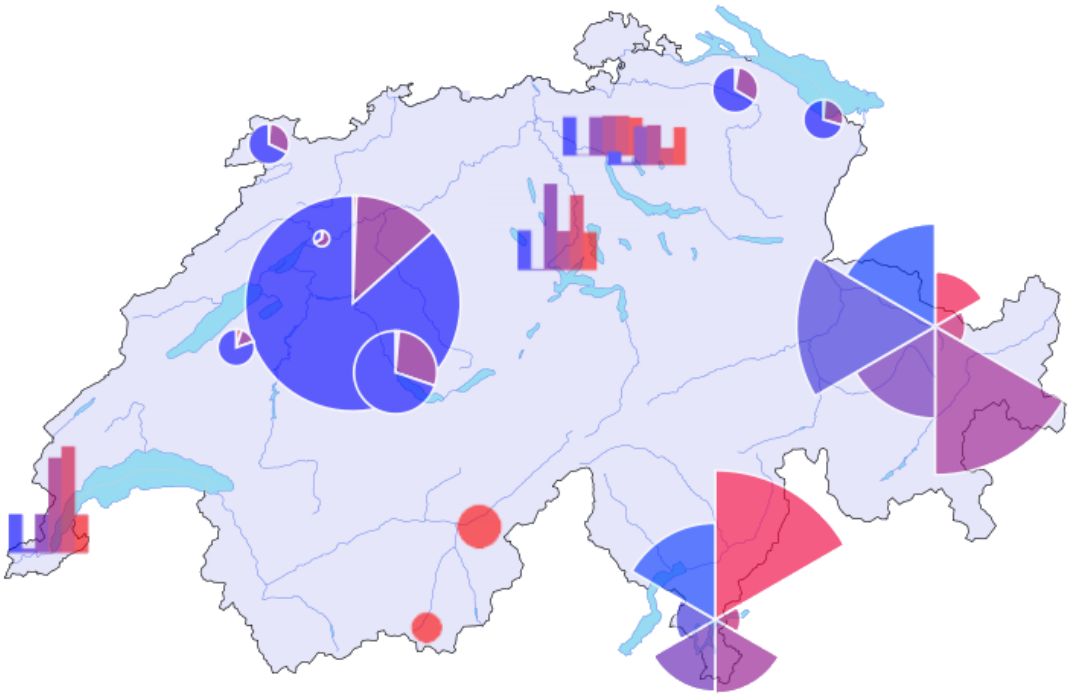
ETH Zürich
Institut für Kartographie

Wintersemester 05/06
Geomatik

Digital Diagram Designer



http://www.carto.net/schmid/diagram

Autor
Andreas Schmid
Lerchenhalde 37
8046 Zürich

Autor
Adrian Weber
Trottenstrasse 16
8037 Zürich

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Auf dem Gebiet der thematischen Kartographie sind Diagramme zentrale Elemente für unzählige Karten. Häufig sind Diagramme die einzige Möglichkeit, einen bestimmten Sachverhalt unkompliziert darzustellen. Es ist allerdings nicht immer einfach, die gewünschten Diagramme zu erzeugen. Besonders wenn eine grosse Anzahl Diagramme kreiert werden muss und diese zudem an der geografisch korrekten Position erscheinen sollen, kann das Erstellen und Platzieren der Diagramme rasch zu einer mühsamen Arbeit werden.

Um diese zu erleichtern, wurden für die Grafikprogramme Adobe Illustrator und Macromedia FreeHand am Institut für Kartographie der ETH Zürich Plugins programmiert, die neben anderen Funktionen die Möglichkeit bieten, einige Diagrammtypen automatisch zu erzeugen. Die so erstellten Diagramme liegen im jeweiligen Grafikformat vor und können dann mit dem Grafikprogramm weiterbearbeitet werden.

Im Rahmen der vorliegenden Arbeit sollen die Grundlagen für einen web-basierten Service zur automatischen Diagrammerzeugung erarbeitet werden: Ein Kartograph soll ausgehend von seinen Daten, die in Tabellenform vorliegen, mit Hilfe dieses Services Diagramme erzeugen lassen können, welche er sofort in seine Karte integrieren kann. Zur einfachen Weiterverarbeitung der Diagramme sollen diese in einem Raster- oder einem Vektorgrafikformat zurückgegeben werden. Weiter soll der Webservice die Möglichkeit bieten, automatisch erzeugte Diagramme direkt in eine Online-Karte einzubinden.
Dafür sollen in dieser Arbeit beispielhaft einige Diagrammtypen und Ausgabeformate programmiert werden. Der Service soll so offen konzipiert sein, dass er jederzeit um weitere Diagramme bzw. Ausgabeformate erweitert werden kann.

2 Konzept

2.1 Technik

2.1.1 Design

Die Auslegung des Webservice ist möglichst offen, damit Erweiterung keine Schwierigkeiten mit sich bringen. Um dies zu gewährleisten, werden die Operationen strikt getrennt und in einzelnen Klassen zusammengefasst. Die Klassen stehen in einem hierarchischen Verhältnis zueinander, was es ermöglicht, ihre Zusammenhänge untereinander einfach zu überblicken. Wenn ein neuer Diagrammtyp oder ein zusätzliches Ausgabeformat hinzugefügt werden soll, muss je nach geometrischer Grundform des Diagrammtyps ein standardisiertes Zwischenspeicherformat berücksichtigt werden. Dieses orientiert sich an einer Grafikbeschreibungssprache von Hansruedi Bär vom Institut für Kartographie der ETH Zürich [siehe Anhang A.4], wo definiert wird, was die jeweilige Form für Parameter verlangt. Für den Sektor sind dies die Zentrumskoordinaten, der Radius, der Startwinkel und der Endwinkelwert. Für das Rechteck werden die Koordinaten der linken oberen Ecke und die Breite und Höhe des Rechtecks verlangt. Diese Parameter werden ergänzt durch Füllfarbe, Strichfarbe und Strichdicke. Bei der Berechnung der Diagramme muss für jeden Sektor bzw. Balken

eine Zeile eines Arrays mit diesen Parametern geschrieben werden. Wenn die Methode aufgerufen wird, welche die Form zeichnet, werden dann diese Parameter übergeben.
Der Webservice wird mit der Skriptsprache PHP (PHP: Hypertext Preprocessor) programmiert.

2.1.2 Programmaufruf

Um den Service aufzurufen, wird lediglich eine korrekt formulierte URL benötigt, die alle erforderlichen Parameter enthält. Zusätzlich wird eine Internetseite gestaltet, welche dank einem Formular die Eingabe der Parameter erleichtert. Aus den Eingaben im Formular wird die entsprechende URL generiert.

2.1.3 Eingabedaten

Als Eingabedaten werden thm-Dateien vorausgesetzt. Dies sind ASCII-Dateien, in welchen die Diagrammdaten sowie die Koordinaten und der Name des Diagramms mit Tabulatoren getrennt abgespeichert sind. Sie können ohne weiteres mit jedem Tabellenkalkulationsprogramm generiert werden. Beim Programmaufruf muss der Ort dieser Datei als Parameter in Form einer URL angegeben werden. Dies bedingt, dass die Datei auf einem Webserver abgelegt ist. Falls der Benutzer diese Möglichkeit nicht hat, kann er auf einer Internetseite die Datei manuell hochladen.

2.1.4 Klassen

Im Skript `main.php` werden folgende Klassen (teilweise mit Unterklassen) eingebunden:

Parameter Hier werden die Parameter von der URL entgegengenommen und in Variablen abespeichert.

DataSet Diese Klasse liest die thm-Datei ein und speichert die Werte in einem mehrdimensionalen Array ab.

Chart Hier findet die Berechnung der Geometrie der Diagramme statt. Danach werden die Zeichenfunktionen aufgerufen.

GLib Diese Zeichenfunktionen sind hier definiert.

ToolBox Diese Klasse enthält diverse Hilfsmethoden wie z.B. die Berechnung eines Skalierungsfaktors für die Diagramme oder die Umrechung von Global- in Pixelkoordinaten.

