject`-Methoden. Die Rasterpunkte lassen sich auslesen (`getGridPoint`, `getGridPointValue`) und verändern (`setGridPoint`, `setGridPointValue`).
  Spezielle Methoden liefern die Dimension des Rasters (`getXDimension`, `getYDimension`), verändern die Skalarwerte an allen Rasterpunkten (`changeValues`), verschieben die Ränder des Rasters in die Mitte (`foldHorizontal`, `foldVertical`) und erstellen eine generalisierte Version des Rasters (`getGeneralized`), in der jeweils eine bestimmte Anzahl Rasterpunkte zu einem neuen zusammengefasst ist.

- `ShapeSet`
  ist eine spezielle Erweiterung von `Shape`, da diese Klasse sich selbst wie ein

komplexes Grafikobjekt verhält, aber gleichzeitig einen Container für diese Objekte darstellt und eine beliebige Anzahl von `Shape`-Objekten beinhalten kann. `ShapeSet` implementiert die Methoden der abstrakten Superklassen `Shape` und `GraphicObject`, indem die Aufrufe der Methoden an jedes einzelne `Shape`-Objekte weitergereicht werden. Außerdem stehen Methoden zur Verwaltung der `Shapes` zur Verfügung (`addShape`, `getShape`, `setShapes`).

5.2.4 Weitere Eigenschaften

Alle von `GraphicObject` abstammenden, nicht-abstrakten Klassen verfügen über mindestens drei *Konstruktoren*:

1. `Klassenname()`
   Der *leere Konstruktor*, der ein Grafikobjekt mit leerem Inhalt oder Standardwerten anlegt.
2. `Klassenname(` unterschiedliche Parameter `)`
   Ein Konstruktor, der Koordinaten, Definitionspunkte oder `Shape`-Objekte bereits als Parameter erhält und damit den Inhalt des Grafikobjekts füllt.
3. `Klassenname( Element )`
   Ein Konstruktor, der einen Knoten eines XML-Strukturbaums als Parameter übergeben bekommt (ein `Element`-Objekt) und anhand dieses Teilbaums über die Schnittstellen des DOM den Inhalt des Grafikobjekts festlegt.

Je nach Klasse erwartet der dritte Konstruktor als Parameter einen Teilbaum mit der in Abschnitt 5.1.2 festgelegten Struktur für komplexe Grafikobjekte oder ein einzelnes Punkt- bzw. Rasterpunkt-Tag mit entsprechenden Attributen. Die korrekte Interpretation der XML-Quelle kann jedoch nur erfolgen, wenn die Java-Klasse die Namen der verwendeten Tags und Attribute kennt. Nur anhand dieser Namen kann die Klasse entscheiden, ob und um welche Eigenschaften des Grafikobjekts es sich handelt. Tabelle 5.1 listet die XML-Merkmale auf, die den Grafikobjekt-Klassen bekannt sein müssen.

Die Forderung, dass alle Grafikobjekt-Klassen dennoch beliebige XML-Tags verarbeiten können, lässt sich nur realisieren, wenn die der Klasse bekannten XML-Merkmale konfigurierbar bleiben. Deshalb verfügt jede der genannten Klassen über Klassenvariablen, in denen der Name ihres XML-Tags und die Attributnamen gespeichert sind. Z.B. verfügt die Klasse `Point` über die Klassenvariablen `XML_TAG`, `XML_ATTR_X` und `XML_ATTR_Y`. Alle diese Klassenvariablen sind als `public` deklariert und dürfen von jeder externen Klasse an die eigenen Bedürfnisse angepasst werden.

Klasse	**bekannte XML-Merkmale**
`Point`	Tagname des Grafikobjekts Attributnamen der x- und y-Koordinate
`LabelPoint`	Tagname des Grafikobjekts Attributnamen der x- und y-Koordinate
`GridPoint`	Tagname des Grafikobjekts Attributnamen der x- und y-Koordinate und des Skalarwertes
`PointShape`	Tagname des Grafikobjekts Definitionspunkt → `Point`
`LineShape`	Tagname der Grafikobjekts Definitions- und Labelpunkte → `Point`, `LabelPoint`
`PolygonShape`	Tagname des Grafikobjekts Attributname des Füllindex Definitions- und Labelpunkte → `Point`, `LabelPoint`
`GridShape`	Tagname des Grafikobjekts Attributnamen der x- und y-Dimension des Rasters Rasterpunkte → `GridPoint`
`ShapeSet`	Tagname des Grafikobjekts Shapes → `...Shape`

Tabelle 5.1: In Grafikobjekt-Klassen bekannte XML-Merkmale

```
import de.flashweather.graph2d.base.Point;
import de.flashweather.graph2d.base.PointShape;
import org.apache.xerces.dom.DocumentImpl;         // DOM-Implementation

public class XMLOutput {

  public static void main( String[] args ) {

    PointShape.XML_TAG = "stadt";                  // konfiguriere
    Point.XML_TAG      = "lage";                   //   Grafikobjekt-
    Point.XML_ATTR_X   = "laengengrad";            //   Klassen
    Point.XML_ATTR_Y   = "breitengrad";

    PointShape p = new PointShape( 8.07, 52.27 );  // neues PointShape
    p.setAttribute("name", "Osnabrück");           // ergänze Attribut

    Document doc = new DocumentImpl();             // neues XML-Dokument
    p.serialize( doc );                            // hänge Teilbaum für
  }                                                //   Grafikobjekt ein
}
```

Quellcode 5.1: XML-Ausgabe eines Java-Grafikobjekts

Quellcode 5.1 verdeutlicht die Verwendung der Klassenvariablen. Das erzeugte `Document`-Objekt enthält zum Ende der Ausführung folgendes XML-Dokument:

```
<?xml version="1.0" encoding="UTF-8"?>
<stadt name="Osnabrück">
    <lage breitengrad="52.27" laengengrad="8.07"/>
</stadt>
```

Wird also ein neues Grafikobjekt erzeugt und sein Inhalt mit Hilfe von Java-Anweisungen erstellt, dienen die Werte der Klassenvariablen bei der Initialisierung zur Festlegung der XML-Tags und -Attribute für eine spätere XML-Ausgabe [15].

Wird ein neues Grafikobjekt jedoch aus einem XML-Teilbaum erzeugt, werden die Werte der Klassenvariablen ebenfalls bei der Initialisierung übernommen. Gleichzeitig werden sie aber auch für die korrekte Zuordnung der Attribute und Sohnknoten der XML-Quelle zu den internen Objektvariablen benötigt. Die vorherige Festlegung des einleitenden XML-Tags für das Grafikobjekt ist nicht erforderlich (schadet aber auch nicht), da der Konstruktor diesen Wert dem eindeutigen Wurzelelement des übergebenen XML-Teilbaums entnehmen kann (im Beispiel: `stadt`).

5.3 Lesemodule

5.3.1 Shapefile

Der Abschnitt 4.1.1 über Landkartendaten und das Dateiformat der ESRI Shapefiles [ESRI1998] deutete bereits an, dass das entsprechende Lesemodul nicht wie alle anderen Programme dieser Diplomarbeit in Java, sondern in Ansi-C implementiert ist. Es wird aber davon ausgegangen, dass die Kartendaten aus den Shapefiles nur ein einziges Mal in ein XML-Format zu konvertieren sind, da sich Landkarten nicht täglich ändern. Anschließend kann auf jedem Rechner mit den XML-Daten weitergearbeitet werden.

Das Shapefile-Format besteht im Wesentlichen aus einer Datei mit der Endung `.shp`, die die eigentlichen Vektorgrafikobjekte (Punkte, Linien oder Polygone) enthält, und einer Datei mit der Endung `.dbf`, die im dBASE-Datenbankformat die Attribute für jedes Grafikobjekt bereithält. Die ebenfalls in Abschnitt 4.1.1 bereits

[15] Dieser Vorgang ist nur dann *thread-safe*, wenn zwischen dem Setzen der Klassenvariablen und dem Erzeugen der Grafikobjekte kein anderer Thread die Werte der Klassenvariablen verändern kann. Das wird z.B. durch einen `synchronized`-Block bzgl. der verwendeten Klassen verhindert, sollten tatsächlich mehrere Threads existieren.

angesprochene kostenlose C-Bibliothek *shapefile* [Warm2000] erleichert die Realisierung eines Lesemoduls für Shapefiles. Mit Hilfe ihrer Programmierschnittstellen SHP-API und DBF-API konnte ein Konverter implementiert werden, der gleichzeitig durch beide Dateien läuft und die Informationen - Grafikobjekte und Attribute - zusammenführt. Eine Ausgabe der Daten erfolgt fortlaufend in einem gültigen XML-Format, das *MapML* genannt wurde. Der Konverter heißt entsprechend `shp2mapml`.

MapML ist keine offizielle XML-Ausprägung, sondern wurde speziell für die Anforderungen dieser Diplomarbeit entworfen. Quellcode 5.2 zeigt seine DTD.

```
<!ELEMENT  map    (part+)>
<!ATTLIST  map    type      (country|river|lake|city)  #REQUIRED>

<!ELEMENT  part   (point+)>
<!ATTLIST  part   id        ID     #REQUIRED
                  type      CDATA  #REQUIRED
                  label     CDATA  ""
                  category  CDATA  #IMPLIED>

<!ELEMENT  point  EMPTY>
<!ATTLIST  point  x         CDATA  #REQUIRED
                  y         CDATA  #REQUIRED>
```

Quellcode 5.2: MapML-DTD

Demnach besitzt eine Landkarte, die mit dem Tag `map` eingeleitet wird, einen Typ (`country`, `river`, `lake` oder `city`) und besteht aus mindestens einem `part`. Jeder `part` besitzt eine Kennung, einen Typ und wahlweise eine Bezeichnung und eine Kategorie. Ein `part` enthält mindestens einen Punkt als Tag mit dem Namen `point`. Die Koordinaten von Punkten sind als Attribute des `point`-Tags definiert.

Die Definition der Stadt Hamburg mit ihren geografischen Koordinaten lautet in MapML z.B.:

```
<?xml version="1.0"?>
<!DOCTYPE map SYSTEM "map.dtd">
<map type="city">
    <part id="city1" type="point" label="Hamburg" category="1">
        <point x="9.9748" y="53.5358"/>
    </part>
</map>
```

5.3.2 GRIB

In Abschnitt 4.1.2 wurde das GRIB-Dateiformat (*Gridded Binary*) [WMO1998] vorgestellt, das weltweit für die Speicherung von Wetterdaten genutzt wird. Es setzt sich aus Blöcken, den GRIB-*Records*, zusammen. Jeder Record enthält die Daten eines Parameters (z.B. Temperatur oder Luftdruck) und besteht wiederum aus fünf Sektionen.

Obwohl GRIB als offener internationaler Standard definiert wurde, konnten im Internet kaum Programme zum Auslesen des Dateiformats gefunden werden. Das mag daran liegen, dass das Format sehr flexibel gehalten ist und alle möglichen Parameter und beliebige Kartenprojektionen, in denen die Koordinaten der Rasterpunkte vorliegen können, berücksichtigt. Ein C-Programm, das eine umfangreiche Unterstützung von Parametertabellen und Rasterdefinitionen bietet und den Inhalt beliebiger GRIB-Dateien verarbeiten kann, heißt *wgrib* [NCEP2000]. Es wurde von einem Mitarbeiter des NCEP (National Climate Prediction Center der USA) entwickelt. Leider listet das Programm die Informationen zum Inhalt der Records nur in einer sehr kryptischen Form auf (siehe Abschnitt 4.1.2). Die Rasterdaten, d.h. die Werte des Parameters an den Rasterpunkten, können nur in separaten Dateien abgelegt und nicht über eine Schnittstelle direkt weiterverarbeitet werden.

Da sich Wetterprognosedaten aber täglich oder sogar stündlich ändern, ist es aus Speicherplatz- und Performancegründen nicht sinnvoll, die gesamten bitcodierten Daten im Textformat auf Festplatte zu speichern. Vorteilhafter ist eine programminterne Auswahl der benötigten Daten, die daraufhin für eine direkte Weiterverarbeitung zur Verfügung stehen. Um eine solche Handhabung der Wetterdaten in Java zu ermöglichen, wurde ein eigenes Lesemodul für GRIB-Dateien implementiert.

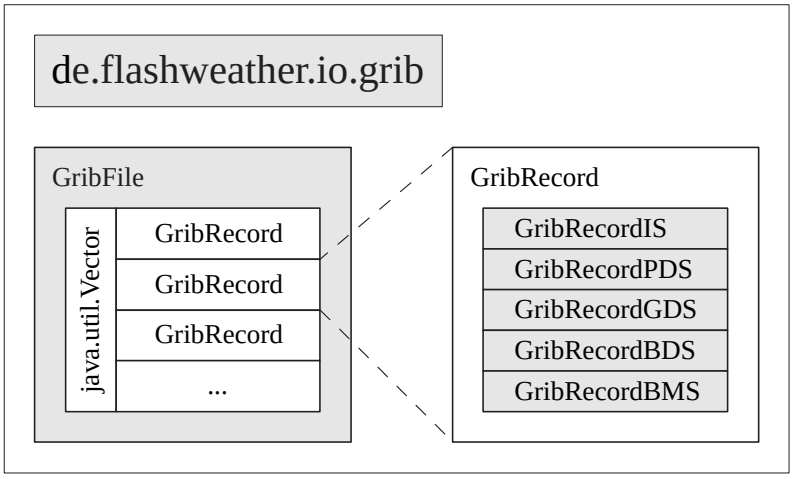


Abbildung 5.4: GRIB-Datei in Java

Abbildung 5.4 stellt die Klassen dar, aus denen sich eine GRIB-Datei in Java zusammensetzt. Im Vergleich zum prozeduralen C-Programm *wgrib* entsteht eine leicht zu überblickende Struktur, in der für jede Detailstufe spezielle Java-Klassen existieren, die zusammengenommen die GRIB-Datei ergeben.

Die Java-Klassen werden folgendermaßen verwendet: Um ein Raster mit den Werten eines bestimmten Parameters zu bekommen, muss aus dem `GribFile` der entsprechende `GribRecord` mit `getRecord` gewählt werden. Die Klasse `GribRecord` verfügt bereits über die elementaren Methoden zum Auslesen des GRIB-Records (`getDescription`, `getGridShape`, `getLevel`, `getTime`, `getType`, `getUnit`), von denen `getGridShape` die wichtigste Methode ist, da sie das eigentliche Raster in Form eines Java-Grafikobjekts liefert. Noch detailliertere Informationen enthalten die Klassen der fünf Sektionen des Records, die die Methoden `getIS`, `getPDS`, `getGDS`, `getBDS` und `getBMS` liefern.

```
import de.flashweather.io.grib.GribFile;
import de.flashweather.io.grib.GribRecord;

public class GribInfo {

  public static void main( String[] args ) {

    try {

      GribFile grib = new GribFile( args[0] );

      // durchlaufe alle Records der GRIB-Datei
      for (int rec=1; rec<=grib.getRecordCount(); rec++) {

        GribRecord record = grib.getRecord( rec );

        System.out.println("Record " + rec );
        System.out.println("  Type " + record.getType() + "\n");
        System.out.println( record + "\n");
      }
    }

    // Öffnen oder Lesen der GRIB-Datei fehlgeschlagen?
    catch ( Exception e ) {
      e.printStackTrace();
    }
  }
}
```

Quellcode 5.3: Hilfsklasse zum Anzeigen des Inhalts einer GRIB-Datei

Ist nicht bekannt, welche Daten eine GRIB-Datei enthält bzw. in welchem Record sich der gesuchte Parameter befindet, hilft ein kleines Tool (Quellcode 5.3). Die `toString`-Methode von `GribRecord` liefert eine Kurzinformation zum Inhalt des Records.

Aufgrund der angesprochenen Vielfältigkeit des GRIB-Dateiformats besitzt die eigene Java-Implementation eines Lesemoduls keinen Anspruch auf Vollständigkeit. Die Tabelle der Parameter wurde dem Programm *wgrib* entnommen und nur die häufig verwendeten Kartenprojektionen für die Definition des zugrunde liegenden Rasters wurden implementiert. Einige speziellere Projektionen und Einstellungen werden nicht unterstützt, doch konnten z.B. die von einem Internetrechner des National Weather Service der USA [NWS2000] heruntergeladenen Wetterprognosedaten sofort problemlos verarbeitet werden.

5.4 Vektorisierung von Rasterdaten

Als erstes Modul aus der Gruppe der Transformationsklassen soll im Folgenden die Implementierung und Verwendung des Algorithmus zur Vektorisierung von Rasterdaten aus Abschnitt 4.2 vorgestellt werden. Die einzelnen Schritte des Algorithmus wurden auf den Seiten 60 bis 65 bereits ausführlich beschrieben, so dass nicht erneut auf den gesamten Ablauf eingegangen wird. Stattdessen sollen einige interessante Passagen genauer erläutert werden, in denen die Arbeitsweise des Algorithmus anhand des Programmcodes und der Kommentare nicht sofort ersichtlich ist.

Die vollständige Implementation des Vektorisierungs-Algorithmus befindet sich in der Klasse `VectorizerImpl` im Paket `de.flashweather.graph2d.vectorize`.

Der Algorithmus benötigt als Input die Rasterdaten in Form eines rechteckigen, voll besetzten Feldes von zweidimensionalen Punkten und die Werte eines Parameters an diesen Punkten. `GridShape` ist die Grafikobjekt-Klasse, die genau für diesen Fall geschaffen wurde. Sie stellt ein rechteckiges Feld von `GridPoint`-Objekten zur Verfügung, die jeweils über eine x-, eine y-Koordinate und einen skalaren Wert verfügen.

Als zweite Eingabe benötigt der Algorithmus die Isowerte, welche als Array von `double`-Werten übergeben werden. Aus den Isowerten entstehen Wertebereiche (siehe Tabelle 5.2). Anhand dieser Wertebereiche und der Werte des Parameters wird das Raster in Flächen zerteilt, in denen der Parameter jeweils nur Werte eines Wertebereichs annimmt. Dabei werden bei der Erstellung der Flächen bzw. für den

Verlauf der Begrenzungslinien, nicht nur die Parameterwerte direkt an den Rasterpunkten berücksichtigt, sondern auch die Werte auf den Gitterkanten des Rasters durch lineare Interpolation hinzugezogen.

Isowerte	-5.0	0.0	5.0	
Wertebereiche	$-\infty$ .. -5.0	-5.0 .. 0.0	0.0 .. 5.0	5.0 .. $+\infty$
Füllindezes	0	1	2	3

Tabelle 5.2: Isowerte, Wertebereiche und Füllindizes

Die entstandenen Flächen, sie wurden im Abschnitt 4.2 als *Isoflächen* bezeichnet, werden von Isolinien und eventuell dem Rand des Rasters begrenzt. Die begrenzenden Isolinien können dabei zu zwei unterschiedlichen Isowerten gehören, da jede Isofläche die Parameterwerte eines Werte*bereichs* enthält, der von *zwei* Isowerten (bzw. $\pm\infty$) bestimmt wird (vgl. Abschnitt 4.2.3). Diese Eigenschaft der Isoflächen ist während der Implementierungsphase stets zu berücksichtigen.

Die Speicherung der Isoflächen in Java erfolgt durch `PolygonShape`-Objekte. Entlang der Isolinie bzw. des Rasterrandes werden solange neue Polygonpunkte eingesammelt, bis der Startpunkt des Polgyons erneut erreicht wird und sich ein geschlossener Linienzug ergibt. Den Füllindex des Objekts bestimmt der Wertebereich der repräsentierten Isofläche (siehe Tabelle 5.2).

Schwierigkeiten bei der Implementierung des Algorithmus machte nicht das Auffinden der Isowerte und das Verfolgen der Isolinien durch das Raster, sondern z.B. die korrekte Markierung der Gitterkanten, auf denen ein Isowert gefunden worden war, und die durchgängige Beschriftung der Isolinien.

Markierung der Gitterkanten

Das auf Snyders Artikel [SnyW1978] basierende *line following* verfolgt den Verlauf einer Isolinie anhand der Gitterkanten durch das rechteckige Raster. Alle Isowerte auf allen Gitterkanten müssen für die erfolgreiche Vektorisierung der Rasterdaten gefunden werden und jeder interpolierte Isopunkt muss Bestandteil einer Isolinie sein. Bereits gefundene Isowerte müssen dabei für die betroffene Gitterkante vermerkt werden, damit sie nicht noch einmal berücksichtigt werden.

Eine Isolinie trennt jedoch *zwei* Isoflächen voneinander. Beide Flächen beanspruchen sie als Begrenzungslinie. Deshalb muss es erlaubt sein, dass eine Isolinie genau zweimal verwendet wird - für zwei verschiedene Isoflächen.

Diese unterschiedliche Verwendung derselben Isolinie kann während der Programmausführung daran erkannt werden, dass eine Isolinie für die beiden beteiligten Flächen einmal die untere Grenze des Wertebereichs darstellt und einmal die obere.

Die Klasse `GridEdges` stellt einige Methoden zur Verfügung, die das Markieren von Gitterkanten eines `GridShape` während der Vektorisierung unterstützen.

