Computergrafikpraktikum 18.08. - 05.09.2008

Oliver Vornberger, Johannes Emden
University of Osnabrück

Es soll ein Webportal zur Visualisierung von Immatrikulationsdaten der Universität Osnabrück implementiert werden. Als Technik kommen MySQL, Adobe Flex und Google Maps zum Einsatz.

Inhaltsverzeichnis

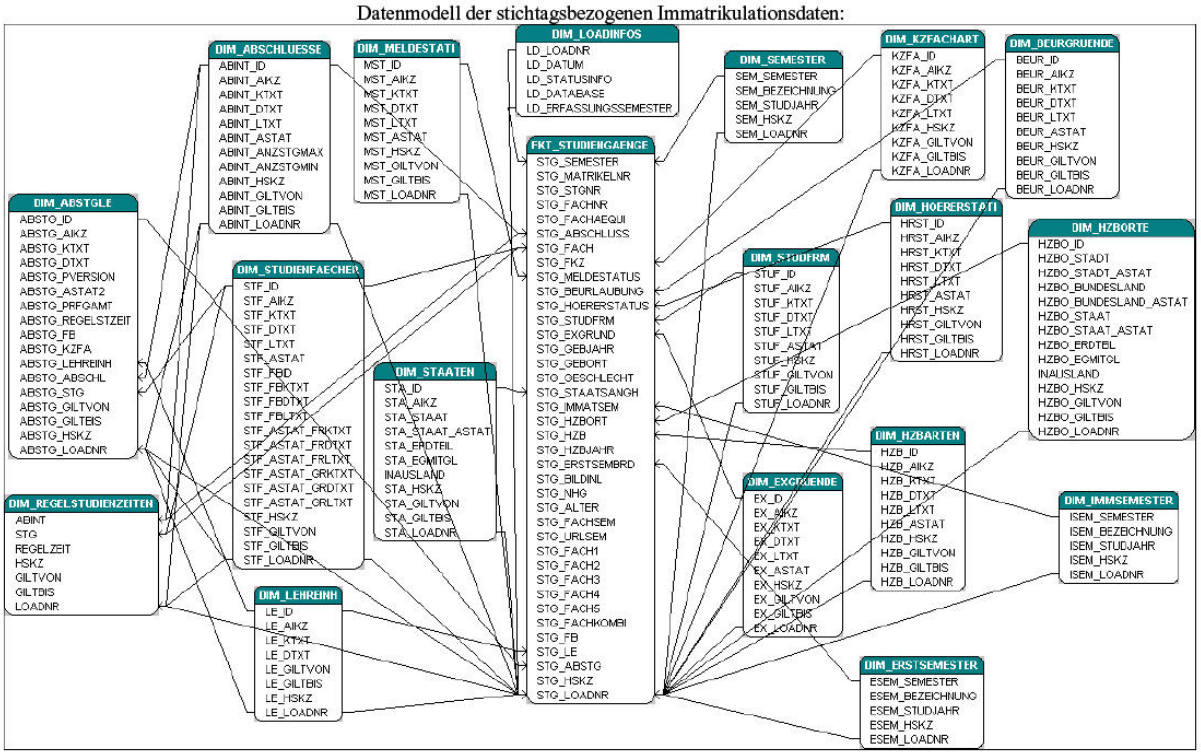
1. Gruppe Datenbank

Johannes Emden

Diese Dokumentation beschreibt Probleme und Lösungen, die bzgl. der Datenbank bei der Entwicklung der *Applikation zur Visualisierung von Immatrikulationsdaten* zu bearbeiten waren. Darüber hinaus wird dargestellt, in welcher Weise das Java-Backend mit der entwickelten Datenbank-Struktur interagiert.