2.1.5 Schnittstellen

Damit jeder Diagrammtyp in jedem Grafikformat ausgegeben werden kann, werden die Diagramme auf eine Grundform wie ein Rechteck oder einen Sektor reduziert. Die Parameter, die diese Formen benötigen, sind in einem Vorschlag für eine allgemeine Grafikbeschreibungssprache im ASCII-Format von Hansruedi Bär vom Institut für Kartographie der ETH Zürich definiert [siehe Anhang A.4].

Wenn der Webservice um einen neuen Diagrammtyp erweitert werden soll, muss die Beschreibung seiner Grundform nach bestimmten Vorgaben in einem Array abgespeichert werden. Erfordert der Diagrammtyp eine neue, noch nicht implementierte Grundform, müssen die Grafikklassen erweitert werden. Falls ein zusätzliches Grafikformat angeboten werden soll, müssen Methoden für das neue Format für alle vorkommenden Grundformen programmiert werden.

2.2 Umzusetzende Klassen und Methoden

Aufgrund des modularen Konzepts ist das Projekt etappierbar. Zunächst werden das Grundgerüst des Webservices erstellt und Klassen für Kreissektoren-, Flügel- und Säulendiagramme und für wertproportionale Kreise implementiert. Als Ausgabeformate sollen PNG, SVG, SVG-Gruppe und ASCII-Grafikbeschreibung umgesetzt werden.

3 Technische Umsetzung

3.1 Die objektorientierte Modellierung des DDD

Um den Digital Diagram Designer möglichst modular aufbauen zu können, wird dieser objektorientiert modelliert. Diese Modellierung soll garantieren, dass die Applikation später einfach erweitert werden kann.
Die Anwendung besteht im wesentlichen aus den drei Teilen Ausgabeformate, Diagramme und Parameter. Nach dieser Einteilung werden auch die verschiedenen Klassen modelliert. Die Modellierung nach dem UML-Schema ist in Abb. 7 im Anhang A.1 ersichtlich.
Eine Gruppe bilden die Diagrammklassen. Diese sind von der abstrakten Klasse CHART abgeleitet und heissen entsprechend ihrer Eigenschaften PIE-, WING- oder BARCHART. Diese Klassen umfassen die Methoden zur Berechnung und Zeichnung der Diagramme. Da in der abstrakten Klasse CHART diese Methoden als abstrakt deklariert sind, müssen diese von den abgeleiteten Klassen überschrieben werden. So kann sichergestellt werden, dass bei einer Erweiterung um neue Diagrammtypen, diese einfach eingefügt werden können. Somit stellt die Klasse CHART eine Schnittstelle für Erweiterungen zur Verfügung.
Dasselbe gilt auch für die abstrakte Klasse GLIB, von der die verschiedenen Ausgabebibliotheken abgeleitet sind. Wird ein neues Ausgabeformat hinzugefügt, muss diese Klasse die abstrakten Methoden aus GLIB überschreiben.
Die Klassen, welche die Parameter enthalten, bilden die dritte Gruppe. Die Superklasse PARAMETER enthält die allgemeinen Parameter sowie Methoden zum Lesen und Überprüfen dieser Parameter. Die von dieser Klasse abgeleiteten Subklassen enthalten die diagrammspezifischen Parameter und heissen dem zugehörigen Diagramm entsprechend PIE-, WING- oder BARPARAMETER. Werden neue Diagrammtypen hinzugefügt, muss ebenfalls eine neue von PARAMETER abgeleitete Klasse implementiert werden, welche die Parameter des neuen Typs liest und überprüft.
Nebst diesen drei Gruppen von Klassen gibt es noch die beiden einzelnen Klassen DATASET und TOOLBOX. In der Klasse DATASET sind die Daten und deren dazugehörige Methoden enthalten und in TOOLBOX stehen verschiedene Hilfsmethoden.
Die zentrale Datei MAIN.PHP enthält Kontrollstrukturen, um entsprechend dem gewünsch-

ten Diagrammtyp und Ausgabeformat neue Objekte der jeweiligen Klasse zu instanzieren.

Konkret ist der Digital Diagram Designer so organisiert, dass jede Klasse in einer eigenen Datei steht, wobei der Name der Datei dem Namen der Klasse entspricht.

3.2 Einlesen der Daten

Daten, die eingelesen werden, müssen in einer tab-getrennten ASCII-Datei vorliegen. In dieser Datei sind die Koordinaten, die ID und die Werte gespeichert. Die Anordnung dieser Informationen lehnt sich an das thm-Format an, welches am Institut für Kartographie verwendet wird.
Jede Zeile repräsentiert ein Diagramm. In den ersten beiden Spalten stehen die kartesischen X- und Y-Globalkoordinaten, in der dritten Spalte ist eine ID gespeichert, die allerdings nicht eindeutig sein muss, da diese im Programm nicht mehr weiter verwendet wird. Doch empfiehlt es sich für den Benutzer, diese eindeutig zu gestalten, um später die Diagramme unterscheiden zu können. Als ID ist z.B. eine Ortsbezeichnung naheliegend. In den nachfolgenden Spalten stehen schliesslich die thematischen Werte.

Eine thm-Datei kann einfach aus einer Tabellenkalkulation wie OpenOffice.org 1.1 erstellt werden. Dazu werden die Daten wie oben beschrieben in ein Tabellenblatt eingefügt. Anschliessend wird beim Speichern des Dokumentes das txt-Dateiformat und tab als Zeichentrennung gewählt. Als Endung muss manuell .thm eingegeben werden.
Es empfiehlt sich, die Daten bereits in der Tabellenkalkulation absteigend zu sortieren, da auf diese Weise in der Ausgabe grössere Diagramme weiter unten liegen zu kommen. Weiter ist darauf zu achten, dass am Ende der Datei keine Leerzeile steht.

3.3 Allgemeine Parameter

Die Parameter, die für jeden Diagrammtyp gelten, werden als allgemeine Parameter bezeichnet.
Als erstes gibt es obligatorische allgemeine Parameter, ohne diese keine Diagramme erzeugt werden können. Dazu gehören die Parameter betreffend den Diagrammtyp, das Ausgabeformat wie auch den Speicherort der Datei mit den thematischen Werten. Weiter sind die Parameter obligatorisch, welche die Ausdehnung des Ausschnittes in globalen Koordinaten und die Grösse des Ausgabebildes in Pixel angeben.
Dazu kommen noch die optionalen allgemeinen Parameter, mit welchen die Einfärbung der Diagramme beeinflusst werden kann. Wird kein optionaler Wert angegeben, wird der vordefinierte Standardwert übernommen.
Eine genaue Beschreibung der Parameter findet sich im Kapitel 4.1.

3.4 Kreissektordiagramme

Kreissektordiagramme (Pie Charts) vermitteln in gut verständlicher Weise den thematischen Sachverhalt und werden oft eingesetzt, sei das einzeln zur Illustrierung eines Textes oder als Gruppe in einer Karte.

Die Vorteile dieses Diagrammtyps liegen in der raumsparenden Form und der guten Zentrierung auf die Bezugsorte. Die Flächensumme der Sektoren zeigt die Summe der Werte. Die Flächen der einzelnen Sektoren vermitteln die prozentualen Anteile der jeweiligen Komponente [1]. Die Kreissektordiagramme sind flächenproportional.
Als spezifische Parameter für diesen Diagrammtyp kann ein Gesamtwinkel wie auch ein Skalierungsfaktor gewählt werden. Beide Parameter sind optional [Siehe Kapitel 4.1].
Mit dem Gesamtwinkel-Parameter können z.B. Halbkreissektordiagramme erstellt werden. Der Skalierungsfaktor kann benutzt werden, um die Grösse der Kreisscheiben zu verändern. Der Skalierungsfaktor wird mit dem Radius multipliziert, welcher so berechnet wird, dass die Fläche der Kreise einem Zehntel der Fläche des Ausgabebildes entsprechen.
Enthalten die thematischen Ausgangsdaten nur eine Spalte mit Werten, wird anstelle eines Kreissektordiagrammes ein wertproportionales Kreisscheibendiagramm gezeichnet.