Das geschilderte Phänomen tritt nur für Isoflächen auf, die mit dem Rand des Rasters in Berührung sind. Die Isolinien aller inneren Polygone werden nur einmal gefunden. Je nach Konfiguration des Vektorisierers werden sie jedoch anschließend für die Bildung eines „Donuts“ aus der umschließenden Fläche (siehe Abbildung 4.5 auf Seite 60) ein zweites Mal verwendet.

Beschriftung der Isolinien

Die Füllung einer Isofläche festzustellen, ist einfach, da sie „der Farbe“ bzw. dem Index des Wertebereichs entspricht (siehe Tabelle 5.2). Doch wie sind Bezeichnungen zu vergeben? Üblicherweise werden nicht die Flächen mit ihrem Wertebereich beschriftet, sondern die Isolinien (z.B. Isothermen, Isobaren), d.h. die Begrenzungslinien mit dem Wert des Parameters versehen.

Für die Speicherung der Isoflächen werden `PolygonShape`-Objekte verwendet, die sich anhand von `LabelPoint`-Objekten die Positionen von Beschriftungen merken. Ein `LabelPoint` gibt jedoch nur eine Position an. Die Art der Beschriftung bestimmt das Polygon selbst. Es wäre möglich gewesen, jedes `LabelPoint`-Objekt mit einer Zeichenkette für die Speicherung eines individuellen *Labels* auszustatten, doch als Bestandteil eines komplexen Grafikobjekts wäre diese Individualität der Labelpunkte kaum zu verwalten gewesen.

Da Beschriftungen direkt auf dem Rand des Rasters nicht sinnvoll erscheinen - sie ragen über den Rand der Rasterfläche hinaus und markieren eventuell keine Isopunkte - beschränkt sich das Problem auf die Abschnitte der Begrenzungslinie des Polygons innerhalb des Rasters. Dort wird das Polygon von Isolinien begrenzt, die zu zwei verschiedenen Isowerten gehören können. Trotzdem sieht ein `PolygonShape` nicht vor, zwischen zwei unterschiedlichen Beschriftungen bzw. zwischen zwei unterschiedlichen Gruppen von `LabelPoints` zu differenzieren. Stattdessen lautet die Regelung: Nur die Isolinie des niedrigeren Isowerts, der die untere Grenze des Wertebereichs bestimmt, wird beschriftet bzw. nur dort werden Labelpunkte gesetzt. Das Polygon erhält als Beschriftungsattribut den Wert des niedrigeren Isowerts, das später an den Positionen der Labelpunkte erscheint. Das hat den Vorteil, dass keine Linie aus Versehen doppelt mit Beschriftungen versehen wird.

Eine Ausnahme der Regelung gibt es allerdings. Sie tritt ein, wenn der Vektorisierer so konfiguriert wurde, dass innere Polygone nicht aus ihrer umgebenen Fläche herausgeschnitten werden. Die Isolinie, die ein inneres Polygon begrenzt,

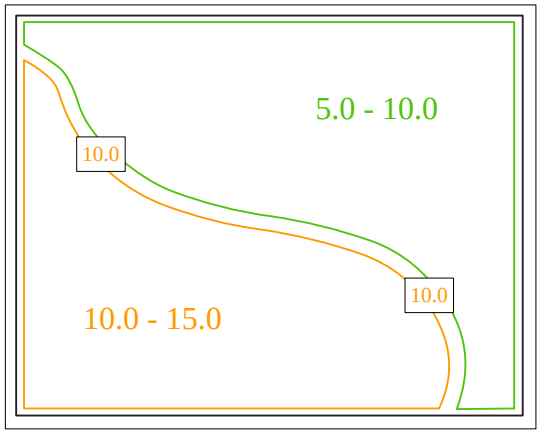
wird in diesem Fall nur einmal verwendet und nicht beschriftet, wenn es sich um die höherwertige Isolinie handelt. Hier muss die Ausnahme eintreten.

Abbildung 5.5 auf Seite 96 zeigt an einfachen Beispielen die Beschriftung der Isolinien für alle möglichen Fälle. Die letzte Grafik zeigt den Spezialfall, bei dem ausnahmsweise die höherwertige Isolinie beschriftet werden muss.

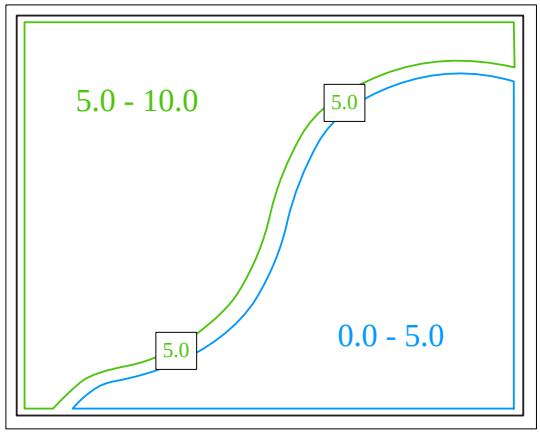
Verwendung der Klasse

Die Klasse `VectorizerImpl` besitzt nur den leeren Konstruktor (warum erklärt Abschnitt 5.6) und stellt folgende Methoden für die Verwendung des Vektorisierungs-Algorithmus zur Verfügung:

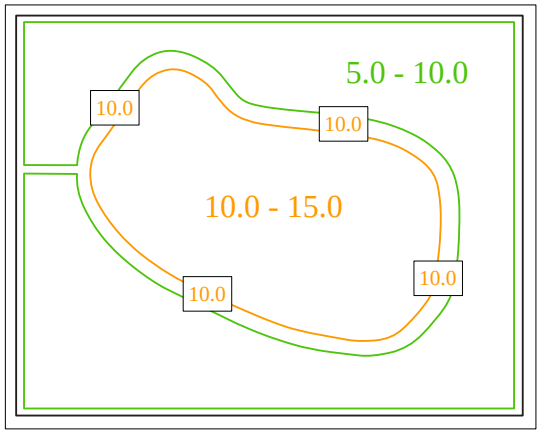
- `void setCutOutInternalPolygons( boolean cutout )`
  beeinflusst das Verhalten des Algorithmus beim Auftreten innerer Polygone. Innere Polygone sind Flächen, die keinen Kontakt mit dem Rand des Rasters haben. Sie werden von einer geschlossenen Isolinie begrenzt und beinhalten eventuell andere innere Polygone. Als Flächen betrachtet, liegen innere Polygone zuerst *auf* anderen Polygonen.
  Der boolesche Parameter `cutout` der Methode bestimmt, ob innere Polygone aus der Fläche, auf der sie liegen, herausgeschnitten werden oder nicht. Wird auf eine spätere Füllung der Flächen wert gelegt, ist der Vorgang zu aktivieren. Sind die eigentlichen Isolinien von Interesse, ist das Ausschneiden der inneren Polygone nicht durchzuführen. Siehe Abbildung 4.8 auf Seite 63.
- `void setLabelPositions( int boundarydistance, int distance )`
  konfiguriert, ob und wie oft Polygonpunkte als Positionen für Beschriftungen verwendet werden. Da Beschriftungen auf dem Rand des Rasters nicht sinnvoll sind, gibt der Parameter `boundarydistance` an, ab dem wievielten Punkt im Innern des Rasters (die Polygonlinie entfernt sich vom Rand) eine Beschriftung im Abstand von `distance` Punkten erfolgen soll. Ein Wert von 0 für `distance` deaktiviert die Beschriftung.
- `ShapeSet getContourPolygonShapeSet( GridShape grid, ... double[] contourvalues )`
  führt die eigentliche Vektorisierung durch. Entsprechend des rechteckigen Rasters `grid` und der Isowerte `contourvalues` liefert die Methode ein `ShapeSet`-Objekt, das die ermittelten Isoflächen als `PolygonShapes` enthält.



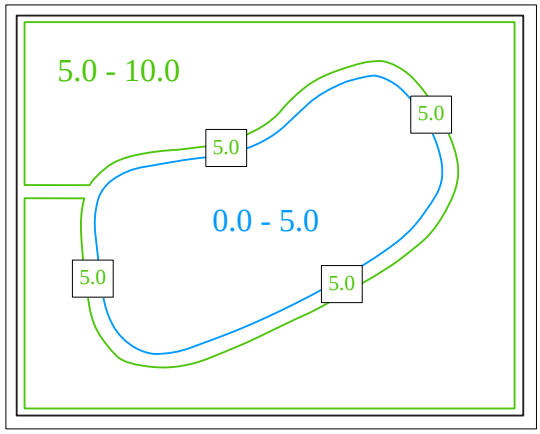
(a) Randpolygone I



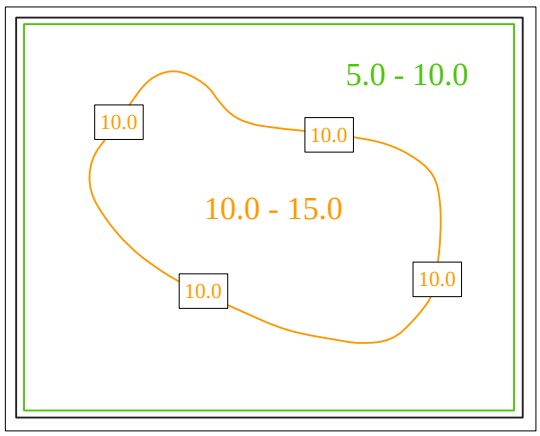
(b) Randpolygone II



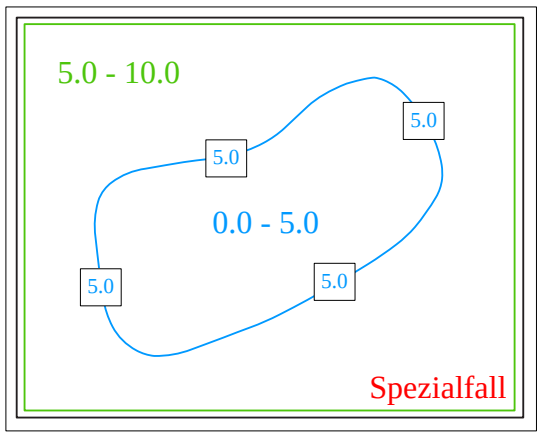
(c) innere Polygone mit Ausschneiden I



(d) innere Polygone mit Ausschneiden II



(e) innere Polygone ohne Ausschneiden I



(f) innere Polygone ohne Ausschneiden II

Abbildung 5.5: Beschriftung der Isolinien

5.5 Transformation der Java-Grafikobjekte

Zu Beginn dieses Kapitels wurde ein allgemeines XML-Format für Grafikobjekte und die dazugehörigen Klassen der Java-Grafikobjekte vorgestellt. Lesemodule für Shapefiles und GRIB-Dateien liefern Karten- und Wetterdaten als XML-Dateien oder direkt als `GridShape`-Objekt. Die weitere Verarbeitung der Daten kann nun anhand von Tranformationen, die die Java-Objekte selbst ermöglichen, oder mit Hilfe von *Transformationsklassen* erfolgen. Zu den Transformationsklassen sollen die Klassen zählen, die auf den Java-Grafikobjekten des Pakets `de.flashweather.graph2d.base` operieren, wie z.B. die Vektorisierungs-Klasse aus dem vorherigen Abschnitt.

Dieser Abschnitt stellt die Implementation weiterer Transformationen und Transformationsklassen vor.

5.5.1 Integrierte Transformationen

Elementare Transformationen

Die in Abschnitt 4.4 vorgestellten *elementaren Transformationen* Translation, Skalierung und Rotation beherrschen alle von `GraphicObject` abstammenden Klassen - also alle Java-Grafikobjekte. Die Skalierung und die Rotation erfolgen dabei bezüglich des Ursprungs des Koordinatensystems. Die zur Verfügung stehenden Methoden lauten im Einzelnen:

- `void translate(double xtranslate, double ytranslate)`
  verschiebt das Objekt entsprechend der Argumente in x- und y-Richtung.
- `void scale(double xscale, double yscale)`
  verändert den Abstand jedes Punkts vom Ursprung entsprechend der Skalierungsfaktoren `xscale` und `yscale`.
- `void rotate(double angle)`
  dreht das Objekt entgegen dem Uhrzeigersinn um `angle` Grad um den Ursprung des Koordinatensystems.
- `void transform(double a11, double a12, ..., double a33)`
  gehört nicht unbedingt zu den *elementaren* Transformationen, sondern führt eine allgemeine Transformation mit der 3x3-Transformationsmatrix durch, die durch die Argumente `a11` bis `a33` zeilenweise definiert wird.

Tatsächlich implementiert werden die genannten vier Methoden in der `Point`-Klasse. `LabelPoint` und `GridPoint` erben sie von der `Point`-Klasse und komplexe Grafik-

objekte führen die Methoden aus, indem sie sie auf jeden ihrer Definitions-, Raster- und Labelpunkte anwenden.

Generalisierung eines Rasters

Eine spezielle Transformation von Rasterdaten ist die *Generalisierung*. Für ein rechteckiges Raster fasst sie zeilen- und spaltenweise eine vorgegebene Anzahl Rasterpunkte zu einem einzigen neuen Punkt zusammen. Sowohl für die Koordinaten der Rasterpunkte als auch für die Parameterwerte an den Rasterpunkten wird der Mittelwert gebildet. Die Dimension des Rasters wird verkleinert, wie Abbildung 5.6 an einem Beispiel verdeutlicht.

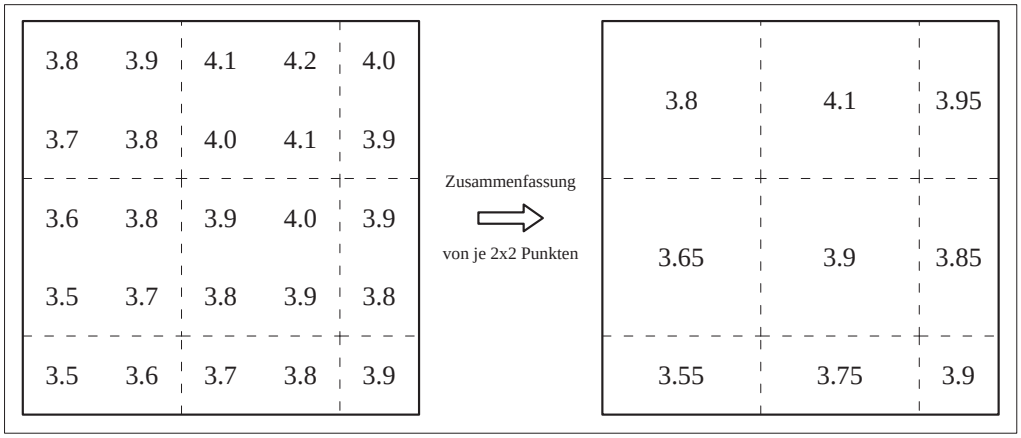


Abbildung 5.6: Generalisierung eines rechteckigen Rasters

Die Klasse `GridShape` verfügt bereits über eine Methode `getGeneralized(int col, int row)`, die ein generalisiertes `GridShape`-Objekt zurückliefert, bei dem jeweils die Rasterpunkte von `col` Spalten und `row` Zeilen des Ausgangsobjekts zu einem Wert zusammengefasst wurden. Das aktuelle Objekt bleibt dabei unverändert.

5.5.2 Externe Transformationsklassen

Die elementaren Transformationen und die Generalisierung eines Rasters sind direkt in den Klassen der Java-Grafikobjekte implementiert. Die folgenden Transformationen *Clipping* und *Projektion* existieren stattdessen in externen Klassen. Sie implementieren komplizierte Algorithmen in umfangreichen Methoden, die nicht mit dem Programmcode der Grafikobjekte vermischt werden sollen. Die Transformationsklassen werden erst bei Bedarf geladen und können leicht ausgewechselt werden, ohne Veränderungen an den Grafikobjekten selbst vorzunehmen.

Clipping

Alle in Abschnitt 4.4 vorgestellten Clipping-Algorithmen implementiert die Klasse `ClipperImpl` aus dem Paket `de.flashweather.graph2d.clip`. Zur Erzeugung eines Clipping-Objekts existiert nur der leere Konstruktor. Die Köpfe der Methoden sind durch das Interface `Clipper` vorgegeben, das eine allgemeine Schnittstelle für Clipping-Algorithmen im Zusammenhang mit den vorliegenden Java-Grafikobjekten spezifiziert. Die Methoden lauten (ohne die Koordinaten des rechteckigen Clipping-Fensters):

- `Shape clip( Shape shape, ...)`
  clippt ein komplexes Grafikobjekt an einem rechteckigen Fenster. Da `Shape` eine abstrakte Klasse ist, ermittelt diese Methode den tatsächlichen Typ von `shape` (eine Subklasse von `Shape`) und ruft die passende `clip`-Methode auf.
- `PointShape clip( PointShape point, ...)`
  gibt das Argument `point` wieder zurück, falls der Punkt im Innern des Clipping-Fensters liegt, `null` sonst.
- `ShapeSet clip( LineShape line, ...)`
  clippt eine Linie, die sich über mehrere Punkte erstrecken kann, an einem rechteckigen Fenster. Diese Methode liefert als einzige nicht den Objekttyp des übergebenen Grafikobjekts zurück, da die Linie beim Clipping in mehr als ein Teilstück zerfallen kann. Deshalb liefert die Methode alle Teilstücke als `LineShape`-Objekte in einem `ShapeSet` zurück.
- `PolygonShape clip( PolygonShape polygon, ...)`
  clippt ein Polygon an einem rechteckigen Fenster nach dem Algorithmus von Sutherland und Hodgeman [SuHo1974].
- `GridShape clip( GridShape grid, ...)`
  löscht alle Zeilen und Spalten des Rasters, die komplett außerhalb des Clipping-Fensters liegen - bis auf die jeweiligen (nicht-sichtbaren) Zeilen und Spalten direkt am Rand des Fensters, die noch bei einer Vektorisierung der Rasterdaten benötigt werden. Die Methode liefert das Resultat als neues `GridShape`-Objekt zurück.
- `ShapeSet clip( ShapeSet shapeset, ...)`
  ruft für alle Elemente des ShapeSet die entsprechende `clip`-Methode auf.

Keine der genannten Methoden verändert das als Argument übergebene komplexe Grafikobjekt. `clip(PointShape, ...)` liefert als einzige das ursprüngliche Grafikobjekt zurück, sofern es im Clipping-Bereich liegt - da es unverändert bleibt. Alle

anderen Methoden verwenden Kopien des Grafikobjekts für die Durchführung des Clipping. Diese Eigenschaft der Implementation kann man sich bei einer Kachelung der ursprünglichen Fläche (vgl. Abschnitt 4.5) zunutze machen.

Projektion

Das Java-Paket `de.flashweather.graph2d.project` enthält mehrere Interfaces und Klassen, die Kartenprojektionen und verwandte Transformationen zur Verfügung stellen. Die Klassenhierarchie ist in Abbildung 5.7 dargestellt.

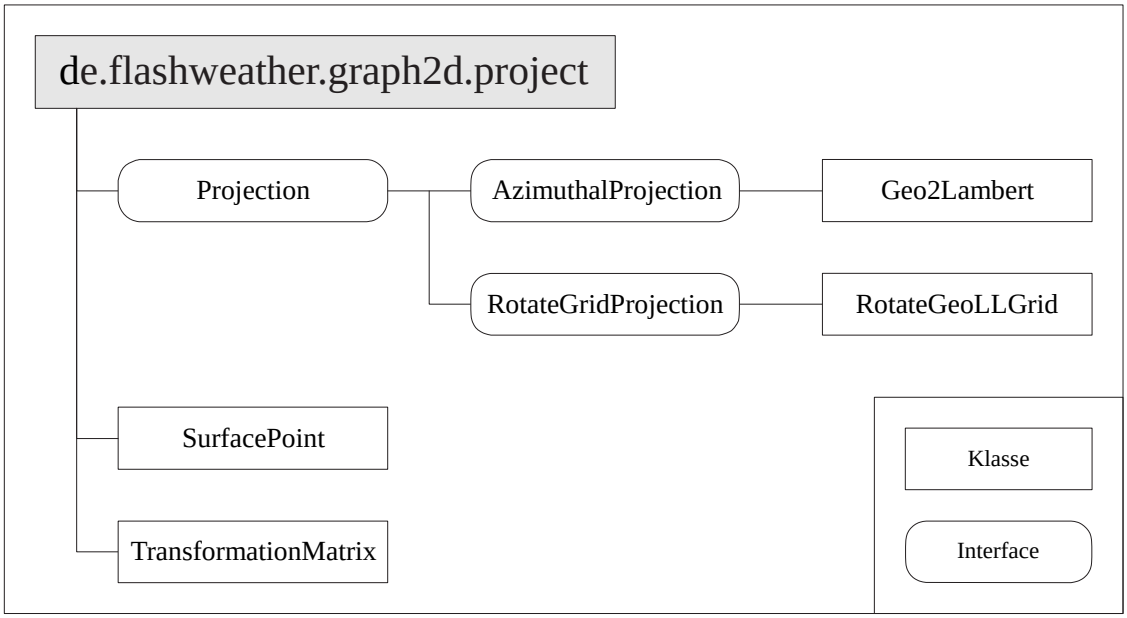


Abbildung 5.7: Klassenhierarchie der Projektions-Klassen und -Interfaces

Das Interface `Projection` definiert für alle Projektionsklassen die beiden Methoden:

- `void project(Point point)`
  wendet die Projektion auf einen Punkt an, der auch ein `LabelPoint` oder `GridPoint` sein kann.
- `void project(Shape shape)`
  wendet die Projektion auf alle Definitions-, Label- und Rasterpunkte eines komplexen Grafikobjekts an.