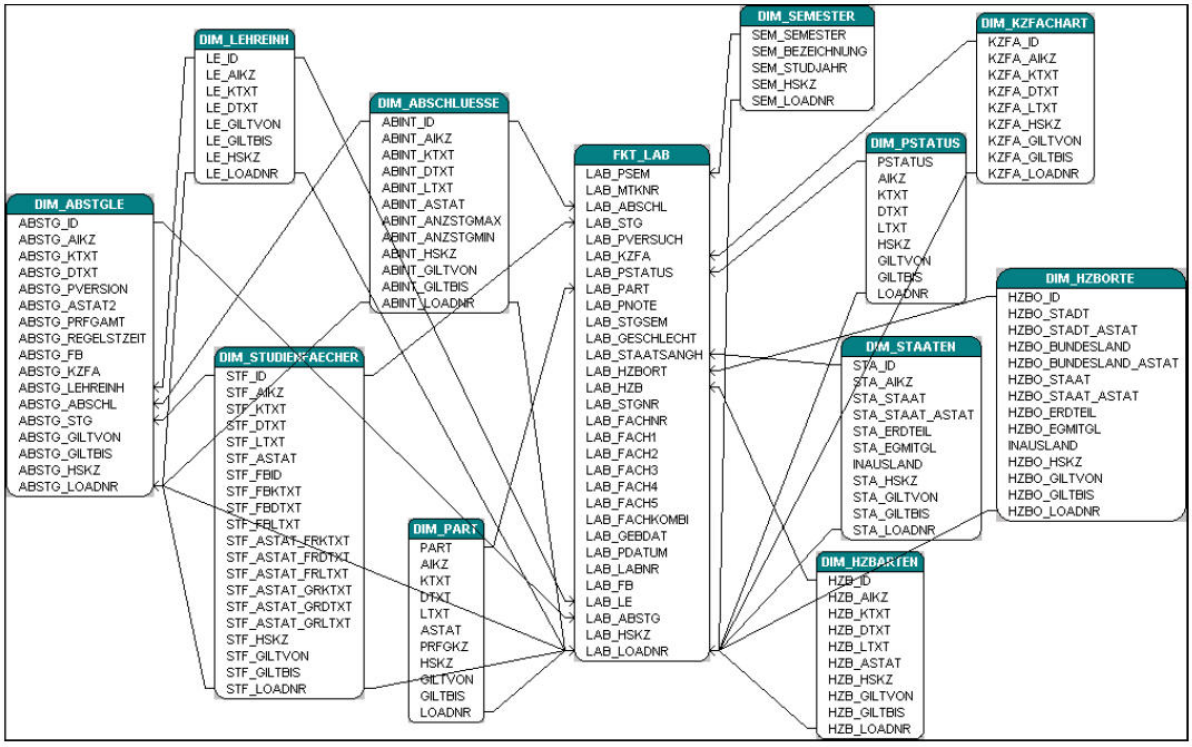
Ausgangssituation

Zu Beginn des *Computergrafikpraktikums 2008* stand ein aus dem operative System des Studentenwerks generiertes Data Warehouse zur Verfügung, in dem anonymisiert Studenten- und Absolventendaten seit dem Sommersemester 1995 abgelegt waren. Dieses Data Warehouse bestand im Wesentlichen aus zwei Faktentabellen, einer für die Studierenden mit ca. 460.000 Datesätzen und einer für die Absolventen mit ca. 26.000 Datensätzen. Darüber hinaus waren weitere Dimensionstabellen mit detailierten Informationen über die Fremdschlüssel aus beiden Faktentabellen vorhanden.



Faktentabelle der Studierenden und ihre Beziehungen zu den Dimensionstabellen

Datenmodell der Prüfungsdaten:



Faktentabelle der Absolventen und ihre Beziehungen zu den Dimensionstabellen

Finden einer geeigneten Datenbank-Struktur für das Live-System

Um eine geeignete Datenhaltung für die *Applikation zu Visualisierung von Immatrikulationsdaten* zu finden, mussten die Daten in eine Form gebracht werden, die den Ansprüchen der verschiedenen Anwendungsteile gerecht wurde und gleichzeitig die Queryzeiten möglichst kurz gestaltet um später dem Benutzer eine komfortablere Bedienung zu ermöglichen. In diesem Zusammenhang konkurrierte der Ansatz, eine effiziente Datenhaltung mit möglichst geringem Speicherverbrauch zu erstellen gegen eine redundante Speicherung mit je nach Anwendungsfall aggregierten Tabellen zu Gunsten der Live-Performance.