3.5 Flügeldiagramm

Die Klasse WingChart, eine Unterklasse von Chart, ermöglicht es, Flügeldiagramme zu erzeugen. Nach [1] sind Flügeldiagramme Abarten der Kreissektordiagramme, da diese Typen gewisse Gemeinsamkeiten teilen. Diese Gemeinsamkeiten werden auch in der vorliegenden Arbeit deutlich. Im Besonderen sind dies die diagrammspezifischen Parameter, die dieselben sind.
So können für die Flügeldiagramme ein Gesamtwinkel und ein Skalierungsfaktor optional gewählt werden. Der vordefinierte Wert für den Gesamtwinkel ist 360 und für den Skalierungsfaktor eins.
Wie bei den Kreissektordiagrammen werden wertproportionale Kreisscheibendiagramme gezeichnet, falls die Ausgangsdaten nur eine Spalte mit Werten enthalten. In diesem Fall ist die Ausgabe identisch mit der Ausgabe als Kreissektordiagramme.
Die Methode computeDiagrams bezieht die Parameter aus der URL und fragt den Inhalt der thm-Datei ab. Auch die Plausibilität der angegebenen Eckkoordinaten wird hier überprüft. Anhand der Datenwerte in der thm-Datei, des automatisch berechneten Massstabfaktors und des vom Benutzer eingegebenen Massstabes wird der Radius der einzelnen Sektoren berechnet. Nach der Berechnung der Farbwerte jedes Sektors wird in einen mehrdimensionalen Array eine Zeile pro Sektor mit dessen geometrischen Daten (Zentrum, Radius, Füllfarbe, Strichfarbe und Strichbreite) geschrieben. Zusätzlich werden noch die Start- und Endwinkelwerte jedes Sektors ausgerechnet und ebenfalls in der jeweiligen Zeile abgespeichert. Die Methode drawDiagrams ruft schliesslich entsprechend dem gewählten Ausgabedatenformat die Methoden head, se bzw. ci bei Kreisscheiben und foot auf, welche die Diagramme schlussendlich Zeichnen.

3.6 Säulendiagramm

Das Säulendiagramm (engl. Bar Chart) ist eine einfach und schnell verständliche Diagrammform, welche wie auch die Kreissektordiagramme einzeln oder mehrere zusammen in einer Karte verwendet werden können. Mit Säulendiagrammen können im Gegensatz zu Kreissektor- und Flügeldiagramme auch negative Werte dargestellt werden.
Die Säulen sind längenproportional unabhängig ihrer Breite.

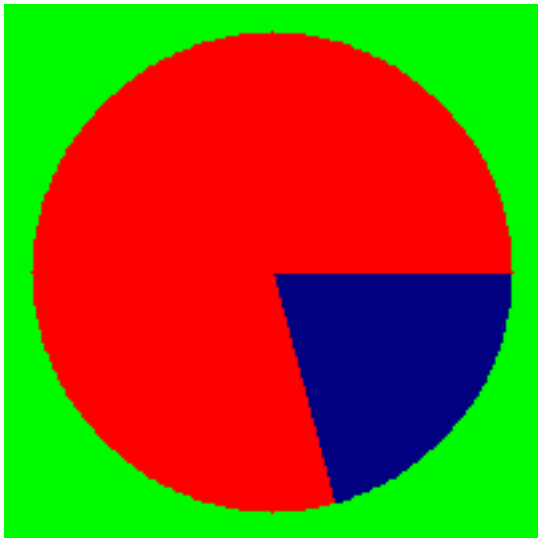


Abbildung 1: Beispiel für ein PNG-Rasterbild ohne Transparenz.

Die Skalierung der Höhe wie auch der Breite sind die beiden diagrammspezifischen optionalen Parameter [Siehe Kapitel 4.1]. Der Skalierungsfaktor für die Breite wird mit einem Fünfzigstel der Ausgabebildbreite multipliziert, was die Breite einer Säule ergibt. Die Höhe der Säulen wird mit dem Skalierungsfaktor für die Höhe verändert. Der Standardwert beider Skalierungsfaktoren beträgt eins.

3.7 Ausgabe als Rasterbild

Für die Ausgabe der Diagramme als Rasterbild wird das PNG-Format verwendet. PNG ist die Abkürzung für Portable Network Graphics. Dieses Format unterliegt keinen Patentbeschränkungen und ist in PHP in der GD-Bibliothek verfügbar. Zudem kann dieses Format Transparenz darstellen, was der wichtigste Vorteil gegenüber anderen Formaten in der GD-Bibliothek darstellt. Weitere Vorteile dieses Formats sind die verlustfreie Kompression und die weite Verbreitung. So wird dieses Format von modernen Web-Browsern und von zahlreichen Bildbearbeitungsprogrammen wie Adobe Photoshop oder The GIMP unterstützt.
Das Rasterbild, das zurückgegeben wird, hat einen transparenten Hintergrund. So kann dieses Bild abgespeichert und weiterverarbeitet werden. Typischerweise geschieht dies in einem Bildbearbeitungsprogramm, wo das Bild z.B. mit einer Pixelkarte zusammengefügt werden kann.

3.7.1 Untersuchung der PNG-Funktionen in der GD-Bibliothek

Die Ausgabe der Diagramme in eine PNG-Rasterdatei ist anfänglich unbefriedigend. Die Ränder der Diagramme sind zu stark verpixelt, wie in Abbildung 1 zu sehen ist. Diese schlechte Auflösung wirkt für das Gesamtbild ziemlich störend.

Es wird keine Möglichkeit gefunden, die geometrische Auflösung der Bildausgabe in der GD-Bibliothek zu erhöhen.
Um dennoch ein ansprechendes Bild zu erreichen, wird folgende Lösung im DDD implementiert: Das Bild wird zuerst doppelt so gross gezeichnet und anschliessend mit der Funktion IMAGECOPYRESAMPLED aus der GD Bibliothek wieder auf die gewünschte Ausgabegrösse verkleinert. Wie in Abbildung 2 ersichtlich, ist die Darstellung gut. Als Nachteil zu erwähnen ist hier, dass diese Lösung längere Rechenzeiten braucht.

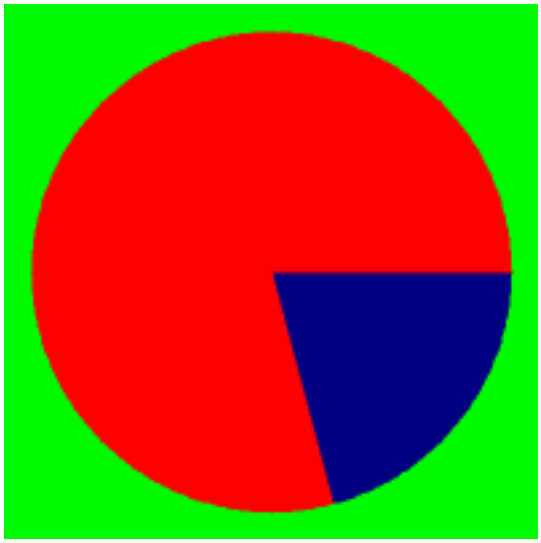


Abbildung 2: Beispiel für ein PNG-Rasterbild ohne Transparenz, das doppelt so gross gezeichnet und anschliessend auf die Ausgabegrösse verkleinert wird.

Es besteht die Idee, dass der Benutzer nicht nur die Farben für die Diagramme wählen kann sondern auch deren Transparenz. Da das PNG-Format Transparenz darstellen kann, wird diese Möglichkeit ausprobiert. Die Resultate fallen ziemlich ernüchternd aus. In keiner der oben beschriebenen Darstellungsweisen ist die Darstellung von transparenten Farben befriedigend. Anstelle von Transparenz entsteht ein unschönes Muster in der Farbe [siehe Abb. 3 und 4].
Somit ist es nicht möglich transparente Diagramme zu erstellen. Die Darstellung von transparenten Diagrammen muss in einem nächsten Arbeitsschritt realisiert werden. Wird z.B. eine erzeugte PNG-Datei in einem Bildverarbeitungsprogramm weiterverarbeitet, kann in diesem die Ebene der Diagramme transparent gewählt werden. Dasselbe gilt für eine Weiterverwendung in SVG, wo dazugeladenen Bildern eine Transparenz zugeordnet werden kann.