Das von `Projection` abstammende Interface `AzimuthalProjection` spezifiziert eine azimuthale Projektion, wie sie in Abschnitt 4.3 vorgestellt wurde. Die zusätzliche Methode `setReferencePoint(double x, double y)` setzt den Berührpunkt der Projektionsebene an die Kugel auf die Koordinaten `x`,`y`. Die Entwurfsachse der Projektion verläuft damit durch diesen Punkt und den Mittelpunkt der Kugel.

Im Fall der Klasse `Geo2Lambert`, die das `AzimuthalProjection`-Interface implementiert, handelt es sich bei den Koordinaten des Berührpunkts um geografische Koordinaten. Die `project`-Methoden der Klasse wandeln geografische Koordinaten in die Bildkoordinaten des flächentreuen azimuthalen Entwurfs von Lambert um (siehe Abbildung 4.13 auf Seite 68).

Das zweite von `Projection` abstammende Interface `RotateGridProjection` spezifiziert keine echte Projektion der Erdkugeloberfläche auf eine Projektionsebene, sondern eine eventuell vor einer Projektion notwendige Rotation des Längen- und Breitengradnetzes. Deshalb befindet sich das Interface und die Klasse `RotGeoLLGrid`, eine Implementation des Interface, im Paket der Projektionen, obwohl die beiden eine spezielle 3D-Transformation definieren und implementieren.

`RotateGridProjection` erweitert das `Projection`-Interface um drei Methoden: `addX`-, `addY`- und `addZRotation(double angle)`, mit denen die erforderlichen Rotationen eingestellt werden. Intern speichert sie die Klasse `RotGeoLLGrid` in einem `TransformationMatrix`-Objekt, einer 3x3-Matrix.

Während der Projektion wird für jeden Punkt als erstes ein `SurfacePoint` erzeugt. Der `SurfacePoint` repräsentiert denselben Punkt auf der Oberfläche der Erdkugel, verwendet aber nicht Längen- und Breitengrad zur Bestimmung der Position, sondern x-, y- und z-Koordinate. An diesem 3D-Repräsentanten des eigentlichen Punkts werden die Rotationen durchgeführt. Anschließend werden die drei Koordinaten wieder in die Punktkoordinaten, geografische Länge und Breite, umgerechnet und dem ursprünglichen Punkt zugewiesen. So erfolgt Punkt für Punkt die Drehung des Gradnetzes der Erde bzw. die Verschiebung des Grafikobjekts auf der Kugeloberfläche.

Die Implementierung der Rotations-„Projektion“ war nötig, da die Wetterdaten des DWD an Rasterpunkten definiert sind, deren Koordinaten die Position innerhalb eines gedrehten Längen- und Breitengradnetzes angeben. Das Gradnetz der Erde wurde so auf der Erdoberfläche positioniert, dass Äquator und Null-Meridian mitten durch Deutschland laufen. Das hat den Vorteil, dass die Schrittweite zwischen zwei benachbarten Rasterpunkten im gesamten Raster etwa gleich bleibt. Denn in der Nähe des Äquators verlaufen die Längengrade noch relativ parallel zueinander, was zur Folge hat, dass auch die Schrittweite zwischen zwei Rasterpunkten, von denen immer gleich viele je Längengrad definiert sind, etwa konstant bleibt.

Eine Umrechnung aller Rasterpunkte in die wahren Längen- und Breitengradkoordinaten muss erfolgen, bevor eine echte Projektion wie `Geo2Lambert` angewendet werden kann.

5.5.3 Transformationen im Überblick

Abbildung 5.8 zeigt noch einmal alle Transformationen und Transformationsklassen im Überblick.

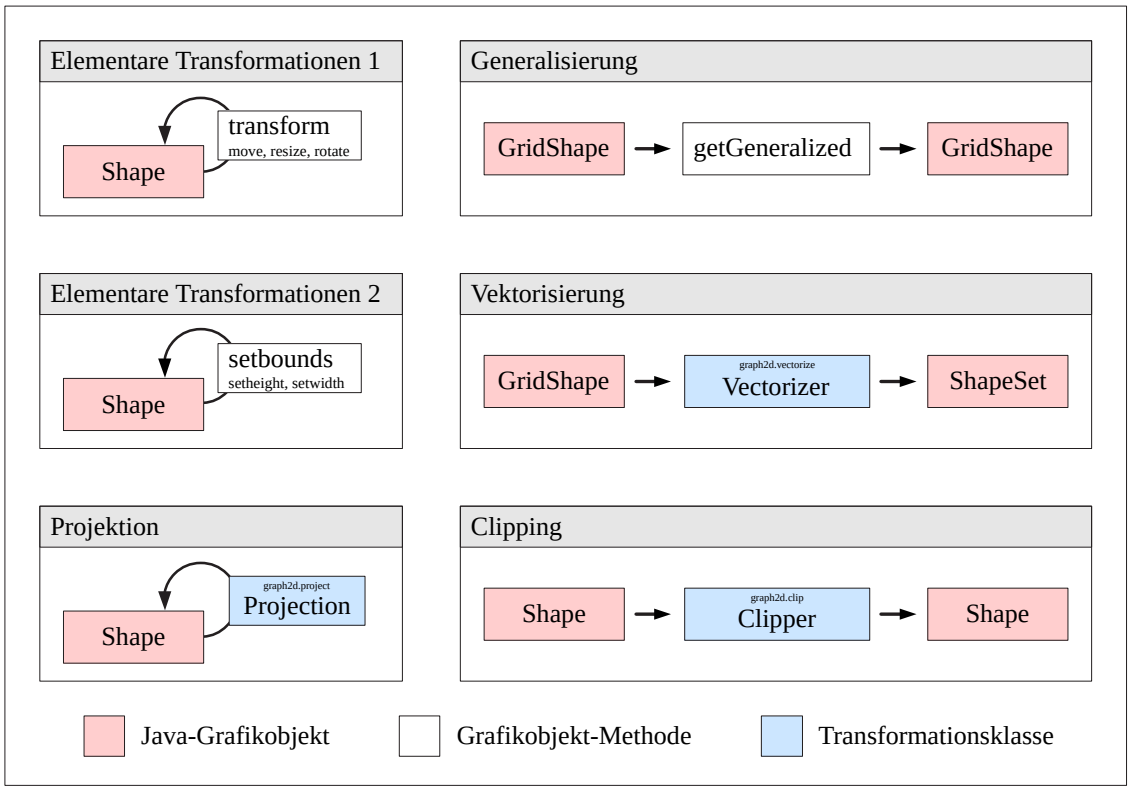


Abbildung 5.8: Transformationen und Transformationsklassen

5.6 Konfiguration und Übersetzung

Die vorherigen Abschnitte stellten Java-Klassen vor, die Grafikobjekte repräsentieren, Wetter- und Kartendaten auslesen und Transformationen auf den Grafikobjekten ausführen. Prinzipiell existieren nun alle Komponenten, die für die Erzeugung eines Flash-Films benötigt werden. Jedoch muss eine Konfiguration der Transformationen noch programmintern erfolgen. Eine Einschränkung, die sehr an das SWF-SDK von Macromedia erinnert, da jede Änderung eines Flash-Films nur durch die Anpassung des Programmcodes, der den Film erzeugt, erreicht werden kann.

Das fehlende Modul im Visualisierungsprozess (Abbildung 5.1 auf Seite 79) ist die externe *Konfiguration* der Transformationen und deren Reihenfolge. In diesem Abschnitt werden deshalb zwei weitere Java-Klassen vorgestellt, die eine Konfigurati-

onsdatei verarbeiten bzw. anhand ihres Inhalts die entsprechenden Transformationen ausführen.

5.6.1 Konfigurationsdatei

Die Konfiguration von nicht-interaktiven Kommandozeilenprogrammen erfolgt entweder über Parameter, die dem Programm beim Start mitgegeben werden, oder in einer Konfigurationsdatei. Die Verarbeitung der Wetter- und Kartendaten zu einem Flash-Film wird durch ein Programm erfolgen, dass aufgrund einer vorher festgelegten Konfiguration ohne weitere Benutzersteuerung ein Resultat in Dateiform liefert. Nur so kann ein Prozess unbeaufsichtigt alle 24 GRIB-Dateien für eine 24-Stunden-Vorhersage bearbeiten.

Da die Konfiguration des Visualisierungsprozesses recht umfangreich werden kann, erfolgt die Konfiguration in Konfigurationsdateien, deren Inhalt Anweisungen im XML-Format enthält und von einer DTD bestimmt wird. Das hat den Vorteil, dass fehlende obligatorische Elemente oder falsch benannte Parameter schon beim Einlesen dem XML-Parser auffallen. Die Korrektheit der Werte für die einzelnen Parameter (z.B. Gleitkommzahl größer Null, Parameter x größer Parameter y) kann jedoch erst zur Laufzeit erfolgen [16].

Die Klasse `ConfigDocument` im Paket `de.flashweather.io.util` ermöglicht das Abfragen der Elemente einer Konfigurationsdatei mit folgendem Aufbau:

```
<config>                                  Beginn der Konfiguration
    <parameter attr1="value1"             Parameter mit Attribut
               attr2="value2"             zweites Attribut
               ...            />          weitere Attribute
    ...                                   weitere Parameter
</config>                                 Ende der Konfiguration
```

Alle dem einleitenden Tag `config` untergeordneten Elemente sind leere Tags, die außer Attributen keinen Inhalt besitzen. Zum Auslesen des Inhalts einer solchen Konfigurationsdatei verfügt die Klasse `ConfigDocument` über folgende Methoden:

- `boolean hasNextElement()`
  liefert true, falls noch weitere Elemente folgen.
- `Element getNextElement()`
  liefert die nächste Konfigurationsanweisung als DOM-Element und macht es zum aktuellen Element. Die Methoden `getNodeName` und `getAttribute` des

[16] Erst in *XML Schema* sind Datentypen und Wertebereichsprüfungen vorgesehen.

DOM-Interface `Element` liefern dann den Namen des Parameters und die Werte der Attribute.

- `void reset()`
  springt an den Anfang der Konfigurationsdatei zurück.

Für die bisher implementierten Transformationen wurden Parameternamen gewählt und die benötigten Einstellungen in Attributen abgelegt. Je Transformation entsteht so ein XML-Tag, das in der Konfigurationsdatei verwendet wird und angibt, welche Transformation wann und mit welchen Einstellungen ausgeführt werden soll. Für die einzelnen Transformationen ergibt sich im Einzelnen:

1. Generalisierung

   ```
   <generalize row="Anzahl der Zeilen"
               col="Anzahl der Spalten"/>
   ```

   Die Attribute geben die Anzahl der Rasterpunkte je Zeile und Spalte an, die zu einem einzigen Rasterpunkt zusammengefasst werden. Die Standardwerte sind jeweils 1.

2. Veränderung der Parameterwerte des Rasters

   ```
   <changevalues add="Summand"
                 mult="Faktor"/>
   ```

   Zu jedem Parameterwert eines Rasterpunkts wird zuerst `add` addiert, bevor das Resultat mit `mult` multipliziert wird.

3. Vektorisierung

   ```
   <vectorize class="voller Name der Vektorisiererklasse"
              isovalues="Liste von Isowerten"
              cutoutinternalpolygons="true|false"
              boundarydistance="Index eines Polygonpunkts"
              labeldistance="Anzahl Punkte zwischen Labeln"/>
   ```

   Die angegebene Klasse muss das Interface `Vectorizer` implementieren. Die Isowerte sind beim Attribut `isovalues` durch Semikolon oder Leerzeichen zu trennen. Standardwert für `cutoutinternalpolygons` ist `true`, für `boundarydistance` und `labeldistance` ist dieser 0 (Beschriftung ist deaktiviert).

4. Projektion

   Azimuthale Projektion

```
<project class="voller Name der Projektionsklasse"
         type="azimuthal"
         refx="x-Koordinate des Referenzpunkts"
         refy="y-Koordinate des Referenzpunkts"/>
```

Drehung des Gradnetzes

```
<project class="voller Name der Projektionsklasse"
         type="rotategrid"
         rotaxis1="x|y|z erste Rotationsachse"
         rotangle1="Rotationswinkel in Grad"
         rotaxis2="x|y|z zweite Rotationsachse"
         rotangle2="Rotationswinkel in Grad"
         rotaxis3="x|y|z dritte Rotationsachse"
         rotangle3="Rotationswinkel in Grad"/>
```

Es existieren zwei unterschiedliche Projektionen, die sich in den verwendeten Attributen unterscheiden. Das Wert des Attributs `type` gibt an, um welchen Projektionstyp es sich handelt. Die angegebene Klasse muss das Interface `AzimuthalProjection` bzw. `RotateGridProjection` implementieren. Standardwert für die Koordinaten des Referenzpunkts und die Rotationswinkel ist 0.

5. Clipping

```
<clip class="voller Name der Clipping-Klasse"
      xmin="minimale x-Koordinate des Clipping-Fensters"
      ymin="minimale y-Koordinate des Clipping-Fensters"
      xmax="maximale x-Koordinate des Clipping-Fensters"
      ymax="maximale y-Koordinate des Clipping-Fensters"/>
```

Die angegebene Klasse muss das Interface `Clipper` implementieren.

6. Translation

```
<translate xtranslate="Verschiebung in x-Richtung"
           ytranslate="Verschiebung in y-Richtung"/>
```

Die Standardwerte sind 0.

7. Skalierung

```
<scale xscale="Skalierungsfaktor in x-Richtung"
       yscale="Skalierungsfaktor in y-Richtung"/>
```

Die Standardwerte sind 1.

8. Rotation

   `<rotate angle="`*Rotationswinkel in Grad*`"/>`

9. Allgemeine Transformation

   `<transform a11="`*Element 1,1 der Transformationsmatrix*`"`
   `a12="`*Element 1,2 der Transformationsmatrix*`"`
   `a13="`*Element 1,3 der Transformationsmatrix*`"`
   `a21="`*Element 2,1 der Transformationsmatrix*`"`
   `...`
   `a33="`*Element 3,3 der Transformationsmatrix*`"/>`

   Transformiert das Grafikobjekt anhand der gegebenen 3x3 Transformationsmatrix. Die Standardwerte ergeben eine Einheitsmatrix.

10. Ausdehnung

    `<setbounds xmin="`*minimale x-Koordinate der Ausdehnung*`"`
    `ymin="`*minimale y-Koordinate der Ausdehnung*`"`
    `xmax="`*maximale x-Koordinate der Ausdehnung*`"`
    `ymax="`*maximale y-Koordinate der Ausdehnung*`"/>`

    Die Gesamtheit aller Grafikobjekte wird so verschoben und skaliert, dass sie das durch die Koordinaten definerte rechteckige Fenster ausfüllt.

11. Höhe

    `<setheight height="`*Höhe der Ausdehnung aller Grafikobjekte*`"/>`

    Skaliert und verschiebt alle Grafikobjekte so, dass ihre gesamte Ausdehnung eine Höhe von `height` hat und im Ursprung beginnt.

12. Breite

    `<setwidth width="`*Breite der Ausdehnung aller Grafikobjekte*`"/>`

    Skaliert und verschiebt alle Grafikobjekte so, dass ihre gesamte Ausdehnung eine Breite von `width` hat und im Ursprung beginnt.

Die DTD `config.dtd` der Konfigurationsdateien ist im Anhang C vollständig aufgelistet.

Bei der Erstellung einer Konfigurationsdatei ist darauf zu achten, dass z.B. ein `generalize`-, `changevalues`- oder `vectorize`-Aufruf nur für Rasterdaten erfolgen kann und `vectorize` die Schnittstelle zwischen Raster- und Vektordaten bildet.

Wer aber verwaltet tatsächlich die Java-Grafikobjekte und startet die Transformationen, die die Konfigurationsdatei vorschreibt?

5.6.2 XML, Grafikobjekte und Transformationen

Die abstrakte Klasse `BaseShapeDocument` im Paket `de.flashweather.io.shape` ist die Basisklasse aller Klassen, die eine Datei repräsentieren, die in XML codierte Grafikobjekte enthält. Sie bildet die Schnittstelle zwischen verwendeter XML-Ausprägung und Java-Grafikobjekten bzw. Transformationen.

Abbildung 5.9 veranschaulicht die Arbeitsweise von `BaseShapeDocument` und einer Subklasse, die die noch abstrakten Methoden aus `BaseShapeDocument` für eine konkrete Anwendung implementiert.

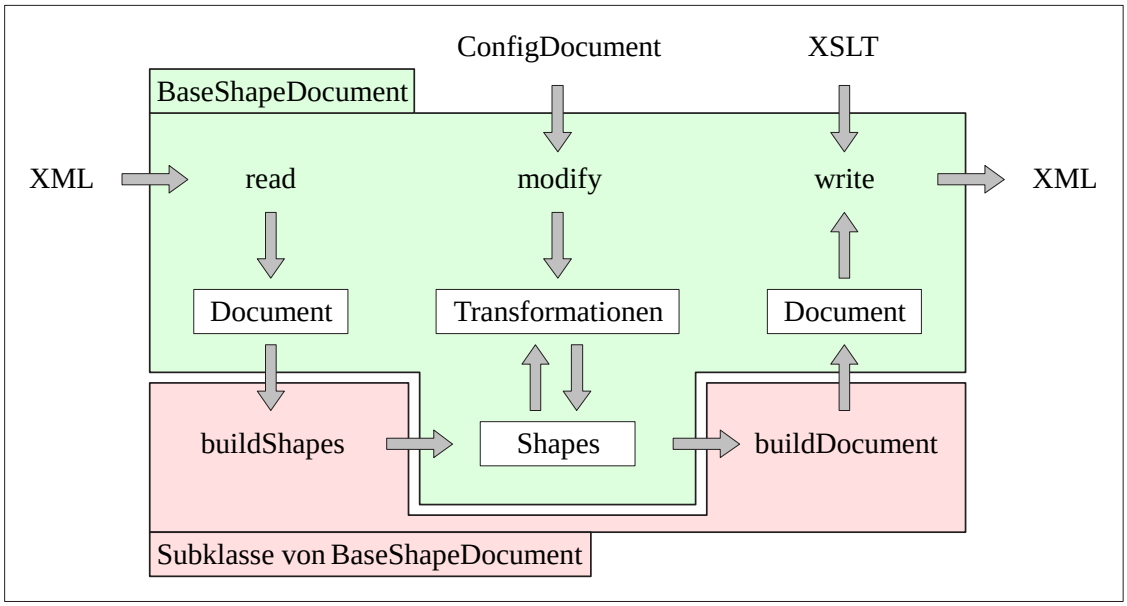


Abbildung 5.9: Die Arbeitsweise von `BaseShapeDocument`

`BaseShapeDocument` bietet eine Methode `read` zum Einlesen einer XML-Datei, die üblicherweise innerhalb eines Konstruktors aufgerufen wird. `read` startet einen DOM-Parser, der aus dem Inhalt der XML-Datei den korrespondierenden *Dokumentobjektbaum* erzeugt und ein *Dokumentobjekt* zurückliefert. Über das Dokumentobjekt vom Typ `Document` und den Schnittstellen des DOM können alle Baumelemente erreicht werden.

Der Dokumentobjektbaum enthält die Grafikobjekte aus der XML-Datei in Textform. Das tatsächliche Umsetzen der Baumobjekte in Java-Grafikobjekte muss in der Methode `buildShapes` außerhalb der Klasse `BaseShapeDocument` für jede XML-Ausprägung individuell erfolgen. Die Java-Grafikobjekte unterstützen das Einlesen und die Interpretation von XML-Teilbäumen, die Struktur und vor allem die Tag- und Attributnamen können jedoch variieren. `buildShapes` ist daher in `BaseShapeDocument` eine abstrakte Methode.

Das Kernstück der Klasse `BaseShapeDocument` ist die Methode `modify`. Sie erhält eine Konfigurationsdatei in Form eines `ConfigDocument`-Objekts und verarbeitet die Anweisungen zu Transformationen, die auf den Java-Grafikobjekten ausgeführt werden. Quellcode 5.4 zeigt einen Ausschnitt aus der Methode, in der für eine `clip`-Anweisung in der Konfigurationsdatei die über das Attribut `class` angegebene Klasse geladen wird und das Clipping durchgeführt wird. Hier wird klar, warum alle externen Transformationsklassen nur leere Konstruktoren besitzen: `Class.newInstance()` führt den leeren Konstruktor der durch `Class` referenzierten Klasse aus.

Sind alle Transformationen durchgeführt, folgt die Methode `buildDocument`, die ein Pendant zur Methode `buildShapes` darstellt. Sie wandelt die Java-Grafikobjekte wieder in einen XML-Objektbaum um und kann deshalb für eine konkrete Anwendung ebenfalls erst in einer Subklasse implementiert werden.