Als essentiell hat sich die Frage herausgestellt, ob bei einem Query nach der Anzahl an Studienfällen oder die der Studienköpfen gesucht war. Letztere ließ sich bei unveränderter Datenbank-Struktur nur in Zeiten jenseites einer Minute herausfinden, während die Suche nach Studienfällen auf einer eigends für diesen Fragetyp erstellte aggregierte Tabelle innerhalb von max. 3 Sekunden erfolgreich abgeschlossenn war. Die Frage nach Studienköpfen war vorallem bei Queries erwünscht, deren Ergebnis auf der Deutschland bzw. Weltkarte dargestellt werden sollten. Da die Berechnung der Studienköpfe je nach gewählten Filterkriterien viel Zeit in Anspruch nahm und auch die Anzahl aller Möglichkeiten, die Anzahl aller Permutationen, die vorhandenen Speicherresourcen weit überstieg, kam ein Ansatz zu Verwendung, der mittels einer Cache-Tabelle einmal berechnete Queries abspeicherte und somit beim erneuten Anfragen auf das bereits berechnete Ergebnis zurückgegegriffen werden kann. Diesem Ansatz liegt die Annahme zu Grunde, dass nach einem gewissen Zeitraum die Ergebnisse aller gängigen Anfragen bereits in der Cache-Tabelle liegen und nur selten neue Berechnungen erforderlich wären.

Überführungsskripte zur automatischen Erstellung der Datenbank

Die Datenbank sollte außerdem durch automatische Updates aus dem Rechenzentrum jeweils einmal im Semester auf den aktuellen Stand gebracht werden können, ohne das händische Änderungen nachträglich erforderlich gewesen wären. Hierzu war es notwendig Skripte zu schreiben, die über das Java-Backend alle

neuen Daten in das Live-System überführen sowie Transformation anstellen um die Datensätze der aggregierten Tabellen anzupassen.

2. Gruppe User Interface

Autoren: Markus Moormann & Matthias Sandbrink

Unter Benutzung von Adobe Flex erstellten wir das UserInterface (UI) für AVID[1], was alle anderen grafischen Komponenten vereint und zu einem Gesamtkonzept zusammen führt.

Unsere Aufgaben stellten sich daher im Wesentlichen in zwei Teilbereichen:

1. Ansprechende Gestaltung der Oberfläche in Form und Optik.
2. Funktionalität des UI durch Event-Handling.

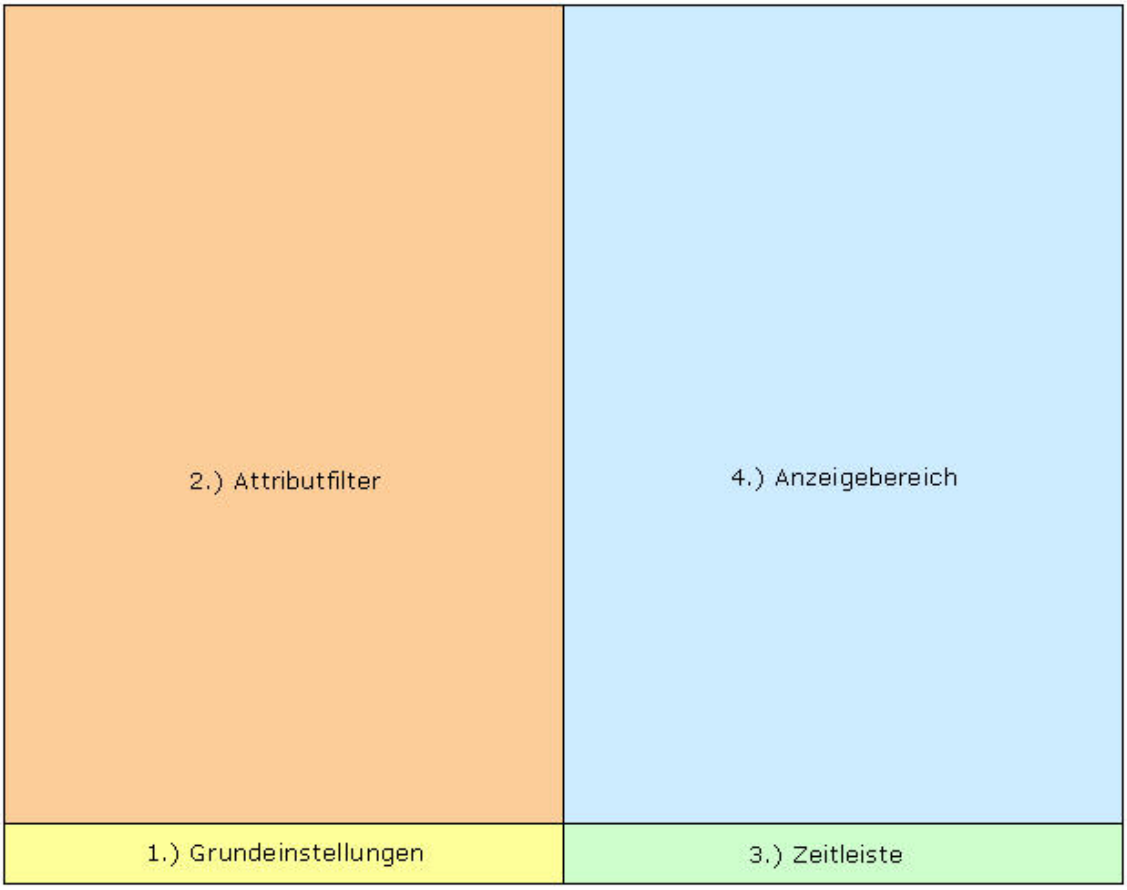
Leider lässt sich nicht auf jedes Detail extra eingehen, weswegen wir aus den beiden Teilbereichen jeweils ein Thema gesondert vertiefen, nämlich die Cairngorm-Struktur für die Funktionalität und das individuelle Anpassen einzelner Steuerungselemente in der Optik. Alles andere kann an dieser Stelle nur kurz Erwähnung finden. Aber in der Summe sollte klar werden, was am Ende alles hinter und in AVID steckt.

2.1 Aufbau

Autoren: Markus Moormann & Matthias Sandbrink

Zur Einarbeitung in das Thema des Praktikums begannen wir zunächst damit im Design-Modus von Flex erste Skizzen der Applikation zu entwerfen. Dabei orientierten wir uns an dem Layout-Schema aus der Aufgabenstellung, das eine Unterteilung in Attributfilter, in der die Suchanfrage spezifiziert werden kann, und Anzeigebereich zur Darstellung der Ergebnisse sowie Grundeinstellungen zwischen Studenten und Absolventen oder weiteren Optionen und eine Zeitleiste vorsah. Hier noch einmal der vorgeschlagene Ansatz aus der Aufgabenstellung:

1 Applikation zur Visualisierung von Immatrikulationsdaten



Die erste Realisierung umfasste für den Anzeigebereich eine Reihe von Tabs, mit denen man zwischen den Darstellungen der anderen Gruppen hin und herschalten konnte. Zur Verbesserung der Übersichtlichkeit wurden später für Flex Charting zwei angelegt; für Torten- und Säulendiagramm jeweils ein Reiter. Der Attributbereich zur Einschränkung der Suchanfrage wurde zunächst mit ComboBoxen für jeden Filter bis auf das Alter der Studenten oder Absolventen, was mit Hilfe eines NumericSteppers angegeben werden sollte, aufgefüllt. Unter weiteren Optionen hatten wir zum damaligen Zeitpunkt noch keine Vorstellungen, sodass lediglich noch ein Button, um die Suche schließlich auszuführen, sowie ein Standard-Slider für die Semesterzeitleiste hinzugefügt wurden und geschaffen war eine erste Arbeitsgrundlage auf der wir in den folgenden Tagen aufbauten.