3.8 Ausgabe als Vektordaten

Die Diagramme können dank der Klasse SVGLIB, einer Unterklasse von GLIB, als Vektordaten im SVG-Format (Scalable Vector Graphics) ausgegeben werden. Die SVG-Spezifikation ist eine Empfehlung des World Wide Web Consortium (W3C) für ein XML-basiertes

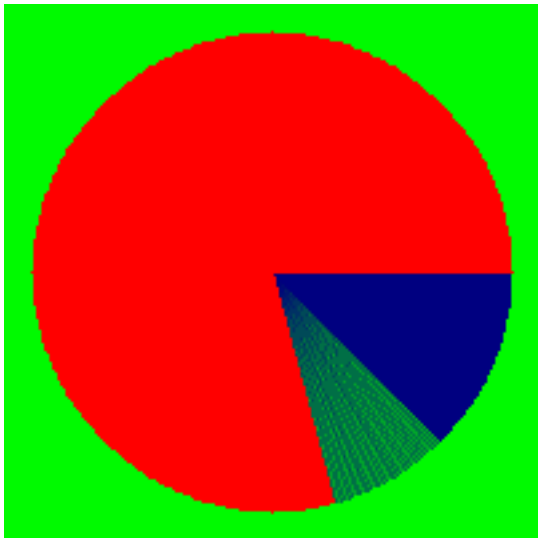


Abbildung 3: Beispiel für ein PNG-Rasterbild mit Transparenz.

Vektorgrafikformat. In der Klasse SVGLib sind die Methoden HEAD, welche den Typ „image/svg+xml“ der Ausgabedatei deklariert und die SVG-Headerinformationen ausgibt, FOOT, die das SVG-End-Tag ausgibt, sowie SE, CI und RE, welche einen Sektor, einen Kreis, bzw. ein Rechteck zeichnen, enthalten. Ein Sektor beispielsweise wird in SVG mit
`<path d="M1293.662,189.967 L1362.764,154.823 A77.526,77.526 0 0,1 1307.860, 266.182 z" fill="#ff9900" stroke="#FFFFFF" stroke-width="2.72"/>` beschrieben. Für SVG-Kartenprojekte, in welche von Digital Diagram Designer erstellte SVG-Diagramme eingebunden werden sollen, darf keine ganze SVG-Datei, sondern lediglich eine einzelne SVG-Gruppe zurückgegeben werden. Diesem Zweck dient die SVGGroupLib, die eine Unterklasse der SVGLib ist. Für die grafischen Elemente Sektor, Kreis und Rechteck ruft sie die entsprechenden Methoden ihrer Oberklasse SVGLib auf. Die Methode HEAD jedoch definiert den Dateityp als „text/xml“, da das Resultat nur eine XML-Datei, aber kein eigentliches Bild ist. Die Diagramme werden in einer Gruppe zusammengefasst, der ein Identifikationsattribut und als XML-Namensraum der SVG-Namespace zugeordnet wird, ohne dass ein SVG-Header mitgegeben wird:
`<g id="dddgroup" xmlns="http://www.w3.org/2000/svg">`
Die Angabe des Namensraums ist notwendig, damit ein SVG-Viewer die Daten nicht nur ins Document Object Model (DOM) einbindet, sondern sie auch grafisch darstellt. Die Methode foot schliesst die Gruppe wieder.

3.9 Ausgabe in einer Grafikbeschreibungssprache

Von Hansruedi Bär vom Institut für Kartographie der ETH Zürich wurde eine universelle Grafikbeschreibung mittels ASCII-Zeichen vorgeschlagen. Die Diagramme können von der Klasse ASCIIGLib auch in diesem Format ausgegeben werden. Die Beschreibung eines Sektors lautet in diesem Fall

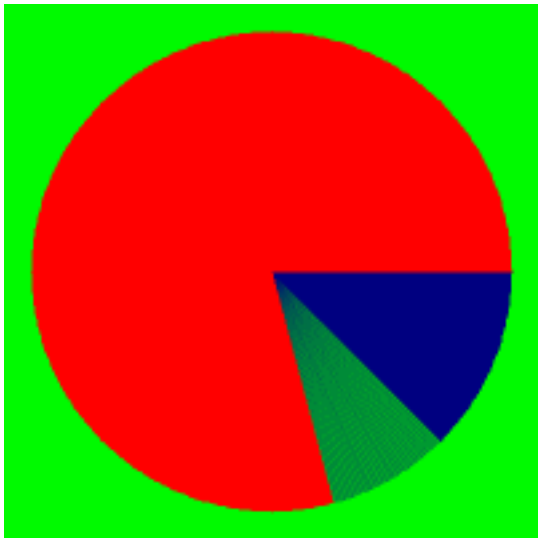


Abbildung 4: Beispiel für ein PNG-Rasterbild mit Transparenz, das doppelt so gross gezeichnet und anschliessend auf die Ausgabegrösse verkleinert wird.

```
pb
se 1293.6622222222 189.967464 77.52559053923 0 62.910918227501
pz
pe
af #ffff00
pf
as #FFFFFF
aw 2.72
ps
```

Auch dieses Format verlangt Headerinformationen wie z.B. die Bildgrösse in Pixel, die durch die Methode head geschrieben werden. Den Abschluss der Textdatei gibt die Methode FOOT aus.
Die Definition der ASCII-Grafikbeschreibung befindet sich im Anhang A.4.

3.10 Eingabeformular der Homepage

Zur Verbesserung der Benutzerfreundlichkeit kann Digital Diagram Designer über ein Formular auf einer HTML-Seite aufgerufen werden [siehe Anhang A.2]. In diesem Formular werden die Parameter in einzelne Felder eingetragen oder sie können mit Radio Buttons ausgewählt werden. Wenn die Schaltfläche zur Generierung der Diagramme betätigt wird, liest ein PHP-Skript die eingegebenen Werte ein, kreiert daraus die korrekte URL und leitet den Browser an diese weiter. Damit kommt der Benutzer ebenfalls zur grafischen Darstellung der Diagramme.

Falls der Benutzer seine thm-Datei nicht selber auf einem Webserver ablegen kann, hat er auf dieser Seite auch die Möglichkeit, sie auf den Server des Digital Diagram Designer hochzuladen. Der Upload an sich wird ebenfalls von einem PHP-Skript erledigt, das der Datei zusätzlich eine eindeutige Identifikationsnummer voranstellt. Damit wird verhindert, dass eine bereits hochgeladene Datei von einer neuen mit gleichem Namen überschrieben wird. Wenn der Vorgang erfolgreich war, wird der Browser zurück auf die Hauptseite mit dem Formular geleitet. Im Feld, wo die URL der thm-Datei eingetragen werden muss, steht dann der Pfad zur soeben hochgeladenen Datei.

4 Anleitung zum Gebrauch des Services

4.1 Inhalt der Online Help Page

Ein Bestandteil des Digital Diagram Designer Webservice ist eine Hilfeseite (`help.html`), wo kurz beschrieben ist, wie er angewendet werden kann. Hier findet sich auch eine Beschreibung der Parameter. In den folgenden Abschnitten findet sich der Inhalt der englischen Hilfeseite.

Quick guide Digital Diagram designer can either be used by calling its URL directly or by filling out a form. The use of the form is described in the Diagram Creation section below.
A sample URL call looks like this:
`http://www.foo.com/DDD/main.php?type=pieChart&output=png&`
`file=http://www.bar.org/data/myfile.thm&width=800&height=600&`
`originx=400000&originy=300000&destx=850000&desty=60000`
These parameters are required. Additionally you can append e.g. startcolor=FF0000, endcolor=00FFFF, angle=180 (pie chart and wing chart only), scale=2.4 (pie chart and wing chart only), barheightscale=1.8 (bar chart only) and/or barwidthscale=0.9 (bar chart only). For a description of the parameters see the Diagram Creation section below.

Input files Digital Diagram Designer needs a thm file in order to create diagrams. Thm files are simple ASCII files containing the thematic data. They can be created by database query tools or simply be exported from spreadsheet programs. A correct thm file looks like this:

```
600000  200000  Bern    1234    363     7950    3426    6859    2331
600324  238530  Thun    5738    412     9705    4738    8694    3948
650023  203214  Zuerich 3425    342     3456    3522    3526    3356
630023  223214  Uster   1125   1246     3456    3522    1526    3356
```

The first column contains the x, the second column the y coordinate value of a diagram, the third colum contains the name of the location or can be used as an identity key. From the fourth column on, the columns contain the absolute thematic values. The columns are separated by tabstopps.