Abschließend ist die Methode `write` für die Ausgabe des erzeugten Dokumentobjektbaums zuständig. `write` kann neben dem Ausgabestrom als zweites Argument ein XSLT-Stylesheet erhalten. Dies führt dazu, dass noch vor der Ausgabe ein XSLT-Prozessor den Inhalt und die Struktur des Objektbaums anhand der Anweisungen des Stylesheets verändert. Der Ergebnisbaum wird dann im XML-Format in den Ausgabestrom geschrieben.

Nach der Transformation der Grafikobjekte ist also eine Transformation der verwendeten XML-Ausprägung ebenfalls möglich. Von `BaseShapeDocument` abstammende Klassen können so aus

- einer Eingabedatei mit in XML codierten Grafikobjekten
- einer Konfigurationsdatei für die Transformationen der Grafikobjekte
- einem XSLT-Stylesheet

in einem Durchgang die für einen Flash-Film benötigte SWFML-Datei erstellen.

5.7 Erzeugung von Flash-Filmen

Der Flash-Generator Saxess Wave, der im Rahmen dieser Arbeit zur Erzeugung der Flash-Filme verwendet wird, wurde in Kapitel 3.7 vorgestellt. Ebenfalls wurde bereits erwähnt, dass sich ein Flash-Film aus einer SWFML-Datei mit dem Kommandozeilenaufruf

```
java com.saxess.visweb.swfio.Driver in.swfml out.swf
```

erzeugen lässt. Eine einzelne SWFML-Datei bestimmt also den Inhalt eines Flash-Films.

```
package de.flashweather.io.shape;

import ...

public abstract class BaseShapeDocument {

   protected Shape shape;

   ...

   public void modify( ConfigDocument config ) throws ... {

      while ( config.hasNextElement() ) {

         // get line from configuration file
         Element confline = config.getNextElement();
         String category  = confline.getNodeName();

         if ( category.compareTo("clip") == 0 ) {

            // instantiate clipping class
            String  classname = confline.getAttribute("class");
            Class   clipclass = Class.forName( classname );
            Clipper clipper   = (Clipper) clipclass.newInstance();

            // get attribute values as doubles
            double xmin = XMLUtil.parseInt( confline, "xmin");
            double ymin = XMLUtil.parseInt( confline, "ymin");
            double xmax = XMLUtil.parseInt( confline, "xmax");
            double ymax = XMLUtil.parseInt( confline, "ymax");

            // wrong arguments?
            if ( xmin > xmax || ymin > ymax )
            throw new IllegalArgumentException("Wrong clip window" +
                      " definition: xmin > xmax || ymin > ymax");

            // clip
            this.shape = clipper.clip(this.shape,xmin,ymin,xmax,ymax);
            continue;
         }

         ...
      }
   }

   ...
}
```

Quellcode 5.4: Methode `modify` der Klasse `BaseShapeDocument`

Vom Deutschen Wetterdienst stehen allerdings je Tag 24 GRIB-Dateien mit Stundenprognosen zur Verfügung, die in einem animierten Wetterfilm nacheinander angezeigt werden sollen. Da aber aus jeder einzelnen GRIB-Datei schließlich eine SWFML-Datei entsteht, muss aus allen 24 SWFML-Dateien der Wetterfilm zusammengestellt werden.

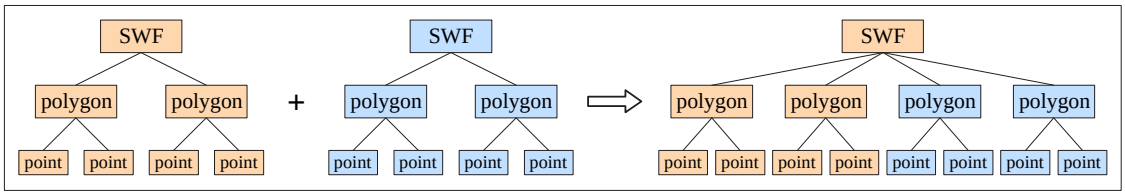


Abbildung 5.10: Verschmelzen von SWFML-Dateien

Das ist möglich, indem die SWFML-Dateien wie in Abbildung 5.10 dargestellt über die resultierenden Dokumentobjektbäume zu einem einzigen Objektbaum bzw. einer einzigen Datei verschmolzen werden, die dann dem Flash Generator übergeben wird. Dabei ist es nicht einmal nötig, den Objektbaum aller SWFML-Dateien tatsächlich wieder in eine Datei zu schreiben, da die Klasse `Driver` von Saxess Wave daraus über folgende Java-Aufrufe direkt einen Flash-Film erzeugt:

```
import com.saxess.visweb.swfio.Driver;
...
Document doc = Dokumentobjektbaum aller SWFML-Dateien;
Driver swfdriver = new Driver( doc );
swfdriver.parse( Dateiname des Flash-Films );
```

Die selbst geschriebene Klasse, die eine beliebige Anzahl von SWFML-Dateien in einen einzigen Objektbaum einhängt und daraus mit Saxess Wave einen Flash-Film generiert, heißt `Swfml2Swf`. Ihr Aufruf lautet:

```
java Swfml2Swf in.swfml [in.swfml ...] out.swf
```

6 Anwendung

Die bisher möglichst allgemein gehaltene Implementation der Java-Grafikobjekte und der Transformationsklassen bildet eine Bibliothek von Java-Klassen zur Bearbeitung von in XML codierten zweidimensionalen Grafikobjekten. Diese *Flash-Weather-Bibliothek* soll nun zur Erzeugung von Flash-Filmen aus den Wettervorhersagedaten des Deutschen Wetterdienstes angewendet werden.

6.1 Benötigte Hard- und Software

Benötigte Hardware

Die Erzeugung der Flash-Filme ist wenig prozessor- aber sehr speicherintensiv, wie die Testläufe in Abschnitt 6.4 zeigen werden. Dabei bestimmen der Umfang der Eingabedaten und die Konfiguration des Visualisierungsvorgangs, wieviel Speicher tatsächlich benötigt wird: je höher die Auflösung der Daten, desto höher der Speicherbedarf.

Neben den Java-Grafikobjekten ist vor allem die verwendete DOM-Technologie, die XML-Dateien als Objektbäume im Speicher hält, für den hohen Bedarf an Speicher verantwortlich. Denn jeder Punkt, der in einer XML-Datei als Tag definiert ist, erweitert den entsprechenden Dokumentobjektbaum um drei `Node`-Objekte (Punkt, x-Attribut, y-Attribut).

Für die Erzeugung eines 170 KB großen Flash-Films, der für ein Deutschland umfassendes Gebiet für jede volle Stunde eines Tages die Wetterdaten und im Hintergrund die Landesgrenze, Flüsse, Seen und Städte zeigt, werden 50 MB Arbeitsspeicher benötigt.

Benötigte Software

Da Sun Microsystems das *Java Development Kit* (JDK) für Linux, Solaris und Windows implementiert und es auf fast alle anderen Plattformen portiert wurde, ist kein spezielles Betriebssystem erforderlich. Die Verwendung der Java2D-API durch Saxess Wave setzt allerdings die Existenz einer grafischen Oberfläche (X11, Windows) voraus.

Folgende Software-Pakete werden für die Erzeugung von Flash-Filmen mit den FlashWeather-Klassen benötigt und sind kostenlos im Internet verfügbar (siehe Anhang A):

- **Sun Java 2 SDK 1.2.2 oder höher** (Java Software Development Kit)
  Saxess Wave baut auf der Java2D-API des Java 2 SDK auf und auch die FlashWeather-Klassen benutzen einige Methoden, die erst ab Version 1.2.2 implementiert sind.
- **Apache Xerces Java Parser 1.1.3 oder höher** (XML-Parser)
  Die Methoden der DOM Level 2 Spezifikation werden von den FlashWeather-Klassen verwendet.
- **Apache Xalan-Java 1.2.D01 oder höher** (XSLT-Prozessor)
  Die Version des XSLT-Prozessors muss zum verwendeten XML-Parser passen.
- **Saxess Wave 0.5 oder höher** (Flash-Generator)

Bis auf das JDK handelt es sich bei den Software-Paketen lediglich um JAR-Dateien, die der `CLASSPATH`-Variablen der Laufzeitumgebung hinzugefügt werden müssen.

Für die Umwandlung von ESRI Shapefiles in MapML-Dateien werden zusätzlich benötigt:

- **shp2mapml** (C-Programm)
- **C-Bibliothek shapefile**

Eventuell müssen C-Programm und Bibliothek für die aktuelle Plattform erst kompiliert werden. Makefiles erleichtern diesen Vorgang. Zur Ausführung von `shp2mapml` unter Linux muss die Umgebungsvariable `LD_LIBRARY_PATH` auf das Verzeichnis der *Shared Library* `libshp.so` verweisen, in der sich die shapefile-Bibliothek befindet.

6.2 Karten- und Wetterdaten

Der Einstieg in die Verwendung der FlashWeather-Klassen erfolgt über die Klasse `BaseShapeDocument`, die die Schnittstelle zwischen XML-Dokument und Java-Grafikobjekten herstellt (vgl. Abbildung 5.9 auf Seite 107). Je XML-Dokumenttyp ist eine individuelle Subklasse von `BaseShapeDocument` zu bilden, die die bisher abstrakten Methoden `buildShapes` und `buildDocument` implementiert.

6.2.1 MapDocument

Das Ausgabeformat des Lesemoduls für ESRI Shapefiles wurde MapML genannt, da es die Elemente einer Landkarte (Städte, Flüsse, Seen und Länderumrisse) in einer XML-Ausprägung speichert. Die Subklasse von `BaseShapeDocument`, die dieses XML-Format in Java-Grafikobjekte überführt und umgekehrt die Grafikobjekte wieder in einen Dokumentobjektbaum verwandelt, heißt deshalb `MapDocument`.

`MapDocument` stellt drei Konstruktoren zur Verfügung, die alle das Ziel haben, aus einer XML-Quelle die entsprechenden Java-Grafikobjekte zu erzeugen. Ein String wird als *URI* (Uniform Resource Identifier) interpretiert, ein `InputSource`-Objekt als Eingabestrom behandelt und ein Dokumentobjekt direkt verarbeitet.

```
import de.flashweather.io.shape.BaseShapeDocument;
import ...

public class MapDocument extends BaseShapeDocument {

   public MapDocument( String uri ) throws ... {
      this( new InputSource( uri ) );
   }

   public MapDocument( InputSource in ) throws ... {
      this( this.read( in ) );
   }

   public MapDocument( Document doc ) {
      this.shape = this.buildShapes( doc );
   }
```

Die Methode `read` stammt aus der Superklasse und erstellt mit Hilfe eines *validierenden* DOM-Parsers einen Dokumentobjektbaum aus den Daten des Eingabestroms. Danach ist sichergestellt, dass das ursprüngliche XML-Dokument seine Dokumenttyp-Definition (DTD) erfüllt und ein *gültiges* Dokument ist. Seine Struktur und die verwendeten Tagnamen entsprechen den Angaben in der DTD.

Die Methode `buildShapes` in Quellcode 6.1 geht deshalb davon aus, dass das Argument ein gültiges Dokument liefert und überprüft nur noch den Namen des Dokumenttyps auf „map“. Die weiteren Anweisungen setzen nun die in der MapML-DTD definierte Struktur des Dokuments voraus. (Insofern sollte der dritte Konstruktor stets nur mit einem Dokumentobjektbaum als Argument aufgerufen werden, den ein validierender XML-Parser erstellt hat.)

Im weiteren Verlauf erzeugt die Methode ein `ShapeSet`-Objekt und verarbeitet alle Elemente des MapML-Dokuments. Jedes `part`-Element wird entsprechend dem Wert seines Attributs „type“ als `PointShape`-, `LineShape`- oder `PolygonShape`-Objekt in den `ShapeSet` eingefügt. Alle Tags, die nicht den Namen „part“ haben, werden als Info-Knoten für die spätere Ausgabe aufbewahrt.

Das Gegenstück, die Umwandlung der `Shape`-Objekte in einen MapML-Objektbaum, implementiert die Methode `buildDocument` in Quellcode 6.2. Sie verwendet die Klasse `DocumentImpl` (Apaches Implementation des `Document`-Interface)

```
protected Shape buildShapes( Document doc ) throws DOMException {

  String docname = doc.getDoctype().getName();
  if ( docname.compareTo("map") != 0 )        // check doctype
     throw new DOMException( ... );

  ShapeSet.XML_TAG = "map";                   // start new set of shapes
  ShapeSet shapeset = new ShapeSet();

  Element root   = doc.getDocumentElement();  // get root element
  String maptype = root.getAttribute("type"); // store maptype attribute
  shapeset.setAttribute("type", maptype );

  NodeList pnodes = root.getChildNodes();     // process all child nodes
  for (int i=0; i<pnodes.getLength(); i++) {  // of "map" root element

    Node pnode = pnodes.item( i );            // found a "part" element?
    if ( ( pnode.getNodeType() != Node.ELEMENT_NODE ) ||
         ( pnode.getNodeName().compareTo("part") != 0 ) ) {
      shapeset.addInfoNode( pnode );  continue;
    }
                                              // found a "part" element
    Element shapeelement = (Element) pnode;   // -> build shape
    String type = shapeelement.getAttribute("type");
    if ( type.compareTo("point") == 0 )
      shapeset.addShape( new PointShape( shapeelement ) );
    else if ( type.compareTo("line") == 0 )
      shapeset.addShape( new LineShape( shapeelement ) );
    else if ( type.compareTo("polygon") == 0 )
      shapeset.addShape( new PolygonShape( shapeelement ) );
  }
  return shapeset;
}
```

Quellcode 6.1: Methode `buildShapes` der Klasse `MapDocument`

```
protected Document buildDocument( Shape shape ) {

  DocumentImpl doc = new DocumentImpl();      // create new document
  DocumentType doctype =
    doc.createDocumentType("MapML",null,"map.dtd");
  doc.appendChild( doctype );                 // append doctype element

  shape.serialize( doc );                     // serialize shapes
  return doc;
}
```

Quellcode 6.2: Methode `buildDocument` der Klasse `MapDocument`

als neues Dokumentobjekt, definiert den Dokumenttyp des Dokuments und ruft die `serialize`-Methode des `ShapeSet`-Objekts auf. Da der `ShapeSet` alle Grafikobjekte bzw. Kartenelemente enthält und als Container die `serialize`-Methode an seine Elemente weiterreicht, erstellen die Java-Grafikobjekte auf diese Weise selbstständig den Dokumentobjektbaum.

6.2.2 WeatherDocument

Eine zweite Subklasse von `BaseShapeDocument` wird für die Wetterdaten des DWD benötigt. Da es sich um Wetterprognosedaten handelt, heißt sie `WeatherDocument`.

Die Implementation der bisher abstrakten Methoden `buildShapes` und `buildDocument` ist analog zur Klasse `MapDocument` durchzuführen. Einen gravierenden Unterschied zwischen beiden Klassen stellt lediglich das Ausgangsformat der Eingabedaten dar. Die Kartendaten können von `MapDocument` nur verarbeitet werden, wenn sie im MapML-Format vorliegen. Die Wetterprognosedaten des DWD befinden sich im Gegensatz dazu in GRIB-Dateien, die zwar über das in Abschnitt 5.3.2 vorgestellte Lesemodul ausgelesen werden können, jedoch anschließend durch ein `GribRecord`- bzw. `GridShape`-Objekt repräsentiert werden.

Dieser Unterschied hat zur Folge, dass in der Klasse `WeatherDocument` ein weiterer Konstrukor implementiert werden muss, der die Eingabedaten in Form eines `GribRecord`-Objekts erhält. Quellcode 6.3 zeigt den neuen Konstruktor, der die eigentlichen Rasterdaten ermittelt und das `GridShape`-Objekt mit weiteren Attributen versieht.

Um sicherzustellen, dass vor einem Aufruf der `write`-Methoden die Rasterdaten in Vektordaten umgewandelt wurden, wird außerdem die Methode `modify` aus `BaseShapeDocument` überschrieben (siehe Quellcode 6.4). Die neue Methode ruft die ursprüngliche `modify`-Methode der Superklasse auf und überprüft anschließend den Resultatwert. Eine Vektorisierung der Rasterdaten ist notwendig, um eine Ausgabe der Wetterdaten im folgenden XML-Format zu ermöglichen, das den Namen *WeatherML* erhielt und die ermittelten Isoflächen speichert:

```
<weather id="..."                    Wetter-Dokument
         type="..."                  Typ/Code des Parameters
         label="..."                 Name des Parameters
         level="..."                 Höhen- oder Druckangabe
         unit="...">                 Name der Größeneinheit
    <date ... />                     Datum (Infoknoten)
    <time ... />                     Zeit (Infoknoten)
```

```
public WeatherDocument( GribRecord grib ) {

  // configure GridPoint classes
  GridPoint.XML_TAG    = "point";

  // get grid of weather data as GridShape object
  this.shape = grib.getGridShape();

  // set attributes of GridShape
  this.shape.setAttribute("type",  grib.getType() );
  this.shape.setAttribute("label", grib.getDescription() );
  this.shape.setAttribute("unit",  grib.getUnit() );
  this.shape.setAttribute("level", grib.getLevel() );

  // add date and time as infonodes to GridShape
  this.shape.addInfoNode( ... );
  this.shape.addInfoNode( ... );
}
```

Quellcode 6.3: Konstruktor in der Klasse `WeatherDocument`

```
public void modify( ConfigDocument config ) throws ... {

  // configure two Shape classes
  PolygonShape.XML_TAG = "area";
  ShapeSet.XML_TAG     = "weather";

  // call metod of super class
  super.modify( config );

  // did not get a result shape?
  if ( this.shape == null )
    throw new IllegalStateException( ... );

  // did not get a ShapeSet as result?
  // -> configuration is incorrect (<vectorize> missing?)
  if ( ! ( this.shape instanceof ShapeSet ) )
    throw new IllegalStateException( ... );

  // add id attributes to shapes
  ShapeSet shapeset = (ShapeSet) this.shape;
  String shapesetid = shapeset.getAttribute("id");

  for (int i=0; i<shapeset.getLength(); i++)
    shapeset.getShape(i).setAttribute("id", shapesetid + "area" + (i+1));
}
```

Quellcode 6.4: Methode `modify` der Klasse `WeatherDocument`

```
    <area id="..."                          eine Isofläche
          label="..."                       Beschriftung der Umrisslinie
          fill="...">                       Füllindex
        <point x=".." y=".." />             Punkt des Umrisses
        ...                                 weitere Umrisspunkte
        <labelpoint x=".." y=".." />        Punkt für eine Beschriftung
        ...                                 weitere Labelpunkte
    </area>                                 Ende der Isofläche
    ...                                     weitere Isoflächen
</weather>                                  Ende des Wetter-Dokuments
```

Das XML-Format WeatherML kann gegebenenfalls erneut von der Klasse `WeatherDocument` eingelesen und weiterverarbeitet werden. Es stellt das Pendant zu MapML dar und die Methoden `buildShapes` und `buildDocument` wurden entsprechend umgesetzt. Abbildung 6.1 stellt die möglichen Datenformate der Karten- und Wetterdaten während des Visualisierungsprozesses gegenüber.

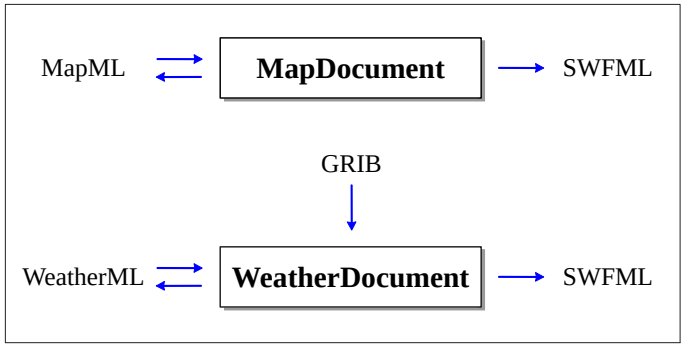


Abbildung 6.1: Karten- und Wetterdatenformate

6.3 Erzeugung eines Wetterfilms

Nachdem nun zwei spezielle Subklassen von `BaseShapeDocument` zur Verfügung stehen, die für die Karten- und Wetterdaten die konkrete Schnittstelle zwischen XML-Datei und Java-Grafikobjekten bilden, fehlen für die Erzeugung der Wetterfilme nur noch wenige Komponenten: Für Karten- und Wetterdaten werden jeweils

- eine Konfigurationsdatei, die die notwendigen bzw. gewünschten Transformationen angibt
- ein XSLT-Stylesheet für die Übersetzung von MapML bzw. WeatherML nach SWFML

- eine Java-Applikationen, die alle vorhandenen Elemente verknüpft

benötigt. Die nächsten Abschnitte erstellen Schritt für Schritt den Flash-Wetterfilm.

6.3.1 Konfiguration

Wetterdaten

Die Konfigurationsdatei für die Transformationen der DWD-Wetterprognosedaten enthält die folgenden Angaben:

1. Generalisierung des Rasters

   ```
   <generalize cols="3"  rows="3" />
   ```

   Fasst jeweils 3x3 Rasterpunkte zu einem einzelnen Punkt zusammen, um die Anzahl der Rasterpunkte und damit Auflösung der Daten zu verringern. Hier kann die Größe des resultierenden Flash-Films entschieden beeinflusst werden.