Data file upload If your thm file is located on a webserver, you can skip the data file upload step. However, if you do not have the possibility to put the thm file on a webserver, you need to upload it manually on the Digital Diagram Designer server. This can be done by using the data file upload service on the Digital Diagram Desinger home page. After you click the Upload File button, the specified file will be uploaded on the Digital Diagram Designer server. Note that the maximum file size is 500KB. The browser will return afterwards to the Digital Diagram Designer home page. In the URL of thm file entry field the path to the uploaded file will be shown. During the upload the file will be renamed. The filename will start with an unique id followed by the original file name, such as 1392643ddc00c804e4data.thm.

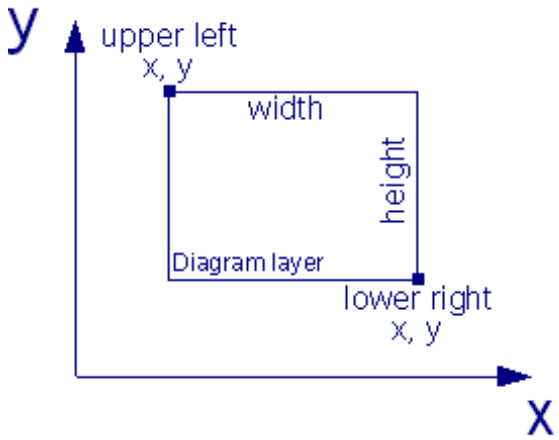


Abbildung 5: Diagram layer extent in global coordinates.

Diagram Creation Digital Diagram Designer needs several parameters in order to create your diagrams:

URL of thm file (parameter `file`):
Enter the URL where the thm file is located, like `http://www.foo.org/file.thm`. If you previously have uploaded your thm file on the Digital Diagram Designer server, the correct URL of the file is already filled in.

Diagram layer extent: Specify the extent of your diagram layer in global coordinates. See picture on the right for detailed information.

Upper left x (parameter `originx`):
The x coordinate value of the upper left corner of your diagram layer

Upper left y (parameter `originy`):
The y coordinate value of the upper left corner of your diagram layer

Lower right x (parameter `destx`):
The x coordinate value of the lower right corner of your diagram layer

Lower right y (parameter `desty`):
The y coordinate value of the lower right corner of your diagram layer

Diagram type and parameters: Specify the diagram type and some additional parameters.

Pie Chart (parameter `type`, value `pieChart`):
The output diagram type will be a pie chart

Wing Chart (parameter `type`, value `wingChart`):
The output diagram type will be a wing chart

Bar Chart (parameter `type`, value `barChart`):
The output diagram type will be a bar chart

Start color (optional parameter `startcolor`):
The hexadecimal color value of the first element of each diagram. The default value is FF0000 (red).

End color (optional parameter `endcolor`):
The hexadecimal color value of the last element of each diagram. The default value is FFFF00 (yellow).

Total diagram angle (optional parameter `angle`, pie chart and wing chart only):
The total angle in degrees of each diagram. The default value is 360.

Diagram area scaling factor (optional parameter `scale`, pie chart and wing chart only):
If the size of the diagrams that Digital Diagram Designer has created is not suitable for your map, you can specify an area scaling factor here. The default value is 1.

Bar width scaling factor (optional parameter `barwidthscale`, bar chart only):
If the width of the bars that Digital Diagram Designer has created is not suitable for your map, you can specify a bar width scaling factor here. The default value is 1.

Bar height scaling factor (optional parameter `barheightscale`, bar chart only):
If the height of the bars that Digital Diagram Designer has created is not suitable for your map, you can specify a bar height scaling factor here. The default value is 1.

Output format and size: Choose an output format for your diagrams and the size of the output image.

PNG (parameter `output`, value `png`):
The output will be a PNG raster image

SVG File (parameter `output`, value `svg`):
The output will be a Scalable Vector Graphics file

SVG Group (parameter `output`, value `svggroup`):
The output will be just a Scalable Vector Graphics group

ASCII (parameter `output`, value `ascii`):
The output will be an ASCII file

Width of the output image (parameter `width`):
Width in pixels of the output image

Height of the output image (parameter `height`):
Height in pixels of the output image. The height will be calculated automatically maintaining the aspect ratio given by the diagram layer extent values. It will be calculated while you are filling in a value into the width input box (Mozilla Firefox) or after the cursor has left the width input box (other browsers). If you wish an other height value than the proposed one simply fill out this input box after filling out the width input box.

Click the Create Diagrams button to start the diagram creation process. If you wish to cancel all the values filled in click the Reset button.

4.2 Bezugsquelle

Das Web-Formular, wo die Parameter eingegeben werden können, kann über die URL `http://www.carto.net/schmid/diagram` aufgerufen werden. Für einen direkten Aufruf des Webservices lautet die Adresse:
`http://www.carto.net/schmid/diagram/main.php`, gefolgt von einem Fragezeichen und den benötigten Parametern mit ihren jeweiligen Werten, die jeweils mit & aneinandergereiht werden.
Ein Beispiel:

```
http://www.carto.net/schmid/diagram/main.php?type=pieChart&
output=png&file=http://www.bar.org/data/myfile.thm&width=800&
height=600&originx=400000&originy=300000&destx=850000&desty=60000
```

5 Anwendungsbeispiel

Zur Illustration einer möglichen Anwendung des Digital Diagram Designer steht unter
`http://www.carto.net/schmid/diagram/demomap/demomap.svg`
eine SVG-Demonstrationskarte zur Verfügung, die Diagramme sowohl im PNG- als auch im SVG-Format integriert. Abbildung 5 zeigt eine Ansicht dieser Karte.
Das Einbinden der Diagramme geschieht interaktiv mit JavaScript. In der SVG-Datei muss für die Diagramme eine leere Gruppe vorhanden sein, wo das Bild später eingehängt werden kann. Für die PNG-Diagramme beispielsweise steht die Gruppe `<g id="PNGdiagrams"/>` zur Verfügung. Wenn man auf den Schriftzug Wing Chart (PNG) klickt, wird neben zwei Funktionen, die bereits eingebundene Diagramme zuerst entfernen, die JavaScript-Funktion `loadPNGDiagrams` aufgerufen. Als Parameter werden die Gruppe, wo die Diagramme eingehängt werden sollen, die Position und die Grösse des Bildes (in Globalkoordinaten), ein Transparenzfaktor und die Digital Diagram Designer URL, welche die entsprechenden Diagramme erzeugt, übergeben. Die angegebenen Koordinaten müssen mit den Koordinatenangaben in der URL übereinstimmen, das heisst, die Bildbreite in Globalkoordinaten

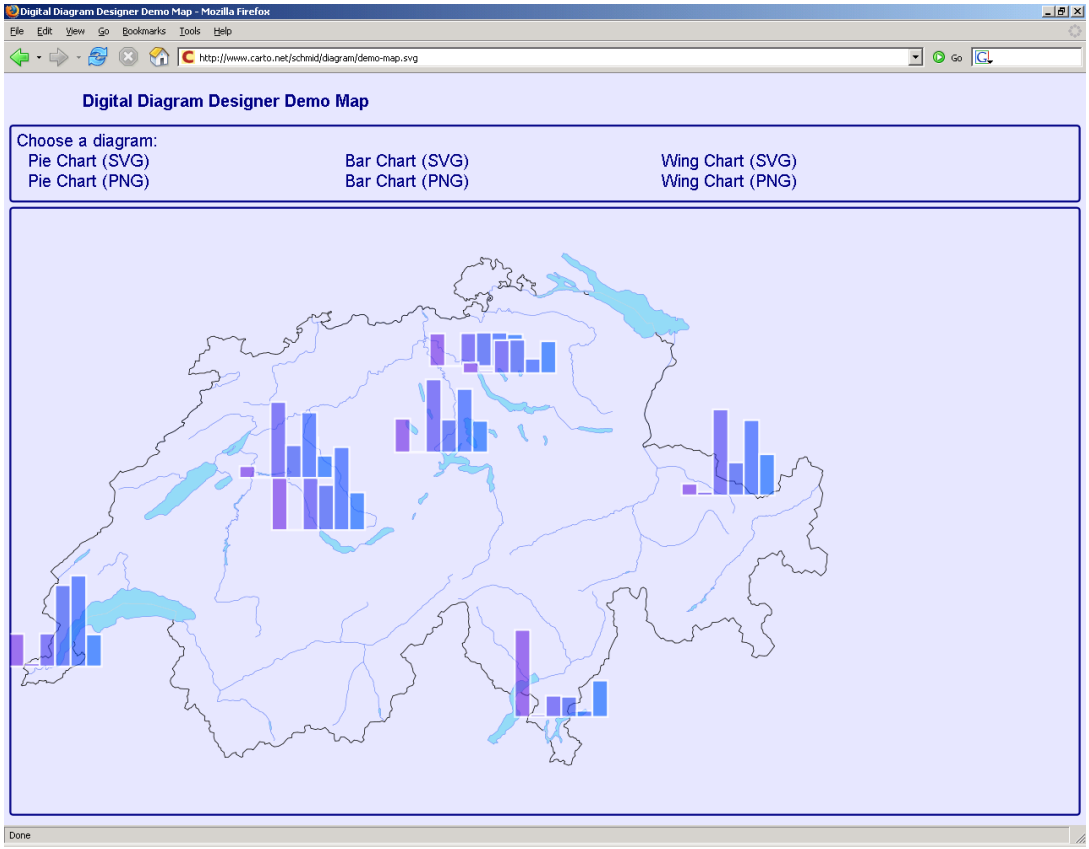


Abbildung 6: Screenshot des Anwendungsbeispiels demomap.svg

muss aus den Parametern `originx` und `destx` berechnet werden, damit der Diagrammlayer richtig dargestellt wird. Die Funktion `loadPNGDiagrams` erstellt ein neues `image`-Element anhand der übergebenen Parameter und bindet es in die SVG-Datei ein. Als Quelle des Bildes (Attribut `href`) wird die Digital Diagram Designer URL angegeben. Der folgende Abschnitt zeigt die Funktion `loadPNGDiagrams`:

```
function loadPNGDiagrams(targetGroupName,x,y,width,height,opacity,\\
diagramsLocation){
  //neue Diagramme einbinden:
  //Hinweis: In Mozilla Firefox muss im Fenster "Options" unter "Content"
  //"Load Images" ausgewaehlt sein. "for the originating website only"
  //darf hingegen nicht gewaehlt werden.
  var svgNS = "http://www.w3.org/2000/svg";
  var xlinkNS = "http://www.w3.org/1999/xlink";
  var newImage = document.createElementNS(svgNS,"image");
  newImage.setAttributeNS(null,"id","dddimage");
  newImage.setAttributeNS(null,"x",x);
  newImage.setAttributeNS(null,"y",y);
```

```
  newImage.setAttributeNS(null,"width",width);
  newImage.setAttributeNS(null,"height",height);
  newImage.setAttributeNS(null,"opacity",opacity);
  newImage.setAttributeNS(xlinkNS,"href",diagramsLocation);
  document.getElementById(targetGroupName).appendChild(newImage);
}
```

Die SVG-Diagramme werden auf eine andere Weise eingebunden. Als Ausgabeformat wird die SVG-Gruppe ausgewählt, die ins DOM (Document Object Model) der SVG-Karte integriert wird. Je nach SVG-Viewer bzw. Browser kommen hier unterschiedliche Funktionen zur Anwendung. In der Funktion `loadSVGDiagrams` wird entweder `getURL` (von Adobe SVG Viewer und Batik unterstützt) oder `XMLHttpRequest` (in Mozilla implementiert) aufgerufen. Nachdem diese Funktionen die Digital Diagram Designer URL angefordert haben, hängen im Fall von `getURL` die Funktionen `addGeomGetURL` und `addGeom`, im Fall von `XMLHttpRequest` die Funktion `addGeom` direkt das Resultat in die Struktur der SVG-Datei ein. Diese Funktionen sind im Anhang A.3.1 dargestellt.
In der URL werden die Werte so angegeben, dass die Diagramme in Globalkoordinaten erstellt werden (`width=840617.6, height=301472.8, originx=0, originy=301472.8, destx=840617.6, desty=0`). Dies hat allerdings den Nachteil, dass die Diagramme grösser sind, als wenn die gleichen Daten als PNG-Diagramme angefordert werden, was durch die Angabe eines Skalierungsfaktors in der URL korrigiert werden muss. Die Diagramme müssen in der SVG-Gruppe mit einem Transformationsattribut um den negativen `originy`-Wert verschoben werden, da der Ursprung des SVG-Koordinatensystems links oben, derjenige der Globalkoordinaten aber links unten liegt. Falls die Diagramme transparent sein sollen, kann dies ebenfalls in dieser Gruppe angegeben werden:
`<g id="SVGdiagrams" transform="translate(0,-301472.8)" opacity="0.6">`.
Es ist möglich, die SVG-Diagramme analog zu den PNG-Diagrammen in einem Pixelkoordinatensystem zu generieren und mit einer entsprechenden Skalierung und Transformation in der Gruppe, wo sie eingehängt werden, den Layer am richtigen Ort zu positionieren. Damit kann das Problem der unterschiedlichen Skalierungen von PNG- und SVG-Diagrammen in der Karte umgangen werden.

Die folgenden Ausschnitte aus dem Code der SVG-Karte illustrieren deren Umsetzung näher. Hier ist auch die Codierung der Sonderzeichen in der URL ersichtlich ('&' wird durch '&', '=' durch '=' codiert), die notwendig ist, da die URL Teil des Wertes eines SVG-Attributs ist.
Der Quellcode der Elemente, die Javascript-Funktionen aufrufen, ist im Anhang A.3.2 zu finden.
Nachstehend ist der Quellcode des SVG-Kartenfensters mit den leeren Gruppen für die Diagramme. Dieses Fenster wird in die Hauptkarte eingebettet.

```
<svg id="mainMap" x="6" y="124" width="786" height="576"
viewBox="480459.1 -301472.8 360158.5 231983.2">
  <g id="mainMapGroup" transform="translate(0,0)">
    <g id="gewaesser">
      <path id="fluss_12" d="M660038 -262206l32 254l54 285l41
```

```
      176l68 352l27 298l-10 68" />
      [...]
    </g>
    <g id="SVGdiagrams" transform="translate(0,-301472.8)" opacity="0.6"/>
    <g id="PNGdiagrams"/>
  </g>
</svg>
```

6 Ausblick

Das vorliegende Resultat dieser Arbeit bildet eine gute Grundlage, auf welcher aufgebaut werden kann. Die Anwendung ist modular, so dass einfach weitere Diagrammtypen und Ausgabeformate hinzugefügt werden können. Die einzelnen Diagramme werden korrekt berechnet und die verschiedenen Ausgaben funktionieren. Soweit darf das Resultat als Erfolg gewertet werden.

Nebst weiteren Diagrammtypen und Ausgabeformaten wäre es vorteilhaft, die nachstehenden Funktionalitäten zu verbessern oder ergänzen.
Die dringensten anstehenden Arbeiten betreffen den Server, auf welchem der Web-Service läuft. So sollte ein Sessions-Management implementiert werden, d.h. jedem Benutzer wird ein Verzeichnis mit einer eindeutigen Sessions-ID zugewiesen, wo er seine Daten abspeichert. Nach Beenden seiner Sitzung wird dieser Ordner wieder gelöscht. So können mehrere Benutzer Dateien mit gleichem Namen hochladen und diese Dateien werden automatisch wieder vom Server gelöscht.
Weiter sollte der Server die hochgeladenen Dateien auf ihren Typ überprüfen. Insbesondere muss dabei unterbunden werden, dass ausführbare Programme hochgeladen werden können, um so böswillige Angriffe auf den Server zu unterbinden.

Um eine erhöhte Benutzerfreundlichkeit zu erreichen, können die Fehlermeldungen des PHP-Skripts noch verbessert werden. Gibt der Benutzer jetzt z.B. ungültige globale Koordinaten als Parameter ein, gibt das Skript einen Fehler aus, ohne mitzuteilen, wo dieser liegt.
Momentan können nur positive kartesische Koordinaten eingegeben werden. Eine Erweiterung wäre die Möglichkeit, dass ebenfalls negative und geografische Koordinaten eingegeben werden können.