2. Rotation des Gradnetzes

   ```
   <project    class="de.flashweather.graph2d.project.RotateGeoLLGrid"
               type="rotategrid"
               rotaxis1="y"  rotangle1="-57.5"
               rotaxis2="z"  rotangle2="-10.0" />
   ```

   Rotiert das Netz der Längen- und Breitengrade um 57,5° um die y-Achse und um 10,0° um die z-Achse. Der Schnittpunkt von Äquator und Nullmeridian liegt dabei auf der x-Achse (vgl. Abbildung 4.11 auf Seite 66).

   Die Positionen der Rasterpunkte sind ursprünglich bezüglich eines rotierten Längen- und Breitengradnetzes definiert und werden in diesem Schritt zu geografischen Koordinaten konvertiert.

3. Lambert-Projektion

   ```
   <project    class="de.flashweather.graph2d.project.Geo2Lambert"
               type="azimuthal"
               refx="10.0"  refy="50.0" />
   ```

   Projeziert alle Rasterpunkte (Punkte auf der Erdoberfläche) mit Hilfe der azimuthalen Lambert-Projektion in die Ebene.

4. Skalierung mit Spiegelung

   ```
   <scale      xscale="+4200.0"  yscale="-4200.0" />
   ```

Ein negativer Wert für die Skalierung in y-Richtung bewirkt eine vertikale Spiegelung des Rasters, die notwendig ist, da die geografischen Breitengrad-Koordinaten in Richtung des Nordpols (nach oben) ansteigen, die y-Koordinatenachse des Flash-Films aber nach unten zeigt. Betragsmäßig identische Skalierungswerte in x- und y-Richtung erhalten das Größenverhältnis von Breite zu Höhe.

5. Verschiebung

```
<translate xtranslate="380"  ytranslate="386" />
```

Diese Anweisung verschiebt die Grafikobjekte nach rechts und unten, da sich der Ursprung des Flash-Films in der linken oberen Ecke befindet.

Die Werte für Skalierung und Translation der Grafikobjekte können berechnet werden, werden aber am einfachsten durch Ausprobieren anhand der Hintergrundelemente ermittelt. Ist z.B. der Umriss von Deutschland korrekt in der Mitte des Flash-Films platziert worden, können bzw. müssen die Werte für die Wetterdaten übernommen werden.

6. Clipping I

```
<clip       class="de.flashweather.graph2d.clip.ClipperImpl"
            xmin="0"  xmax="800"
            ymin="0"  ymax="600" />
```

Ein Clipping der Rasterdaten verkleinert das Raster und verringert damit den Aufwand für die nachfolgende Vektorisierung. (Der Clipping-Algorithmus eliminiert prinzipiell alle Spalten und Zeilen des Rasters, die komplett außerhalb des Clipping-Fensters liegen. Jeweils eine nicht-sichtbare Zeile und Spalte an den Rändern des Clipping-Fensters wird jedoch verschont, damit das Raster immernoch die gesamte Fläche des Fensters bedeckt.)

7. Vektorisierung

```
<vectorize  class="de.flashweather.graph2d.vectorize.VectorizerImpl"
            isovalues="0.1 0.3 0.5 0.7 0.9"
            cutoutinternalpolygons="true" />
```

Die Vektorisierung ist die einzige Transformation die für jeden Parametertyp der Wetterdaten unterschiedlich konfiguriert werden muss. Die Isowerte sind dem Wertebereich des Parameters entsprechend zu formulieren und das Ausschneiden der Polygonflächen hängt genauso wie die Beschriftung der Isoflächen von dem gewünschten Layout des Flash-Films ab.

Das Beispiel stammt aus der Konfigurationsdatei für die Bewölkungsvorhersage, die den Grad der Bedeckung in Prozent angibt. Beschriftungspunkte werden nicht ermittelt.

8. Clipping II

```
<clip       class="de.flashweather.graph2d.clip.ClipperImpl"
            xmin="0"  xmax="800"
            ymin="0"  ymax="600" />
```

Das zweite Clipping ist ein Polygon-Clipping und schneidet die aus den Rasterdaten erzeugten Vektorgrafikobjekte auf die gewünschte rechteckige Größe des Flash-Films zurecht. Die Transformation der Wetterdaten ist damit abgeschlossen.

Kartendaten

Die entsprechende Konfigurationsdatei für die Transformation der vorliegenden Kartendaten enthält nur die Punkte 3, 4, 5 und eventell Punkt 8. Eine Generalisierung (Punkt 1) ist genauso wie eine Vektorisierung (Punkt 7) nicht möglich, da die MapML-Daten bereits Vektorgrafikobjekte beschreiben. Eine Rotation des zugrundeliegenden Gradnetzes (Punkt 2) muss nicht durchgeführt werden, da die Definitionspunkte der Grafikobjekte bereits in geografischen Koordinaten angegeben sind. Das Clipping (Punkt 8) kann eventuell entfallen, wenn die Kartenelemente komplett übernommen werden sollen (z.B. gesamte Deutschlandkarte), es muss auf jeden Fall nur einmal durchgeführt werden.

6.3.2 XSLT-Stylesheet

Die Hauptaufgabe der Konfigurationsdatei ist es, die Wetter- und Kartendaten für die Visualisierung soweit vorzubereiten, dass die Grafikobjekte eines gewünschten Ausschnitts in Koordinatenangaben vorliegen, die direkt als Pixel eines Flash-Films verwendet werden können. Das XSLT-Stylesheet bestimmt nun noch:

- die Breite und Höhe des Flash-Films, die mit der Größe des letzten Clipping-Fensters übereinstimmen sollten
- die Abspielrate des Films in Bildern (Frames) pro Sekunde
- Füllfarben (Farbwert = RGB-Wert + Transparenzwert)
- Linienfarben und -dicken
- Textfarben, Schriftarten und Schriftgrößen
- die Positionen von Beschriftungen

Die Definition einer Stadt in MapML lautet z.B.

```
<map type="city">
    <part id="city1" type="point" label="Hamburg" category="1">
        <point x="9.9748" y="53.5358"/>
    </part>
</map>
```

und durchläuft die in Quellcode 6.5 genannten XSLT-Templates. Die tatsächliche SWFML-Ausgabe in diesem Ausschnitt der Datei `map2swfml.xslt` ist blau markiert.

Der Aufbau der XSLT-Datei `weather2swfml.xslt`, die die Anweisungen für die Übersetzung einer WeatherML-Quelle nach SWFML enthält, ist nur unwesentlich komplizierter. Für jeden Parametertyp (Temperatur, Niederschlag, ...) ist den Isoflächen bzw. den daraus resultierenden Polygonen die dem Füllindex entsprechende Füllfarbe zuzuordnen. Die Anzahl der möglichen Füllfarben wird durch die Anzahl der Isowerte bestimmt, die während der Vektorisierung verwendet wurden.

6.3.3 Java-Applikation

Eine Applikation, die alle bisher entwickelten Komponenten verbindet und die Erzeugung einer SWFML-Datei aus einer MapML-, einer WeatherML- oder einer GRIB-Datei ermöglicht, setzt sich aus den folgenden Anweisungen zusammen:

1. Die Erzeugung der Objekte für Wetter- und Kartendaten

   `GribFile gribfile = new GribFile(` *Name der GRIB-Datei* `);`
   `GribRecord gribrec = gribfile.getRecord(` *Nummer des GRIB-Records* `);`
   `WeatherDocument doc = new WeatherDocument( gribrec );`

   oder

   `WeatherDocument doc = new WeatherDocument(` *Name der WeatherML-Datei* `);`

   oder

   `MapDocument doc = new MapDocument(` *Name der MapML-Datei* `);`

2. Die Erzeugung der Objekte der Konfigurationsdateien

   ```
   ConfigDocument config
         = new ConfigDocument( Name der Konfigurationsdatei );
   FileInputStream xslt
         = new FileInputStream( Dateiname des XSLT-Stylesheets );
   ```

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

    <!-- movie and shape layout -->
    <xsl:variable name="moviewidth">800</xsl:variable>
    <xsl:variable name="movieheight">600</xsl:variable>
    <xsl:variable name="movierate">1</xsl:variable>
    <xsl:variable name="moviebgcolor">FFB0B0B0</xsl:variable>
    <xsl:variable name="labeltextcolor">FF000000</xsl:variable>
    <xsl:variable name="labeltextsize">8</xsl:variable>

    <!-- rule for document root: start and end swfml -->
    <xsl:template match="map">
        <SWF rate="{$movierate}"     w="{$moviewidth}"
             color="{$moviebgcolor}" h="{$movieheight}">
            <xsl:apply-templates select="part" mode="defineshapes"/>
            <xsl:apply-templates select="part" mode="placeshapes"/>
        </SWF>
    </xsl:template>

    <!-- define all cities -->
    <xsl:template match="map[@type='city']/part" mode="defineshapes">
        <ellipse ID="{@id}" lc="FF000000" lw="1" color="FFFF0000"
                 x="{floor(point/@x) - 2}" w="4"
                 y="{floor(point/@y) - 2}" h="4"/>
        <text ID="{@id}label" color="{$labeltextcolor}"
              size="{$labeltextsize - @category}"
              x="{floor(point/@x) - $labeltextsize}"
              y="{floor(point/@y - $labeltextsize)}">
            <xsl:value-of select="@label"/>
        </text>
    </xsl:template>

    <!-- rule for all points -->
    <xsl:template match="point">
        <point x="{floor(@x)}" y="{floor(@y)}"/>
    </xsl:template>

    <!-- place city shapes -->
    <xsl:template match="map[@type='city']/part" mode="placeshapes">
        <PlaceObject ID="{@id}"      depth="3"/>
        <PlaceObject ID="{@id}label" depth="4"/>
    </xsl:template>

    ...

</xsl:stylesheet>
```

Quellcode 6.5: Ausschnitt aus `map2swfml.xslt`

3. Die Erzeugung des Ausgabestreams

   ```
   FileOutputStream output
         = new FileOutputStream( Dateiname der Ausgabedatei );
   ```

4. Die Transformation der Grafikobjekte

   ```
   doc.modify( config );
   ```

5. Die Ausgabe der Grafikobjekte als XML-Datei

   ```
   doc.write( output );
   ```

   oder (mit Übersetzung der XML-Ausprägung)

   ```
   doc.write( xslt, output );
   ```

Ein Vertreter dieser Applikationen ist z.B. `Grib2ml`, das als Input eine GRIB-Datei erwartet. Quellcode 6.6 zeigt einen Ausschnitt des Programmcodes. Ein Aufruf erfolgt von der Kommandozeile mit den Parametern:

```
java Grib2ml gribfile recordnumber configfile outputfile
```

oder wie in Quellcode 6.6 mit der Übersetzung von WeatherML in eine andere XML-Ausprägung

```
java Grib2ml gribfile recordnumber configfile xsltfile outputfile
```

Der Befehl

```
java Grib2ml dwd_tmp_199912261500.grb 1 grib_dwd_tmp.configml \
             weather2swfml.xslt tmp15.swfml
```

verarbeitet die Temperaturprognose des DWD für den 26.12.1999 um 15 Uhr (Zeitzone: GMT) zu einer SWFML-Datei. Dabei wird eine Kontrollausgabe erzeugt, wie in Quellcode 6.7 aufgelistet.

Zwei weitere Applikationen, die SWFML-Dateien erzeugen, sind `Weather2ml` und `Map2ml`. Beide Java-Programme lesen WeatherML- bzw. MapML-Dateien ein und geben dasselbe Format wieder aus oder wandeln es nach SWFML um, sofern ein XSLT-Stylesheet - z.B. `weather2swfml.xslt` bzw. `map2swfml.xslt` - als Kommandozeilenparameter angegeben wird.

Die Applikation `Swfml2Swf` erzeugt abschließend aus allen SWFML-Dateien den Flash-Film. `Swfml2Swf` wurde bereits in Abschnitt 5.7 vorgestellt. Der Quellcode befindet sich mit allen anderen Programmen dieser Beispielanwendung der Flash-Weather-Bibliothek auf der beigefügten CD-ROM im Verzeichnis `/beispiel`.

```
...

// get GRIB record
GribFile grib = new GribFile( args[0] );
int recordnumber = Integer.parseInt( args[1] );
GribRecord gribrec = grib.getRecord( recordnumber );

// create WeatherDocument from GRIB record
WeatherDocument weather = new WeatherDocument( gribrec );

// read configuration and transform graphic objects
ConfigDocument config = new ConfigDocument( args[2] );
weather.modify( config );

// use XSLT stylesheet during output of graphic objects
FileInputStream xslt = new FileInputStream( args[3] );
OutputStream out = new FileOutputStream( args[4] );
weather.write( xslt, out );

...
```

Quellcode 6.6: Ausschnitt aus `Grib2ml.java`

```
Reading GRIB file information...

GRIB record:
    IS section:
        Grib Edition 1
        length: 117980 bytes
    PDS section:
        parameter: Temperature
        level: 2 m above ground
        time: 26.12.1999  16:0
        dec.scale: 0
        GDS exists
        BMS exists
    GDS section:
        Rotated LatLon Grid  (325x325)
        lon: -12.5 to 7.75  (dx 0.062)
        lat: 3.25 to -17.0  (dy -0.063)
        south pole: lon 10.0 lat -32.5
        rot angle: 0.0
    BMS section:
        bitmap length: 105625
    BDS section:
        min/max value: -15.167572 20.332428
        ref. value: -15.167572
        bin. scale: -2
        num bits: 8

Generalizing...
    grid size before: 325x325
    ... combining 3 columns and 3 rows ...
    new grid size: 109x109

Projecting...
    rotated lat/lon grid -> geographic coordinates

Projecting...
    geographic coordinates -> Azimuthal Lambert projection
    reference point: (10.0,50.0)
    new bounds: x=[-0.21727598, 0.13271724] y=[-0.17233898, 0.18625306]

Resizing grid extent...
    new bounds: x=[-912.5591, 557.4124] y=[-782.2628, 723.8237]

Moving grid...
    new bounds: x=[-532.5591, 937.4124] y=[-396.26285, 1109.8237]

Clipping...
    grid size before clip: 109x109
    clip window: min=(-20.0,-20.0) max=(850.0,620.0)
    ... clipping ...
    grid size after clip: 66x48

Vectorizing...
    iso values: -25.0 -20.0 -15.0 -10.0 -5.0 0.0 5.0 10.0 15.0 20.0 25.0 30.0 35.0 40.0
    ... searching for iso values and building polygons ...
    ... cleaning up polygons ...
    found 20 polygons (10 internal) with 1382 points and 37 labels

Clipping...
    20 shapes before clip
    clip window: min=(0.0,0.0) max=(800.0,600.0)
    ... clipping ...
    17 shapes after clip

Building weather document...

Translating document...

Printing document...
```

Quellcode 6.7: `Grib2ml`-Kontrollausgabe

6.4 Testläufe

6.4.1 Testumgebung

Um die Lauffähigkeit der oben genannten Progamme zu testen und ihren Ressourcenbedarf zu messen, wurde die Erzeugung von Flash-Wetterfilmen auf folgenden Rechnern und Betriebssystemen durchgeführt:

- Pentium III Celeron 633Mhz 128MB, Linux 7.0
- Pentium II Celeron 466Mhz 128MB, Linux 6.3
- Pentium II Celeron 466Mhz 256MB, Windows 2000
- Pentium I 90Mhz 32MB, Linux 6.3

6.4.2 Programmaufrufe

Der Test umfasst die folgenden vier Programmaufrufe, bei denen die Konfigurationsdateien und XSLT-Stylesheets aus Abschnitt 6.3 zum Einsatz kommen:

1. **Map2ml**: Erzeugen einer SWFML-Datei aus einer MapML-Datei
   Die MapML-Datei hat eine Dateigröße von 143KB und die erzeugte SWFML-Datei ist 140KB groß.

   ```
   java Map2ml country.mapml map_deu.configml \
               map2swfml.xslt country.swfml
   ```

2. **Grib2ml**: Erzeugen einer SWFML-Datei aus einem GRIB-Record
   Die GRIB-Datei hat eine Dateigröße von 115KB und enthält nur einen Datenrecord. Die erzeugte SWFML-Datei ist 45KB groß.

   ```
   java Grib2ml dwd_tmp_15.grib 1 grib_dwd_tmp.configml \
               weather2swfml.xslt tmp15.swfml
   ```

3. **Grib2ml**: Erzeugen aller 24 SWFML-Dateien einer Tagesprognose
   Die GRIB-Dateien haben alle eine Dateigröße von 115KB und enthalten nur einen Datenrecord.

   ```
   for I in 00 01 02 03 04 05 06 07 08 09 10 11 \
            12 13 14 15 16 17 18 19 20 21 22 23
   do
       java Grib2ml dwd_tmp_"$I".grib 1 grib_dwd_tmp.configml \
                   weather2swfml.xslt tmp"$I".swfml
   done
   ```

4. **Swfml2Swf**: Erstellen des Flash-Films aus 24 + 4 SWFML-Dateien
   Die SWFML-Dateien habe eine durchschnittliche Dateigröße von 45KB, der erzeugte Flash-Film hat eine Dateigröße von 170KB.

```
java Swfml2Swf country.swfml river.swfml lake.swfml city.swfml \
               tmp*.swfml tmp.swf
```

6.4.3 Messergebnisse

Auf jedem der vier Testrechner wurden alle vier Programmaufrufe durchgeführt. Die Laufzeit wurde mit dem `time`-Kommando gemessen und der maximale Arbeitsspeicherbedarf anschliessend in einem unabhängigen Test mit Hilfe von `top` oder dem NT-Taskmanager ermittelt. Die Testergebnisse können daher Ungenauigkeiten enthalten, geben aber eine recht gute Größenordnung der Laufzeit und des verwendeten Hauptspeichers wieder.

Die Ergebnisse der Testläufe enthält Tabelle 6.1.

	Pentium III 633Mhz 128MB, Linux 7.0	**Pentium II 433Mhz** 128MB, Linux 6.3
1. **Map2ml**	14.5s / 16.2MB	18.9s / 14.8MB
2. **Grib2ml**	8.5s / 19.5MB	11.3s / 18.9MB
3. **Grib2ml**	3min 18s / 19.5MB	4min 18s / 18.9MB
4. **Swfml2Swf**	1min 7s / 51.0MB	1min 34s / 51.8MB

	Pentium II 433Mhz 256MB, Windows 2000	**Pentium I 90Mhz** 32MB, Linux 6.3
1. **Map2ml**	10.4s / 16.2MB	1min 3s / 15.4MB
2. **Grib2ml**	8.9s / 18.8MB	37.4s / 19.1MB
3. **Grib2ml**	3min 36s / 18.8MB	14min 5s / 19.1MB
4. **Swfml2Swf**	47s / 52.9MB	5min 56s / 52.0MB

Tabelle 6.1: Messergebnisse der Testläufe

Eine Optimierung der Laufzeit ist jederzeit durch die Flagge `-Xms<Speichergröße>` des Java-Interpreters möglich, die der Applikation schon zu Beginn der Ausführung die angegebene Menge Arbeitspeicher reserviert. Z.B. verkürzt der Aufruf

```
java -Xms20MB Grib2ml dwd_tmp_15.grib 1 grib_dwd_tmp.configml \
                      weather2swfml.xslt tmp15.swfml
```

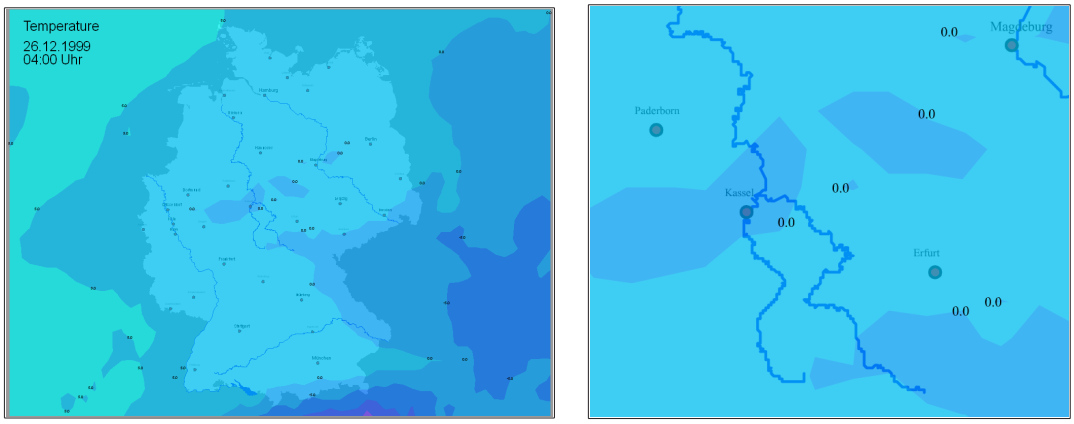
die Laufzeit des zweiten Testkommandos auf dem Penium III auf 7.3s (vorher 8.5s).

6.5 Wetterfilme

Vom Deutschen Wetterdienst stehen Prognosedaten für den 26.12.1999 zur Verfügung, die die Parameter Temperatur, Niederschlag, Bewölkungsgrad und Luftdruck umfassen. Das rote Rechteck in Abbildung 6.2 markiert das Gebiet, das die Daten abdecken. Die daraus erzeugten Flash-Filme zeigen die Abbildungen 6.3 bis 6.6.

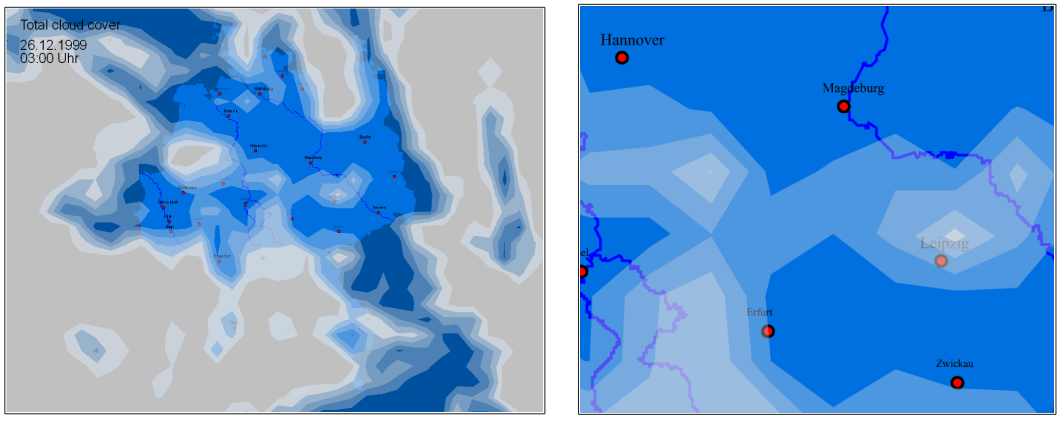
Leider können in dieser Arbeit nur pixelbasierte Abbildungen (Screenshots) der Wetterfilme gezeigt werden, da die Füllfarben der Isoflächen die Transparenzfähigkeit von Flash ausnutzen, die eine Postscript-Grafik nicht wiedergeben kann.

Abbildung 6.2: Abgedecktes Gebiet der DWD-Daten



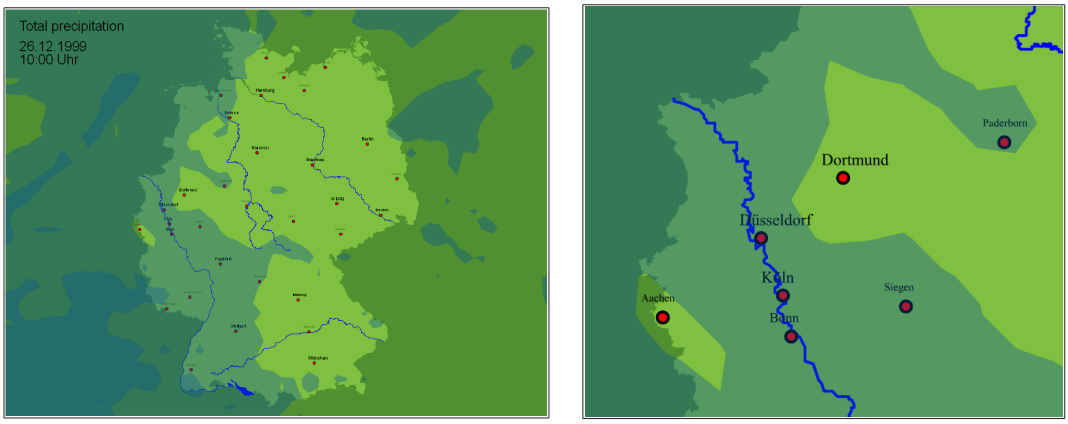
(a) Gesamtbild (b) Ausschnitt

Abbildung 6.3: Deutschland: Temperatur



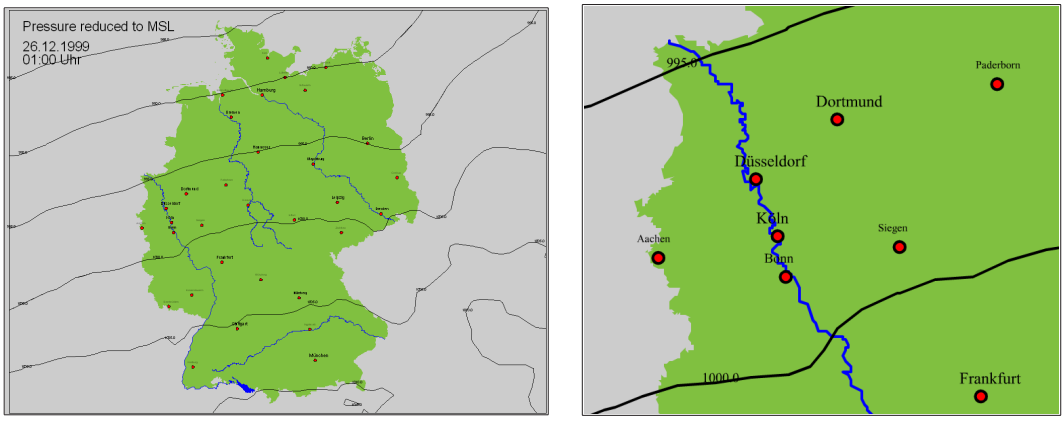
(a) Gesamtbild (b) Ausschnitt

Abbildung 6.4: Deutschland: Bewölkungsgrad



(a) Gesamtbild (b) Ausschnitt

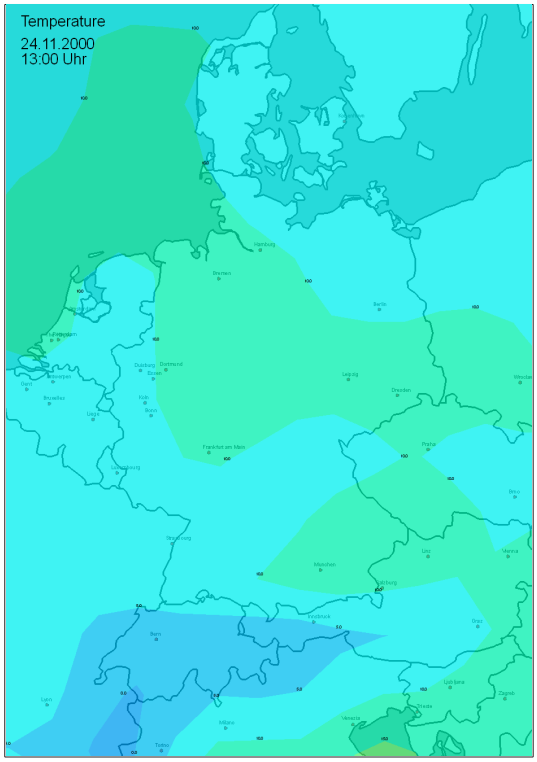
Abbildung 6.5: Deutschland: Niederschlag



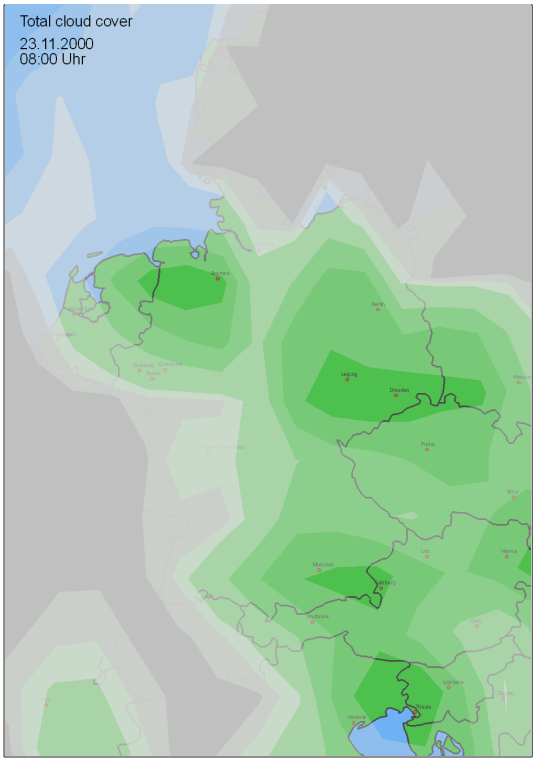
(a) Gesamtbild (b) Ausschnitt

Abbildung 6.6: Deutschland: Luftdruck

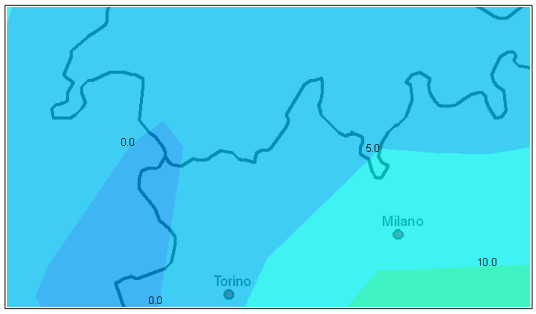
Weitere GRIB-Dateien wurden von einem Internetrechner des National Weather Service (NWS) der USA heruntergeladen [NWS2000]. Sie enthalten die Wetterprognose für die gesamte Erde, die Auflösung des Rasters ist jedoch viel geringer als bei den Daten des DWD. Außerdem liegt zwischen den Prognosen ein zeitlicher Abstand von drei Stunden. Die Abbildungen 6.7 und 6.8 zeigen Momentaufnahmen aus möglichen Wetterfilmen.



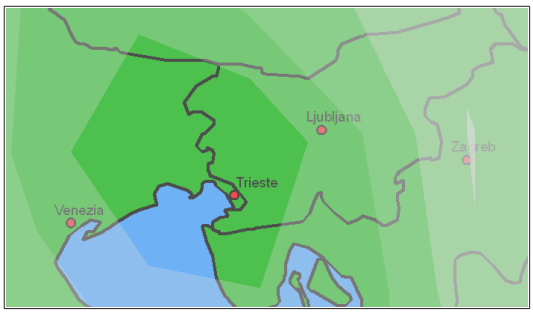
(a) Temperatur (Gesamtbild)



(b) Bewölkung (Gesamtbild)



(c) Temperatur (Ausschnitt)



(d) Bewölkung (Ausschnitt)

Abbildung 6.7: Deutschland: Temperatur und Bewölkungsgrad

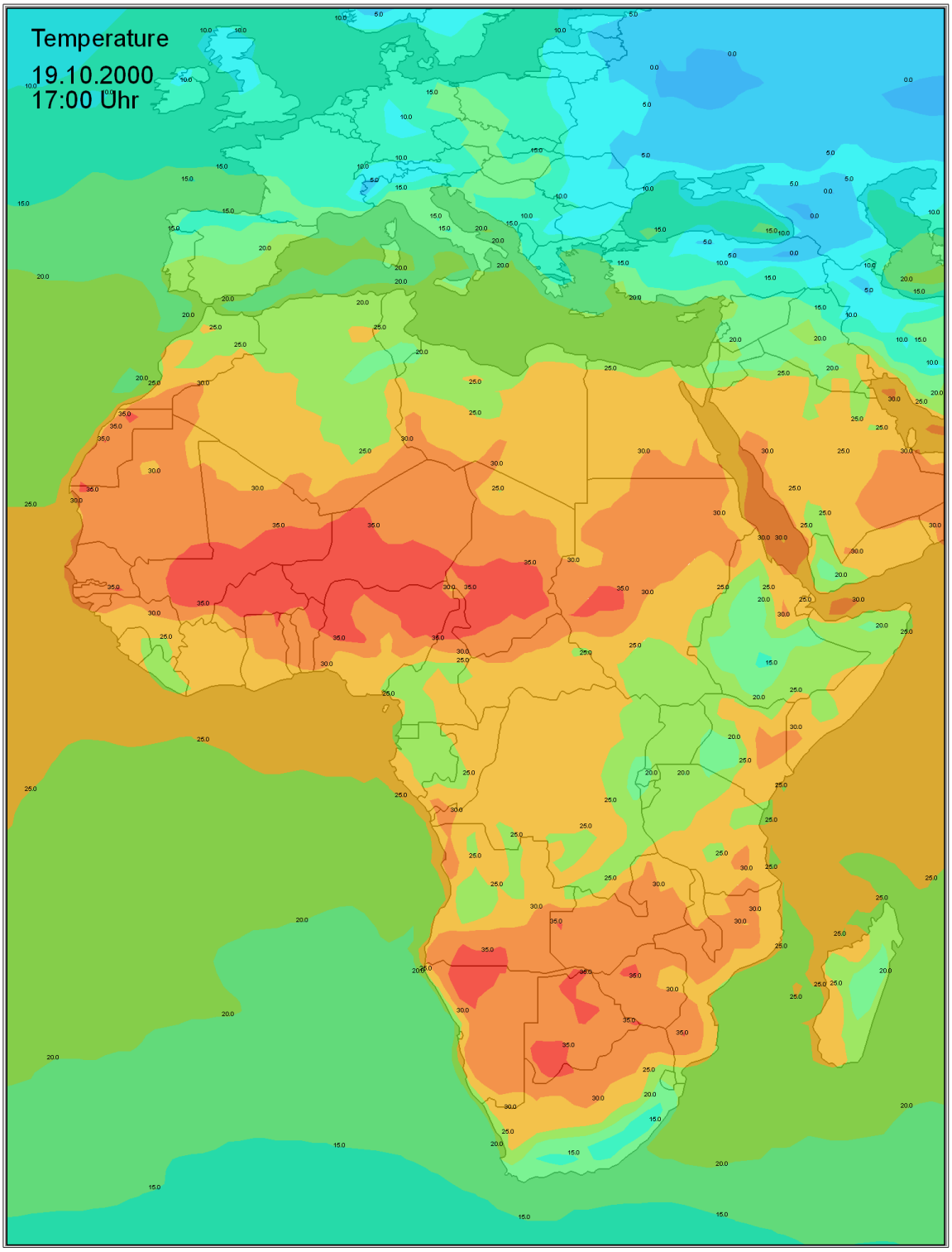


Abbildung 6.8: Afrika: Temperatur

III Resumée und Ausblick

7 Resumée

Die vorliegende Diplomarbeit führt am Beispiel von Wetterprognosedaten eine „Visualisierung von raum- und zeitbezogenen Daten mittels Macromedia Flash“ durch. Daher der Titel *Flash Weather*. Wetterdaten werden weltweit in einem standardisierten Dateiformat ausgetauscht, das die Werte eines Parameters (wie Temperatur oder Niederschlag) für die Punkte eines rechteckigen, regelmäßigen Rasters angibt. Eine Kernaufgabe dieser Diplomarbeit war deshalb die Entwicklung eines Algorithmus zur Vektorisierung dieser Rasterdaten.

Die Aufgabenstellung der Diplomarbeit gliederte sich in zwei Teile: Die Verarbeitung und Vorbereitung von raum- und zeitbezogenen Daten, zu der auch die Vektorisierung von Rasterdaten gehört, und die Visualisierung der Daten mit Macromedia Flash.

Für die Datenvisualisierung wurde Macromedia Flash gewählt, weil es gegenüber pixelbasierten Formaten wie JPEG oder GIF erhebliche Vorteile besitzt und dank seiner Verbreitung inzwischen zu einem Quasi-Standard im Internet geworden ist. Macromedia stellt außerdem ein kostenloses Software Development Kit (SDK) zur Verfügung, das eine programmgesteuerte Erzeugung von Flash-Filmen ermöglicht.

Um die Verwendung des SDK zu vereinfachen, war eine erste Überlegung, anstatt der Programmierschnittstelle und der damit verbundenen „Programmierung“ von Flash-Filmen ein in *XML* (eXtensible Markup Language) definiertes Schnittstellenformat zu entwickeln, das ein auf dem SDK basierender Generator in einen Flash-Film umwandelt. Mit der Entwicklung eines solchen Flash-Generators befasst sich die Diplomarbeit meines Kommilitonen Ralf Kunze. Im Rahmen der vorliegenden Arbeit konnte zur Erzeugung der Flash-Filme vorerst ein sehr ähnliches Softwaretool verwendet werden, das jedoch nicht den vollen Funktionsumfang von Flash umsetzen kann.

Der Inhalt dieser Diplomarbeit umfasst daher zum größten Teil die Vorbereitung von raum- und zeitbezogenen Daten für die Visualisierung. Zu diesem Zweck wurden eine Reihe von Java-Klassen implementiert, die Grafikobjekte (Punkte, Linien, Polygone) repräsentieren, computergrafische Transformationen auf diesen Grafikobjekten durchführen und die Ein- und Ausgabe von Daten bzw. Konfigurationsdateien unterstützen. Die Klassen bilden zusammen die *FlashWeather-Bibliothek*.

Da als Dateneingabeformat nicht beliebige Dateiformate unterstützt werden konnten, wurde XML als Schnittstellenformat gewählt. XML ist ein noch recht junger Standard des W3C, der aber als der zukünftige Standard für den Datenaustausch

eingeschätzt wird. Zudem steht leistungsfähige Software zur Verarbeitung von XML-Daten kostenlos zur Verfügung.

Die Grafikobjekt-Klassen der FlashWeather-Bibliothek besitzen auf XML basierende Ein- und Ausgabeschnittstellen, über die die Grafikobjekte eingelesen und später wieder gespeichert werden können. Zusätzliche Lesemodule wurden für das standardisierte Wetterdatenformat und für Dateien des Geografischen Informationssystems ARC/INFO implementiert.

Für die computergrafische Verarbeitung der Grafikobjekte stehen neben den elementaren Transformationen mehrere Clipping-Algorithmen, Projektionen und die Vektorisierung von Rasterdaten als Teil der FlashWeather-Bibliothek zur Verfügung. Ihre Anwendung lässt sich durch eine ebenfalls in XML definierte Konfigurationsdatei steuern.

Auf diese Weise entsteht ein Visualisierungsprozess der nahezu beliebige in XML definierte Daten verarbeiten kann. Ein geringer Programmieraufwand muss für jedes neue Datenformat nur einmal erfolgen, die weitere Konfiguration der angewendeten Transformationen und die Definition des Layouts der Flash-Filme erfolgen über Konfigurationsdateien bzw. Stylesheets.

Abbildung 7.1 fasst den Datenfluss von der Datenvorbereitung zur Visualisierung zusammen.

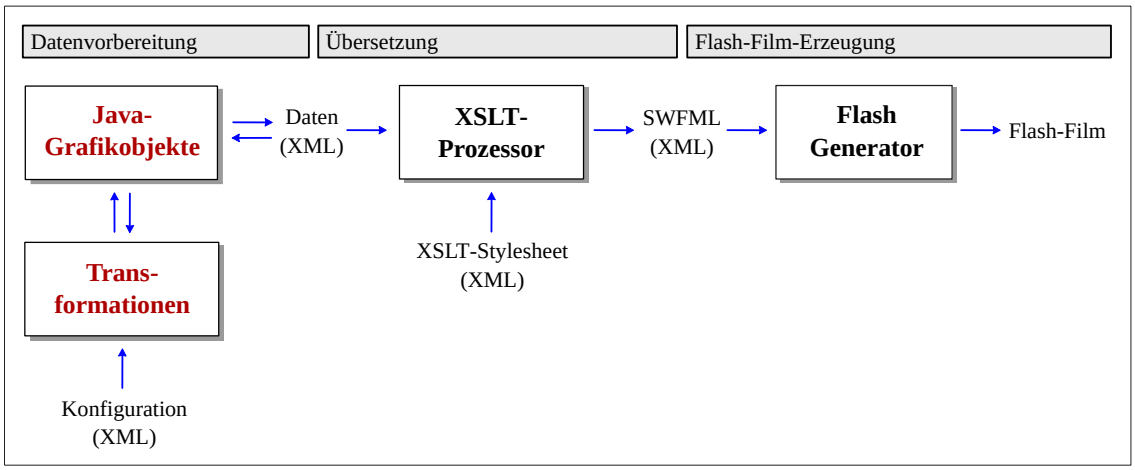


Abbildung 7.1: Datenfluss von der Datenvorbereitung zur Visualisierung

Die Vorteile eines Visualisierungsprozesses, wie er in dieser Diplomarbeit entwickelt wurde, entstehen hauptsächlich durch die Verwendung der standardisierten XML-Techniken. Ein XML-Parser kann direkt als Lesemodul und Syntaxprüfer für beliebige in XML definierte Sprachen eingesetzt werden. Ein XSLT-Prozessor führt die Übersetzung einer Sprache in eine andere durch.

Die FlashWeather-Bibliothek basiert auf diesen Standardschnittstellen und integriert bereits existierende Software. Ohne Programmierung können auf in XML codierten Grafikobjekten computergrafische Transformationen mit frei wählbaren Parametern durchgeführt werden und anschließend unterschiedlichste Wetterfilme aus diesen Daten generiert werden.

8 Ausblick

Die Implementation der FlashWeather-Bibliothek und der Beispielklassen ermöglicht eine einfache Bearbeitung von Wetterdaten, Kartendaten und in XML codierten Grafikobjekten und die Erzeugung von Flash-Filmen aus diesen Komponenten.

Während der Bearbeitung dieser Diplomarbeit gab es darüber hinaus Ideen zur leichteren Verwendung der FlashWeather-Klassen und zur Optimierung des Layouts der Flash-Filme, die nicht umgesetzt wurden. Im Folgenden werden einige davon kurz beschrieben und es wird erläutert, warum die Ideen zurückgestellt wurden.

Morphing

Flash bietet die Möglichkeit, Grafikobjekte (beinahe) stufenlos ineinander übergehen zu lassen, so dass aus einer ruckelnden Bewegung eine fließende wird. Diese *Morphing* genannte Technik erfordert aber die genaue Kenntnis, welches Grafikobjekt in welches Grafikobjekt des nachfolgenden Animationsbildes übergeht.

Das Beispiel der Wetterdaten liefert für jede stündliche Prognose eine variable Anzahl von Isoflächen unterschiedlicher Größe. Die Aufgabe, den Übergang zwischen den Isoflächen von zwei Stundenprognosen zu ermitteln, erfordert daher einen aufwendigen Algorithmus: Flächen können sich auflösen, hinzukommen oder miteinander verschmelzen. Deswegen wurden weitere Überlegungen zu diesem Thema vorerst zurückgestellt.

Verbesserung des Vektorisierers

Der als Teil der FlashWeather-Klassen implementierte Vektorisierungs-Algorithmus verbindet die ermittelten Randpunkte der Isoflächen, d.h. die Polygonpunkte, durch einfache Geradenstücke. Dieser eckige Kurvenverlauf der Isolinien könnte durch die Verwendung von Splines oder Bezier-Kurven abgerundet werden.

Flash unterstützt bei Polygonen neben geraden Begrenzungslinien auch quadratische Bezier-Kurven, deren Verwendung einen weicheren Kurvenverlauf zur Folge hätte. Diese Möglichkeit wurde aber vorerst nicht weiter verfolgt. Die Verwendung von Bezierkurven vergrößert zudem das Datenvolumen durch die zusätzlich notwendigen Stützpunkte je Kurvenstück erheblich.

Weitere Verwendung

Prinzipiell lassen sich mit Hilfe der FlashWeather-Klassen alle in XML definierten Flächen, Linien und Punkte einlesen, transformieren und wieder ausgeben. Ihr Format muss jedoch die in Kapitel 5 festgelegte Struktur aufweisen. Dank der XSLT-Transformationssprache lassen sich jedoch auch andere XML-Formate mit Hilfe eines XSLT-Prozessors wie dem Apache Xalan in diese Struktur konvertieren.

Neben der Darstellung von Wetterprognosedaten ist es denkbar, die Verwendung der FlashWeather-Klassen z.B. auf Daten aus Geografischen Informationssystemen (GIS) auszudehnen. Ein Lesemodul für ESRI Shapefiles existiert bereits. In einer Darstellung der GIS-Daten als Flash-Film könnten z.B. unterschiedliche Ausprägungen der zugrunde liegenden Karte mit den dazugehörigen sozioökonischen Daten in verschiedenen Konstellationen über einen bestimmten Zeitraum betrachtet werden.

Eine Verwendung des Visualierungsprozesses im Rahmen einer “on the fly”-Generierung von Flash-Filmen wie beim Macromedia Generator ist jedoch derzeit aus Performancegründen nicht denkbar.

Eigener Flash-Generator

Wie bereits im Abschnitt 4.5 angesprochen, entwickelt mein Kommilitone Ralf Kunze im Rahmen seiner Diplomarbeit einen eigenen Flash-Generator auf Basis des Macromedia SWF-SDK. Dieser wird im Gegensatz zum aktuell verwendeten Flash-Generator Saxess Wave alle Möglichkeiten des Flash-Formats, d.h. auch ActionScript und Bitmap-Füllungen, über eine XML-Schnittstelle zur Verfügung stellen.

Der neue Generator wird eine leicht veränderte und erweiterte Form der Schnittstellensprache SWFML benutzen, kann aber in den bisherigen Visualisierungsprozess sehr leicht integriert werden. Lediglich die XSLT-Stylesheets für die Übersetzung von z.B. MapML oder WeatherML nach SWFML müssen für seine Verwendung angepasst werden.

Erst mit der kompletten ActionScript-Unterstützung durch den neuen Flash-Generator kann eine grafische Benutzeroberfläche für die Steuerung der Wetterfilme realisiert werden.

Benutzeroberfläche

Die bisher erzeugten Flash-Filme zeigen beliebige raum- und zeitbasierte Daten, jedoch kann der zeitliche Ablauf bis auf das Starten und Stoppen der Animation

vom Benutzer nicht beeinflusst werden, sondern ähnelt einer Diashow. Auch an dieser Stelle wird die Diplomarbeit von Ralf Kunze nach der Fertigstellung des Flash-Generators einige Erweiterungen vornehmen:

- Der zeitliche Ablauf der Animation wird über Play- und Stop-Buttons oder über einen Schieberegler zu steuern sein.
- Mehrere Flash-Filme werden sich in beliebiger Reihenfolge und Transparenz übereinander positionieren lassen. So können z.B. Temperatur und Bewölkung für ein Gebiet gleichzeitig betrachtet werden.
- Die Hintergrundelemente werden sich ebenso wie die Filme der eigentlichen Daten an- und abschalten lassen.