Weitere Verbesserungsmöglichkeiten sind eine verbesserte Farbauswahlmöglichkeit und das Gruppieren der SVG-Diagramme.
In der aktuellen Version des Digital Diagram Designer kann eine Anfangs- und eine Endfarbe gewählt werden. Dazwischen werden die benötigten Farben gleichmässig abgestuft. Die Gefahr darin besteht, dass dem Kartenleser eine thematische Abstufung wie bei einer Choroplethenkarte vorgetäuscht wird, die gar nicht vorhanden ist. Die Stärke der jetzigen Implementation ist die grosse Flexibilität, indem der Benutzer seine Diagramme farblich beliebig an seine Karte anpassen kann. Diese Flexibilität darf bei einer Neuimplementation

der Farbgebung nicht wesentlich eingeschränkt werden.
Bei einer Weiterverarbeitung der SVG-Diagramme wäre eine Gruppierung der Diagramme sehr nützlich. Vorläufig muss eine Gruppierung, wenn sie nötig ist, manuell vorgenommen werden.

Die beiden Autoren freuen sich über den gelungenen Prototypen und hoffen, dass die Arbeit am Digital Diagram Designer fortgeführt wird und dieser Service zahlreiche Benutzer für sich gewinnen kann.

Literatur

[1] Imhof, Eduard: Thematische Kartographie. Berlin: Walter de Gruyter 1972.