Eine Benutzeroberfläche wird also die Steuerung der Flash-Filme ermöglichen. Entsprechend wäre eine Benutzerschnittstelle zur Erzeugung der Flash-Filme denkbar. Sie könnte das Erstellen der benötigten Konfigurationsdateien und XSLT-Transformationsdateien erleichtern und damit die Durchführung des Visualisierungsprozesses vereinfachen.

Scalable Vector Graphics (SVG)

Im Rahmen dieser Diplomarbeit wurde Macromedia Flash als Ausgabeformat gewählt. Die Gründe dafür und die Vorteile eines Vektorgrafikformats wurden in Kapitel 2 erörtert. Flash hat sich innerhalb der letzten Jahre durch seine Verbreitung des Flash-Players bzw. des Flash-Plugins zu einem Quasi-Standard im Internet entwickelt. Doch auch wenn Macromedia das Dateiformat SWF offen gelegt hat, handelt es sich doch um die Entwicklung einer einzelnen Firma. Deshalb gibt es Bestrebungen, einen offenen Standard für ein Vektorgrafikformat für das Internet zu entwickeln.

Unter der Zusammenarbeit von nahezu allen großen Herstellern von Bildbearbeitungssoftware - so auch Macromedia - wurde die Spezifikation *Scalable Vector Graphics (SVG) 1.0* entwickelt und am 2. November 2000 vom W3C als *Candidate Recommendation* verabschiedet. SVG befindet sich damit in der letzten Vorstufe zu einem offenen Standard. Die Spezifikation definiert die Eigenschaften und die Syntax einer in XML definierten Sprache zur Beschreibung von zweidimensionalen Vektorgrafiken und gemischten Vektor-/Rastergrafiken [W3C2000g].

SVG besitzt als Vektorgrafikformat viele Vorteile gegenüber herkömmlichen Bildformaten wie JPEG oder GIF. Wie Flash unterstützt es Animation und Interaktivität mit dem Benutzer. Im direkten Vergleich mit Flash hat SVG die folgenden Vorzüge [Sun2000]:

- **Textformat**
  SVG ist in XML definiert und ist daher ein textbasiertes Format, das mit einer Vielzahl von Programmen - vom Bildbearbeitungsprogramm bis zum Texteditor - bearbeitet werden kann. Seine Dateigröße kann für die Übertragung im Internet durch Komprimierung erheblich reduziert werden. Flash dagegen besitzt sein eigenes bitcodiertes Format.

- **Text in Grafiken**
  Enthalten SVG-Grafiken Text, so kann dieser selektiert und durchsucht werden. Copy & Paste der Texte ist möglich und Suchmaschinen können Textpassagen aus SVG-Grafiken in ihren Index aufnehmen.

- **Vorteile von XML**
  Weil SVG eine XML-Ausprägung ist, verfügt es über alle Vorteile von XML:
  - Es existiert eine umfangreiche Unterstützung durch Softwaretools.
  - Das Dokument lässt sich über Standardschnittstellen wie das DOM auslesen und manipulieren.
  - SVG-Dateien erweitern die Dokumentstruktur einer HTML-Datei, in die sie eingebettet sind; JavaScript hat so Zugriff auf die komplette Grafikstrukur.
  - XSLT-Stylesheets lassen sich für eine Manipulation bzw. Übersetzung des Dokuments anwenden.

Als ein Nachteil von SVG ist seine momentane Verbreitung zu nennen. SVG ist noch nicht als endgültige Recommendation verabschiedet und bisher gibt es nur für neuere Programme erste Import- und Export-Funktionen. Für die Darstellung von SVG-Dateien in einem Webbrowser existiert von Adobe ein etwa 3 MB großes Plugin.

Da aber viele große Firmen an der Spezifikation von SVG mitgearbeitet haben, wird eine Unterstützung des Grafikformats zunehmen und eventuell kann schon die nächste Version der Webbrowser von Microsoft und Netscape SVG ohne Plugin darstellen. Wird dann SVG als offener Standard und mit seinen erweiterten Möglichkeiten Flash nach und nach ersetzen?

Dank der XSLT-Transformationssprache ist es möglich, die FlashWeather-Klassenbibliothek auch ohne anschließende Generierung eines Flash-Films zu nutzen. Stattdessen können z.B. MapML- und WeatherML-Dateien über XSLT-Stylesheets in das SVG-Format konvertiert werden und auf diesem Weg SVG-Wetterfilme generiert werden.

Literatur

[Apac2000a] **Apache Software Foundation**
Apache Software Foundation Homepage
http://www.apache.org

[Apac2000b] **Apache Software Foundation**
Xerces Java Parser
http://xml.apache.org/xerces-j

[Apac2000c] **Apache Software Foundation**
Apache Software License
http://xml.apache.org/LICENSE

[Beck2000] **Becker**, **Oliver**
Weitere Spezifikationen im XML-Umfeld, HU-Berlin 05/2000
http://www.informatik.hu-berlin.de/ xing/Einstieg

[Behm1999] **Behme**, **Henning**
XSLT: Transformation von XML-Dokumenten, iX 11/1999
http://www.heise.de/ix/artikel/1999/11/181

[BeMi1998] **Behme, Henning & Mintert, Stefan**
XML in der Praxis, Addison-Wesley Longman, München 1998
http://www.mintert.com/xml/buch

[Bour1987] **Bourke, Paul**
CONREC, A Contouring Subroutine
http://www.swin.edu.au/astronomy/pbourke/projection/conrec

[Bosa1997] **Bosak, Jon**
XML, Java and the future of the Web, Sun Microsystems, 10.03.1997
http://www.ibiblio.org/bosak/xml/why/xmlapps.htm

[Card1999] **Cardhouse.com**
A Tepid Raster To Vector Algorithm
http://www.cardhouse.com/computer/vector.htm

[Debo2000] **Debon, Olivier**
Multi-platform Flash plugin download section
http://www.geocities.com/TimesSquare/Labyrinth/5084/flash

[DWD1999] **Deutscher Wetterdienst**
Quaterly Report No.20, June-August 1999, S.43
http://www.dwd.de/research/publications/QuarterlyReport

[DWD2000] **Deutscher Wetterdienst**
Deutscher Wetterdienst Homepage
http://www.dwd.de

[ESRI1998] **ESRI Inc.**
ESRI Shapefile Technical Description
http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf

[Faus2000] **Faust, Hans-Jürgen**
Netze im Net: Echte Entwürfe
http://home.main-rheiner.de/hansjuergen.faust

[Fell1992] **Fellner, Wolf-Dietrich**
Computergrafik, 2. Auflage
Reihe Informatik, Band 58, BI-Wissenschaftsverlag, 1992

[Hosc1984] **Hoschek, Josef**
Mathematische Grundlagen der Kartographie, 2. Auflage
TH-Darmstadt, BI-Wissenschaftsverlag, 1984

[IDC2000] **International Data Corporation**
Online User Forecast, Stand: Dezember 2000
http://www.idc.com

[Libe2000] **Libenzi, Davide**
Ras2Vec raster to vector conversion program
http://www.mycio.com/davidel

[Mach1997] **Macherius, Ingo**
XML: Professionelle Alternative zu HTML, iX 6/1997
http://www.heise.de/ix/artikel/1997/06/106

[Macr2000a] **Macromedia Inc.**
Macromedia Corporate Fact Sheet
http://www.macromedia.com/macromedia/proom

[Macr2000b] **Macromedia Inc.**
Macromedia Flash White Paper, Stand: November 2000
http://www.macromedia.com/software/flash/survey

[Macr2000c] **Macromedia Inc.**
Flash File Format SDK Documentation
http://www.macromedia.com/software/flashplayer/downloads

[Macr2000d] **Macromedia Inc.**
Flash File Format SDK License
http://www.macromedia.com/software/flashplayer/downloads

[Matr2000] **Matrix.Net Inc.**
Internet Reaches 100 Mio. Hosts, Press Release 01. November 2000
http://www.matrix.net/company/news

[Megg2000] **Megginson Technologies**
SAX: Simple API for XML, Version 2.0
http://www.megginson.com/SAX

[NCEP2000] **National Centers for Environmental Prediction**
wgrib - decode/manipulate GRIB files
http://wesley.wwb.noaa.gov/wgrib.html

[NWS2000] **National Weather Service**
U.S. National Weather Service Homepage
http://www.nws.noaa.gov

[Saxe2000] **Saxess Software Design**
Saxess Wave: Shockwave Flash encoder
http://www.saxess.com

[Sosn1999] **Sosna, Dr. Dieter**
GIS: Kartennetz-Entwürfe, Universität Leipzig
http://www.informatik.uni-leipzig.de/~sosna/karten/netze.html

[SnyJ1987] **Snyder, John P.**
Map Projections: A Working Manual, USGS Prof. Paper 1395
Washington DC, U. S. Government Printing Office, S. 182-190, 1987

[SnyW1978] **Snyder, William V.**
Algorithm 531 - Contour Plotting [J6]
ACM Trans. on Math. Software, Vol. 4, No. 3, S.290, Sep. 1978

[Sun2000] **Sun Microsystems Inc.**
Sun XML, Developer Connection: Introduction to SVG
http://www.sun.com/software/xml/developers/svg

[SuHo1974] **Sutherland, Ivan E. & Hodgman, G. W.**
Reentrant Polygon Clipping
Communications of the ACM, Vol. 17, No. 1, S.32, Jan. 1974

[Vose1998] **Voser, Stefan A.**
GIS-Tutorial: Koordindatensysteme
http://www.gis-tutor.de/theorie/grundlag/koordsys/koordina.htm

[Warm2000] **Warmerdam, Frank**
Shapefile C Library
http://members.home.com/warmerda

[Webe2000] **Weber, Martin**
AutoTrace - converts bitmap to vector graphics
http://homepages.go.com/~martweb/AutoTrace.htm

[WMO1998] **World Meteorological Organization**
A Guide to Grib Edition 1
http://www.wmo.ch/web/www/reports/Guide-binary-2.html

[W3C2000a] **World Wide Web Consortium**
About the World Wide Web Consortium (W3C)
http://www.w3.org/Consortium

[W3C2000b] **World Wide Web Consortium**
Extensible Markup Language (XML)
http://www.w3.org/XML

[W3C2000c] **World Wide Web Consortium**
Web Style Sheets
http://www.w3.org/Style

[W3C2000d] **World Wide Web Consortium**
Extensible Stylesheet Language (XSL)
http://www.w3.org/Style/XSL

[W3C2000e] **World Wide Web Consortium**
W3C Technical Reports and Publications
http://www.w3.org/TR

[W3C2000f] **World Wide Web Consortium**
Document Object Model (DOM)
http://www.w3.org/DOM

[W3C2000g] **World Wide Web Consortium**
Scalable Vector Graphics (SVG)
http://www.w3.org/Graphics/SVG

[Wolt1999] **Wolter, Sascha**
Flash 4, 1. Auflage
Galileo Press GmbH, Bonn 1999

[ZAIT2000] **Zentrum für angewandte Informationstechnologien**
Webbüro, Universität Bremen
http://www.zait.uni-bremen.de → Webbüro

Anhang

A Externe Softwarepakete

Folgende Softwarepakete werden für die Verwendung der FlashWeather-Klassen benötigt und können kostenlos aus dem Internet bezogen werden. Die Benutzung einiger Produkte unterliegt lizenzrechtlichen Bestimmungen.

- **Sun Java 2 SDK**
  http://java.sun.com
- **Apache Xerces Java Parser**
  http://xml.apache.org
- **Apache Xalan-Java**
  http://xml.apache.org
- **Saxess Wave Flash-Generator**
  http://www.saxess.com
- **C-Bibliothek shapefile**
  http://members.home.com/warmerda

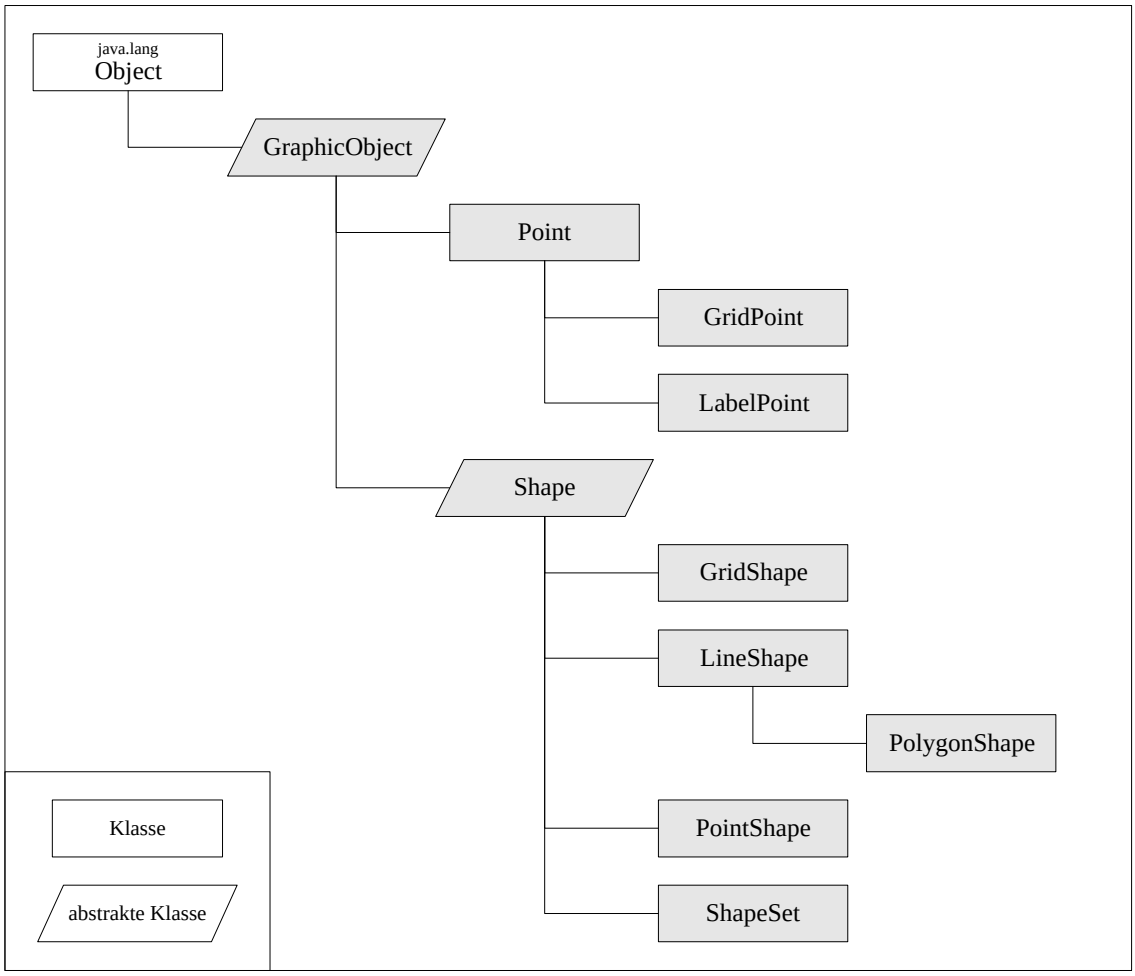
B FlashWeather-Bibliothek

Die FlashWeather-Bibliothek gliedert sich in Grafikobjekt- und Transformationsklassen, die sich in den Java-Paketen `de.flashweather.graph2d` befinden, und Ein-/Ausgabe-Klassen mit der Paket-Bezeichnung `de.flashweather.io`. Ihre mit `Javadoc` erzeugte API-Dokumentation ist auf der beigefügten CD-ROM im Verzeichnis `/docs/flashweather` zu finden.

Die folgenden Grafiken veranschaulichen zusätzlich die Klassenhierarchie innerhalb der einzelnen Pakete. Die Klassen der FlashWeather-Bibliothek sind grau hinterlegt.

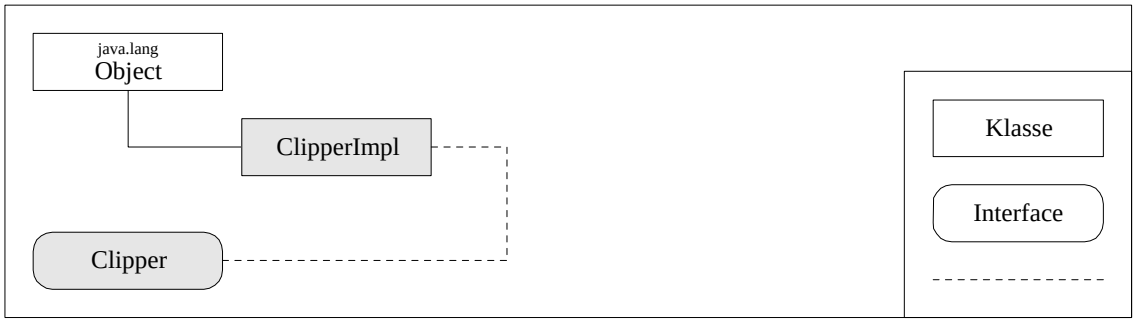
de.flashweather.graph2d.base

Dieses Paket enthält die Java-Grafikobjekte.



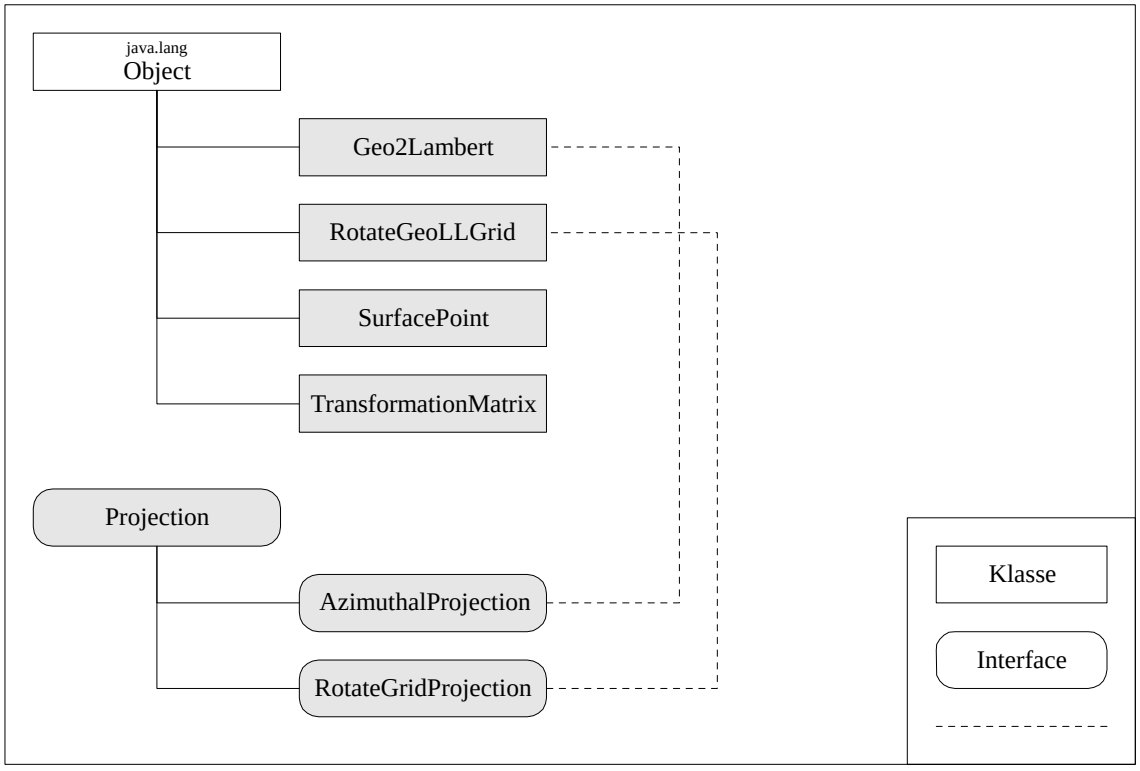
de.flashweather.graph2d.clip

Dieses Paket enthält ein Interface für Clipping-Algorithmen, die auf den Java-Grafikobjekten operieren, und eine Klasse, die das Interface implementiert.



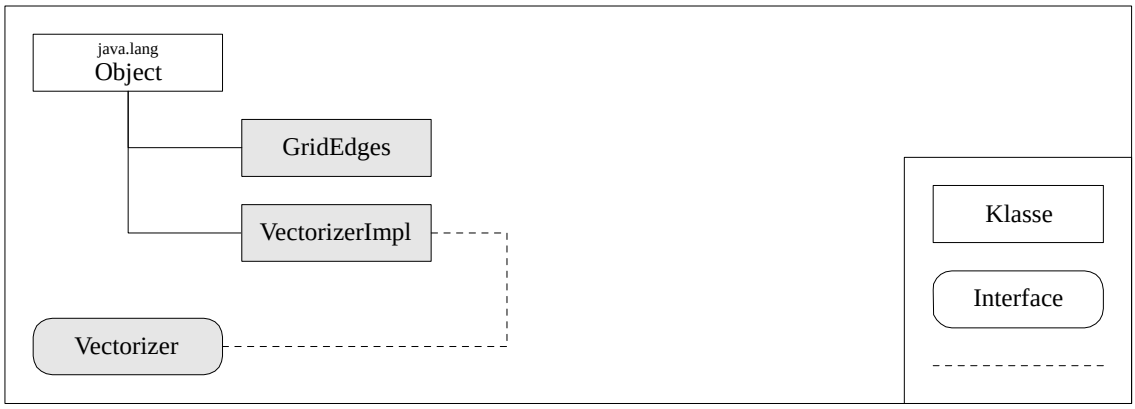
de.flashweather.graph2d.project

Dieses Paket enthält ein Interface für eine Projektion der Java-Grafikobjekte, zwei darauf aufbauende spezielle Projektion und die Umsetzung dieser speziellen Projektionen mit einigen Hilfsklassen.



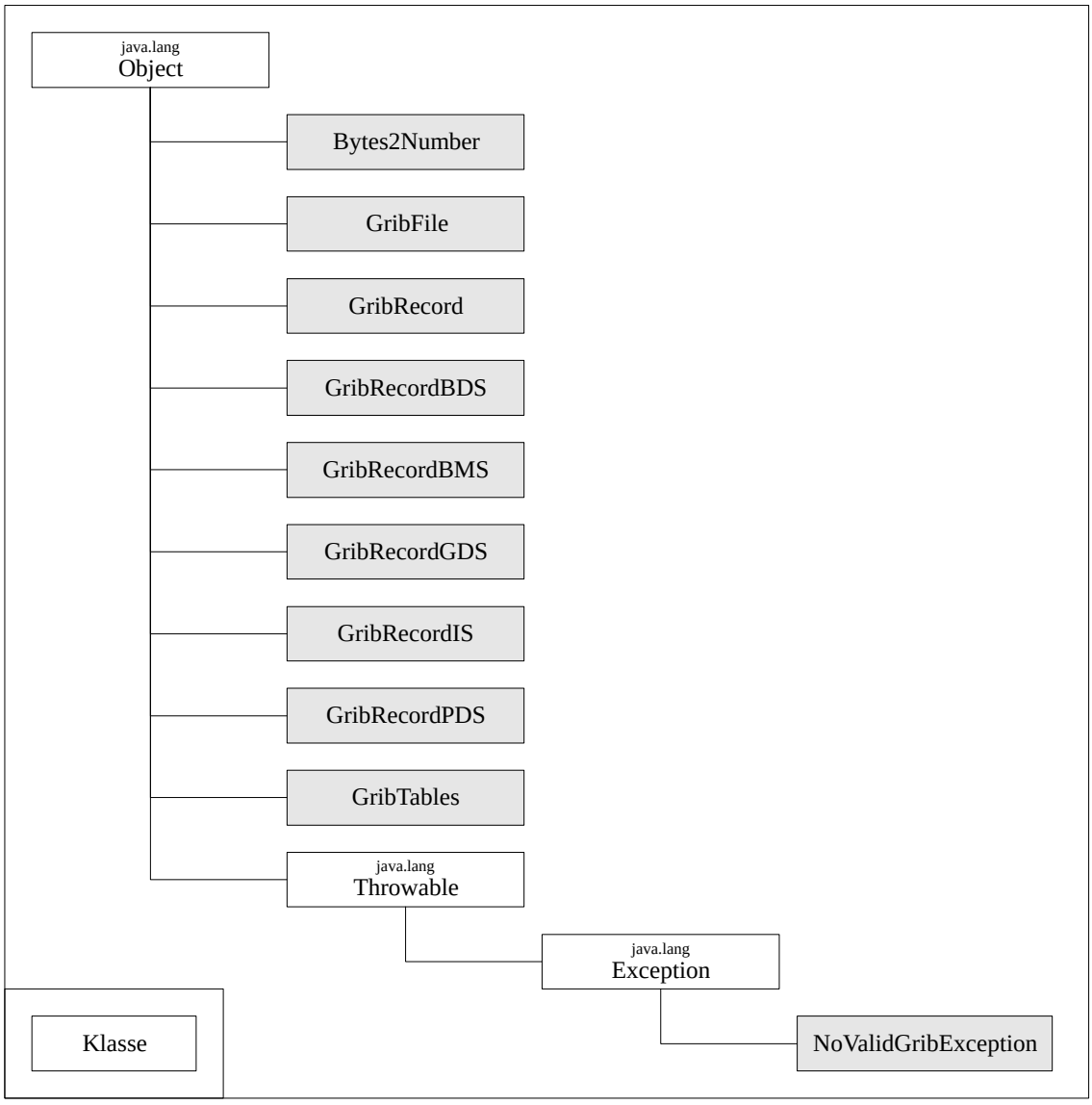
de.flashweather.graph2d.vectorize

Dieses Paket enthält ein Interface für einen Vektorisierungs-Algorithmen, der auf den Java-Grafikobjekten operiert, und eine Klasse, die das Interface umsetzt und den erweiterten Snyder-Algorithmus implementiert.



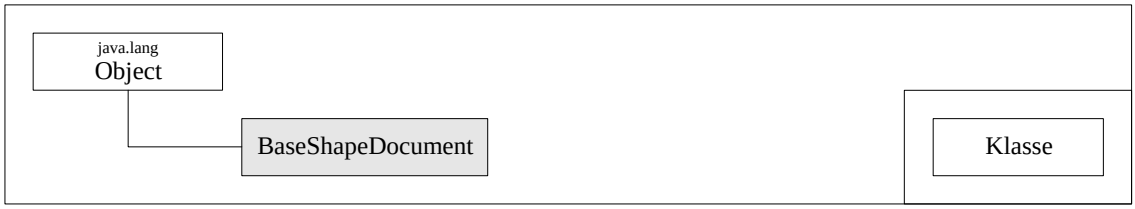
de.flashweather.io.grib

Die Klassen dieses Pakets werden für das Auslesen einer GRIB-Datei benötigt. Abbildung 5.4 auf Seite 90 veranschaulicht, wie sich aus den Klassen eine GRIB-Datei zusammensetzt.



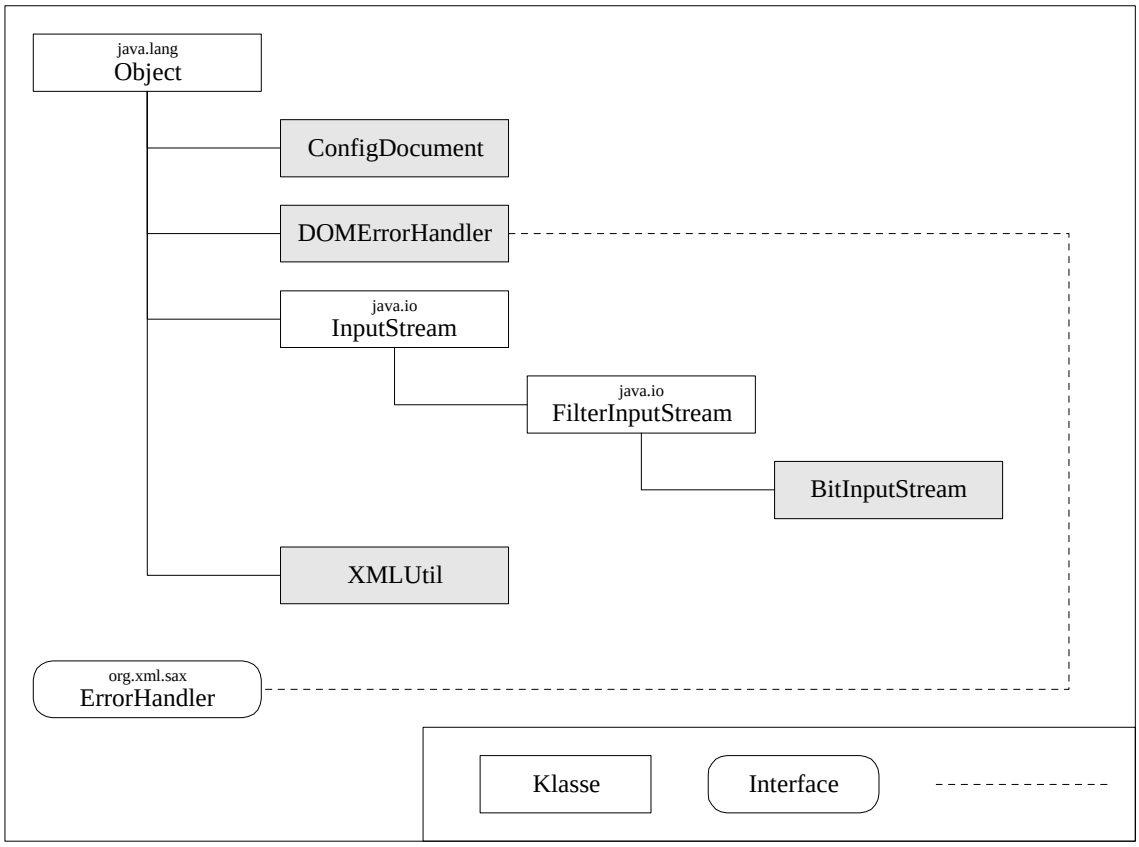
de.flashweather.io.shape

Subklassen der einzigen Klasse dieses Pakets bilden die Schnittstelle zwischen XML und Java-Grafikobjekten und repräsentieren Dateien, die in XML codierte Grafikobjekte enthalten.



de.flashweather.io.util

Dieses Paket enthält einige Hilfsklassen für die Ein- und Ausgabe von XML- und GRIB-Dateien zur Verfügung.



C XML-Formate

config.dtd

```
<?xml encoding="ISO-8859-1"?>

<!ELEMENT  config          (generalize|changevalues|vectorize|project|
                            clip|transform|translate|scale|rotate|
                            setbounds|setwidth|setheight)*>
<!ATTLIST  config          version       CDATA                    "1.0">

<!ELEMENT  generalize      EMPTY>
<!ATTLIST  generalize      cols          CDATA                    "1"
                           rows          CDATA                    "1">

<!ELEMENT  changevalues    EMPTY>
<!ATTLIST  changevalues    add           CDATA                    "0.0"
                           mult          CDATA                    "1.0">

<!ELEMENT  vectorize       EMPTY>
<!ATTLIST  vectorize       class                  CDATA           #REQUIRED
                           boundarydistance       CDATA           "0"
                           labeldistance          CDATA           "0"
                           isovalues              CDATA           #REQUIRED
                           cutoutinternalpolygons (true|false)    "true">

<!ELEMENT  project         EMPTY>
<!ATTLIST  project         class         CDATA                    #REQUIRED
                           type          (rotategrid|azimuthal)   #REQUIRED
                           refx          CDATA                    "0.0"
                           refy          CDATA                    "0.0"
                           rotaxis1      (x|y|z)                  "x"
                           rotangle1     CDATA                    "0.0"
                           rotaxis2      (x|y|z)                  "y"
                           rotangle2     CDATA                    "0.0"
                           rotaxis3      (x|y|z)                  "z"
                           rotangle3     CDATA                    "0.0">

<!ELEMENT  clip            EMPTY>
<!ATTLIST  clip            class         CDATA                    #REQUIRED
                           xmin          CDATA                    #REQUIRED
```

```
                        ymin        CDATA                   #REQUIRED
                        xmax        CDATA                   #REQUIRED
                        ymax        CDATA                   #REQUIRED>

<!ELEMENT  transform    EMPTY>
<!ATTLIST  transform    a11         CDATA                   "1.0"
                        a12         CDATA                   "0.0"
                        a13         CDATA                   "0.0"
                        a21         CDATA                   "0.0"
                        a22         CDATA                   "1.0"
                        a23         CDATA                   "0.0"
                        a31         CDATA                   "0.0"
                        a32         CDATA                   "0.0"
                        a33         CDATA                   "1.0">

<!ELEMENT  translate    EMPTY>
<!ATTLIST  translate    xtranslate  CDATA                   "0.0"
                        ytranslate  CDATA                   "0.0">

<!ELEMENT  scale        EMPTY>
<!ATTLIST  scale        xscale      CDATA                   "1.0"
                        yscale      CDATA                   "1.0">

<!ELEMENT  rotate       EMPTY>
<!ATTLIST  rotate       angle       CDATA                   "0.0">

<!ELEMENT  setbounds    EMPTY>
<!ATTLIST  setbounds    xmin        CDATA                   #REQUIRED
                        ymin        CDATA                   #REQUIRED
                        xmax        CDATA                   #REQUIRED
                        ymax        CDATA                   #REQUIRED>

<!ELEMENT  setwidth     EMPTY>
<!ATTLIST  setwidth     width       CDATA                   #REQUIRED>

<!ELEMENT  setheight    EMPTY>
<!ATTLIST  setheight    height      CDATA                   #REQUIRED>
```

map.dtd

```
<?xml encoding="ISO-8859-1"?>

<!ELEMENT  map        (part+)>
<!ATTLIST  map        type        (country|river|lake|city)  #REQUIRED>

<!ELEMENT  part       (point+)>
<!ATTLIST  part       id          ID                         #REQUIRED
                      type        CDATA                      #REQUIRED
                      label       CDATA                      ""
                      category    CDATA                      #IMPLIED>

<!ELEMENT  point      EMPTY>
<!ATTLIST  point      x           CDATA                      #REQUIRED
                      y           CDATA                      #REQUIRED>
```

weather.dtd

```
<?xml encoding="ISO-8859-1"?>

<!ELEMENT  weather        (date?, time?, area*,gridshape*)>
<!ATTLIST  weather        version      CDATA       "1.0"
                          id           ID          #REQUIRED
                          type         (tmp|tcdc)  #REQUIRED
                          label        CDATA       ""
                          level        CDATA       ""
                          unit         CDATA       "">

<!ELEMENT  date           EMPTY>
<!ATTLIST  date           day          CDATA       #REQUIRED
                          month        CDATA       #REQUIRED
                          year         CDATA       #REQUIRED>

<!ELEMENT  time           EMPTY>
<!ATTLIST  time           hour         CDATA       #REQUIRED
                          minute       CDATA       #REQUIRED>

<!ELEMENT  area           (point|labelpoint)+>
<!ATTLIST  area           id           ID          #REQUIRED
                          fill         CDATA       #REQUIRED
                          label        CDATA       "">

<!ELEMENT  point          EMPTY>
<!ATTLIST  point          x            CDATA       #REQUIRED
                          y            CDATA       #REQUIRED
                          value        CDATA       "0.0">

<!ELEMENT  labelpoint     EMPTY>
<!ATTLIST  labelpoint     x            CDATA       #REQUIRED
                          y            CDATA       #REQUIRED>
```

D Inhalt der CD-Rom

arbeit	**Diplomarbeit**
arbeit/latex	Latex-Quellen und Grafiken
arbeit/pakete	Tex-Pakete und Dokumentation
arbeit/pdf	Acrobat Reader PDF-Version
beispiel	**Anwendung der FlashWeather-Klassen**
beispiel/data	Ausgabedaten der Beispiele
beispiel/flash	Flash-Wetterfilme
classes	**FlashWeather-Klassen**
data	**Eingabedaten und Tools**
data/grib	GRIB-Wetterdaten vom DWD und NCEP
data/grib-tools	GRIB-Tool wgrib vom NCEP
data/shp	ESRI Shapefiles
data/shp-tools	shp2mapml und C-Bibliothek shapelib
docs	**Dokumentation**
docs/data	GRIB- und Shapefile-Dateiformat
docs/flashweather	FlashWeather-API
docs/java	Java 2 API
docs/swf	Flash Dateiformat Spezifikation
docs/w3c	XML, XSLT, DOM, SVG Spezifikationen
docs/wave	Saxess Wave Dokumentation
docs/xalan	Xalan-Java API
docs/xerces	Xerces-J API
jars	**Java-Pakete**

software	**benötigte Softwarepakete**
software/flash	Flash Netscape-Plugin für Linux
software/java	Java 2 SDK 1.3
software/swf-sdk	Macromedia SWF-SDK
software/wave	Saxess Wave Flash-Generator 0.5
software/xalan	Apache Xalan-Java XSLT-Prozessor
software/xerces	Apache Xerces-J XML-Parser

Erklärung

Hiermit erkläre ich, dass ich die Diplomarbeit selbstständig angefertigt und keine Hilfsmittel außer denen in der Arbeit angegebenen benutzt habe.

Osnabrück, den ....

(Unterschrift)