A Anhang

A.1 UML-Schema des DDD

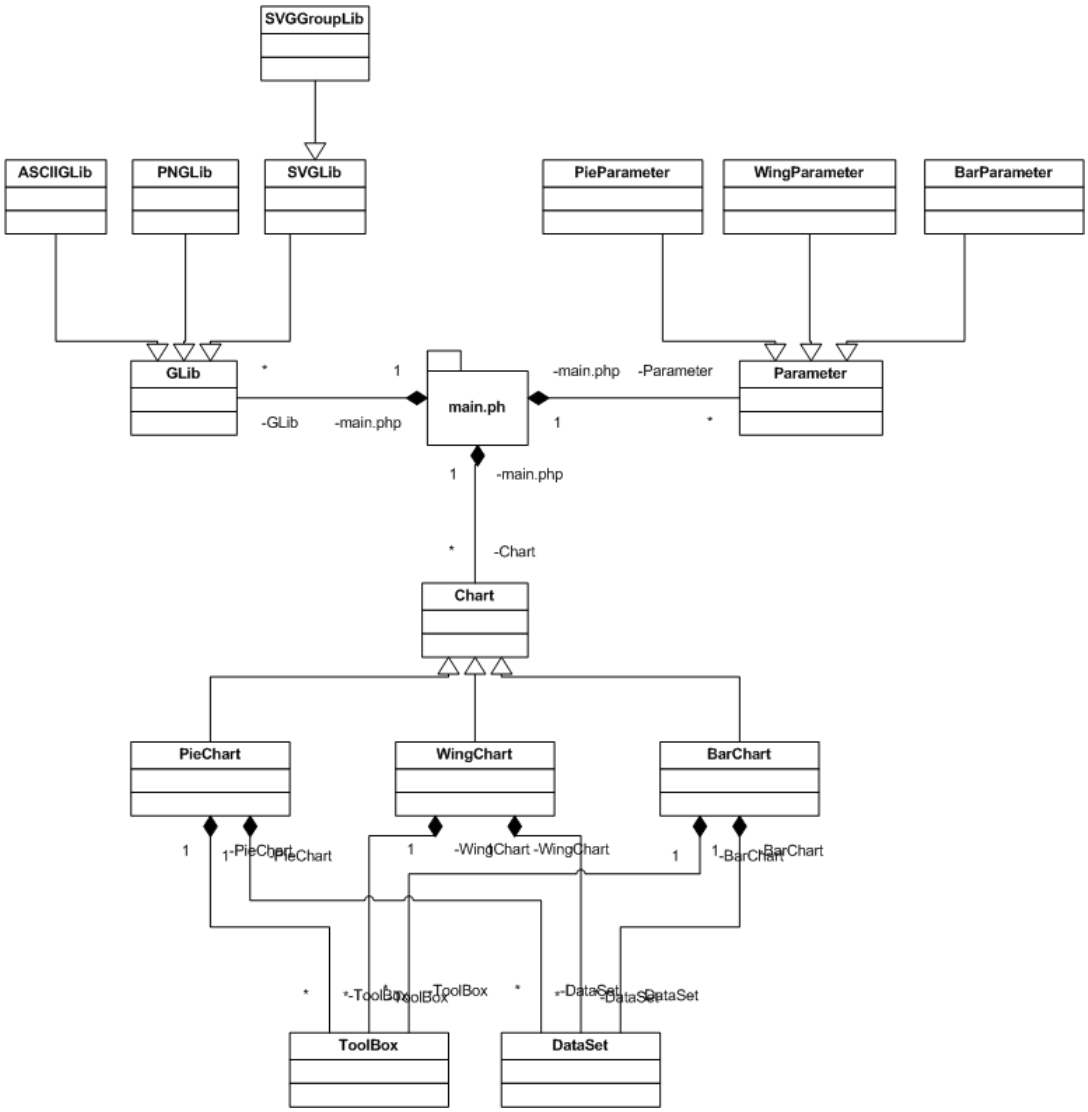


Abbildung 7: UML-Schema des Digital Diagram Designer

A.2 Screenshot http://www.carto.net/schmid/diagram

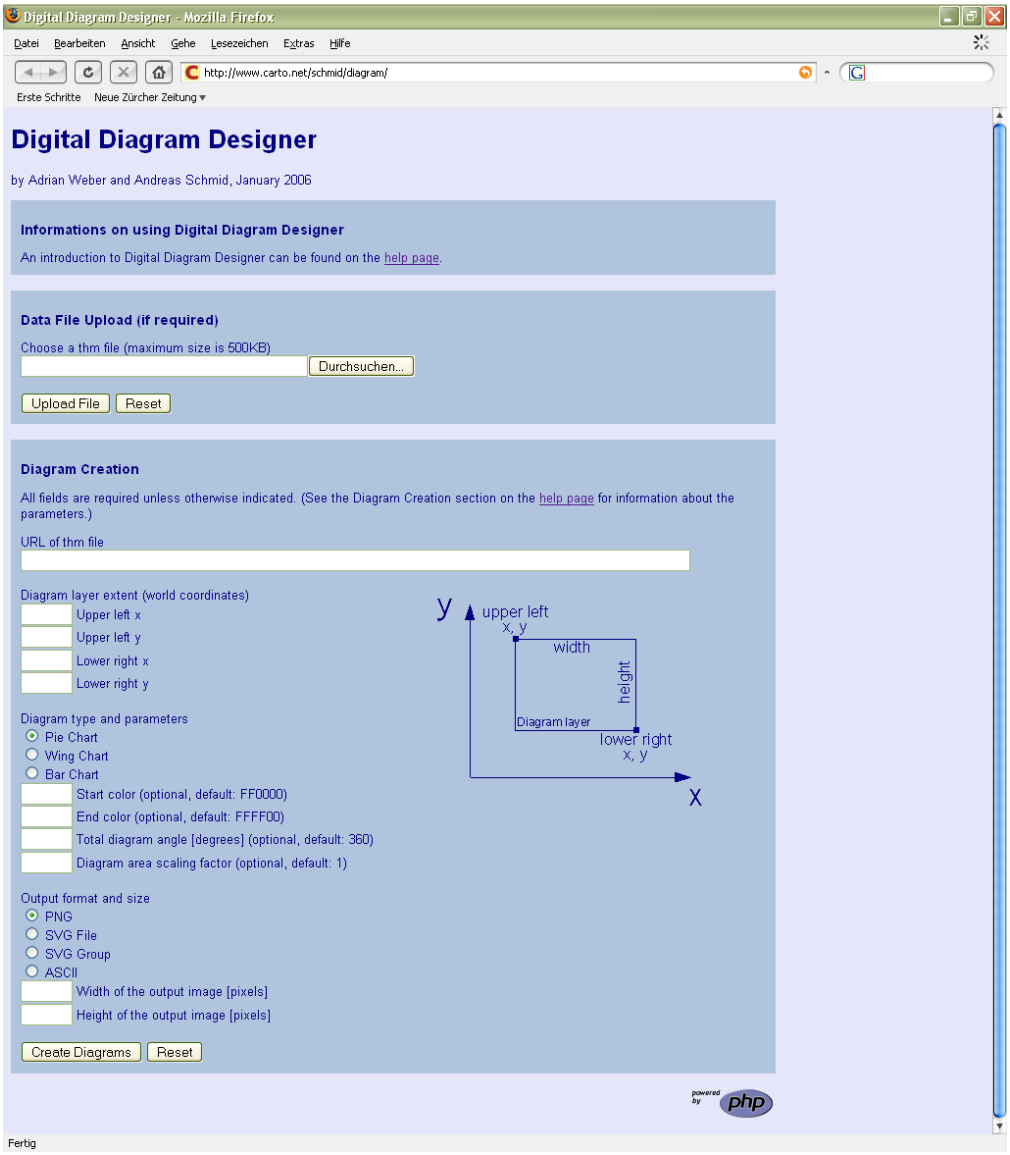


Abbildung 8: Screenshot http://www.carto.net/schmid/diagram

A.3 Quellcode aus dem Anwendungsbeispiel

A.3.1 JavaScript-Funktionen zum Laden der SVG-Diagramme

```
function loadSVGDiagrams(targetGroupName,diagramsLocation){
  targetGroup = targetGroupName;//globale Variable
  //neue Diagramme einbinden:
  //The following code and the functions addGeomGetURL and addGeom are
  //from http://www.carto.net/papers/svg/
  //postgis_geturl_xmlhttprequest/index.shtml:

  //call getURL() if available
  if (window.getURL) {
    getURL(diagramsLocation,addGeomGetURL);
  }
  //call XMLHttpRequest() if available
  else if (window.XMLHttpRequest) {
    //this nested function is used to make XMLHttpRequest threadsafe
    //(subsequent calls would not override the state of the request
    //and can use the variable/object context of the parent function)
    //this idea is borrowed from http://www.xml.com/cs/user/view/
    //cs_msg/2815 (brockweaver)
    function XMLHttpRequestCallback() {
      if (xmlRequest.readyState == 4) {
        if (xmlRequest.status == 200) {
         addGeom(xmlRequest.responseXML.documentElement);
        }
      }
    }
    var xmlRequest = null;
    xmlRequest = new XMLHttpRequest();
    xmlRequest.open("GET",diagramsLocation,true);
    xmlRequest.onreadystatechange = XMLHttpRequestCallback;
    xmlRequest.send(null);
  }
  else alert("Sorry, your browser/svg viewer neither supports
  window.getURL nor window.XMLHttpRequest!");
}

function addGeomGetURL(data) {
  //check if data has a success property
  if (data.success) {
    //parse content of the XML format to the variable "node"
    var node = parseXML(data.content, document);
    addGeom(node.firstChild);
  }
```

```
  else {
    alert("something went wrong with dynamic loading of geometry!");
  }
}

function addGeom(node) {
document.getElementById(targetGroup).appendChild(node);
//append new content
//Hinweis: targetGroup ist eine globale Variable
}
```

A.3.2 Elemente zum Aufruf der JavaScript-Funktionen

```
<g class="allText navigationText" cursor="pointer">
  <text x="22" y="85" onclick="removeDiagrams('SVGdiagrams');
  removeDiagrams('PNGdiagrams');loadSVGDiagrams('SVGdiagrams',
  'http://www.cart.net/schmid/diagram/main.php?file&#061;
  http://schmandr.pytalhost.net/sampledata/beschnachsektor.thm&#038;
  type&#061;pieChart&#038;output&#061;svggroup&#038;
  width&#061;840617.6&#038;height&#061;301472.8&#038;originx&#061;
  0&#038;originy&#061;301472.8&#038;destx&#061;840617.6&#038;
  desty&#061;0&#038;startcolor&#061;ff0000&#038;endcolor&#061;
  0000ff&#038;scale&#061;.5&#038;angle&#061;360')">
  Pie Chart (SVG)
  </text>
  <text x="322" y="85" onclick="removeDiagrams('SVGdiagrams');
  removeDiagrams('PNGdiagrams');loadSVGDiagrams('SVGdiagrams',
  'http://www.cart.net/schmid/diagram/main.php?file&#061;
  http://schmandr.pytalhost.net/sampledata/data2.thm&#038;
  type&#061;barChart&#038;output&#061;svggroup&#038;
  width&#061;840617.6&#038;height&#061;301472.8&#038;
  originx&#061;0&#038;originy&#061;301472.8&#038;
  destx&#061;840617.6&#038;desty&#061;0&#038;
  barheightscale&#061;0.003&#038;barwidthscale&#061;0.4&#038;
  startcolor&#061;6426da&#038;endcolor&#061;005aff')">
  Bar Chart (SVG)
  </text>
  <text x="622" y="104" onclick="removeDiagrams('SVGdiagrams');
  removeDiagrams('PNGdiagrams');loadPNGDiagrams('PNGdiagrams',
  480459.1,-301472.8,360158.5,231983.2,0.6,
  'http://www.cart.net/schmid/diagram/main.php?file&#061;
  http://schmandr.pytalhost.net/sampledata/data2.thm&#038;
  type&#061;wingChart&#038;output&#061;png&#038;
  width&#061;580&#038;height&#061;373.6&#038;originx&#061;480459.1&#038;
  originy&#061;301472.8&#038;destx&#061;840617.6&#038;
```

```
  desty&#061;69489.6&#038;startcolor&#061;FF0035&#038;
  endcolor&#061;0033FF&#038;scale&#061;3')">
  Wing Chart (PNG)
  </text>
</g>
```

A.4 ASCII Interface for Graphics Commands

```
Proposed ASCII Interface for Graphics Commands
Hansruedi Baer, baer@karto.baug.ethz.ch
2005-11-9

Graphics Command Definitions
- One line per command
- Commands are two-characater codes
- Codes and arguments are separated by single spaces
i: an integer number
f: a floating point number
h: a hexdecimal character [0-9,a-f]
c: single character
b: boolean [t|f] (true, false)
..: repeat

Missing
- text output
- coordinate transformation
- no raster images


General functions
>s i i                  set the screen size: width heigth (required)
>r i i                  set the image resolution [pixels per inch]:
                        x y (optional)
bc                      clear image buffer
bs                      write out image buffer (required)
>>                      start drawing (required)
<<                      finish drawing (required)
##                      ignored (comments etc.)

Path definition
pb                      begin path definition
pe                      end path definition
pz                      close a path

Path elements
```

```
mv f f                  move current postion: x y
ln f f                  draw line: x y
cq f f f f              draw a quadratic Bezier curve: x1 y1 x2 y2
cc f f f f f f          draw a cubic Bezier curve: x1 y1 x2 y2 x3 y3
ca f f f f f            draw an arc: x y radius fromangle toangle

re f f f f              add a rectangle to a path: x y width height
ci f f f                add a circle to a path: x y radius
se f f f f f            add a sector to a path: x y radius fromangle
toangle

Clipping
c+                      add current path to clip path
c-                      remove clip path

Attributes
af hhhhhhhh             set the rgb fill color: red/green/blue/alpha
as hhhhhhhh             set the rgb stroke color: red/green/blue/alpha
aw f                    set the stroke width: width
ad i f..                set the line dash: number dash gap dash ...
ao f                    set the dash offset:
ac c                    set the line cap: b|r|s (butt, round, square)
aj c                    set the line join: m|r|b: (miter, round, bevel)
am f                    set the miter limit: miterlimit
ar c                    set the fill rule: e|n (even-odd, non-zero)
aa b                    turn anti-aliasing on or off: t|f (on, off)

Drawing
pf                      fill the current path
ps                      stroke the current path
pa                      fill and stroke the current path

Gradients
pl hhhhhhhh hhhhhhhh    fill path using a linear gradient: color1 color2
   f f f f b b          x1 y1 x2 y2 ext1 ext2
pr hhhhhhhh hhhhhhhh    fill path using a radial gradient: color1 color2
   f f f f f f b b b    x1 y1 r1 x2 y2 r2 ex11 ext2 rel
```