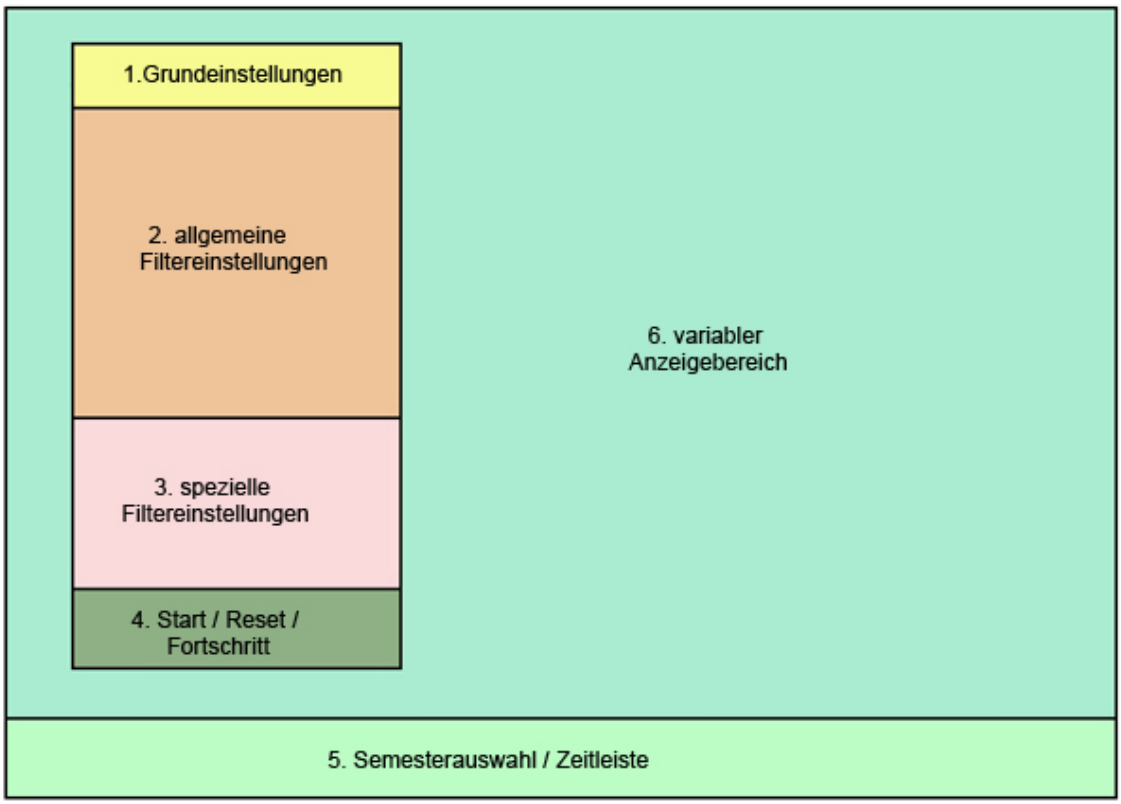
So war der Prototyp von AVID natürlich noch zu statisch und zu langweilig. Da Flex aber von sich aus eine bunte Vielzahl von Effekten zur Aufwertung und Auflockerung einer Applikation bereit stellt, griffen wir in den nächsten Wochen gerne darauf zurück. Darüber hinaus musste in ActionSkript für die Funktionalität von AVID und seinen Features gesorgt werden. Als Überblick ist hier die Kurzfassung der Liste unserer Aufgaben, die wir in den nächsten Tagen bearbeitet haben, einmal abgedruckt. Auf alles lässt sich leider nicht im Detail eingehen. Deswegen vertiefen wir auch nur die zwei umfangreichsten Punkte in gesonderten Kapiteln und handeln alles weitere kurz in folgender Liste ab:

- Um die Auswahl der Filter intuitiver zu ermöglichen, sollte der Nutzer zuerst die Entscheidung treffen ob eine Studenten- oder Absolventenstatistik visualisiert haben möchte. Deshalb wurde in der obersten Zeile des Attributbereiches eine ToggleButtonBar verwendet, mit der man effektvoll zwischen den einzelnen Elementen eines Viewstacks wechseln konnte. Auf diese Weise sind die spezifischen Attribute eines Absolventen wie z.B. Abschlussnote auch nur sichtbar und auswählbar, wenn oben Absolventen als Ziel markiert sind.
- Eine weitere Idee war die Vorstellung, dass der Attributbereich sich nach abgeschickter Suche einklappen lassen sollte, damit der Nutzer dem Anzeigebereich beim Präsentieren der Ergebnisse seine ganze Aufmerksamkeit zukommen lassen kann. Bei der Umsetzung benutzten wir einen Zoom-Effekt, der den Container des Attributbereiches auf die Breite null schrumpft, während der Container des Anzeigebereichs sich dazu dynamisch anpasst und immer die ihm maximal zur Verfügung stehende Breite voll ausnutzt.
- Damit AVID nach mehr als einer Spielerei mit dem Flexbaukasten aussieht, entschieden wir uns von bestimmten Bauteilen wie dem Slider oder der ComboBox eigene speziell auf unsere Bedürfnisse ausgerichtete Klassen abzuleiten. Bei unserer ExtendedComboBox sollte sich nur die Menuleiste der Auswahl der Länge

der verfügbaren Einträge anpassen, nicht aber die ganze ComboBox. Ähnliche Verbesserungen waren bei den Slidern von Nöten. Den *CustomComponents* widmen wir aber aufgrund des erhöhten Aufwands ein eigenes Kapitel.

- Mithilfe eines Preloaders sollte sicher gestellt werden, dass der User mit der Applikation erst in Kontakt kommt, wenn alle Anfangsdaten in den Filtern vollständig geladen sind und das Programm bereit ist.
- Außerdem war es unser Anspruch, dass sich die Auswahllisten hinter den ComboBoxen untereinander anpassen sollten, um zu gewährleisten, dass jede einstellbare Suche auch zu einem Ergebniss führt. Dies wurde auch umgesetzt, jedoch zum Teil wieder aufgehoben: Es sollten sich nur die ComboBoxen entsprechend anpassen, die vom User noch unberührt geblieben sind und standardmäßig "alle" ausgewählt hatten. Auf das MVC-Konzept, wonach wir der Applikation Leben eingehaucht haben, gehen wir ausführlich im nächsten Kapitel ein.
- Eine ProgressBar sollte dem User möglichst realistisch den Fortschritt beim Abholen der Daten in der Command-Kette vermitteln. Die Schwierigkeit dabei ist, dass man nicht weiß wieviele Daten insgesamt kommen werden, weshalb wir mit einer Schätzung (die Länge der Datensätze zuvor) arbeiten, die innerhalb der Kette dynamisch korrigiert wird, wenn sich die Länge einer Teilliste von der Länge der vorherigen unterscheidet.
- Jede ComboBox, außer die Auswahl des Geschlechts, ist mit einem RadioButton verbunden. Für den ausgewählten RadioButton für die zugehörige ComboBox deaktiviert, weil nun über den gesamten Filter gruppiert wird und dabei alle Listeneinträge beim Säulendiagramm mit einem Trefferbalken versehen entlang der waagerechten Achse aufgereit werden.
- Zudem sollte die Kopplung einiger Steuerungs-Elemente implementiert werden: Filter werden deaktiviert, wenn nach ihrem Attribut gruppiert wird. Slider und ComboBox zur Auswahl der Semester passen sich auch gegenseitig an, d.h. der Slider ist z.B. deaktiviert, wenn nach Semester gruppiert wird, oder bewegt sich zur Auswahl der ComboBox und umgekehrt. Versucht der Benutzer Aktionen an deaktivierten Elementen durchzuführen, wird ein Hinweis ausgesprochen.
- Nebenbei wurde eine Passwort geschützte Admin-Area entwickelt, wo Serverdaten eingesehen werden können.
- Als letztes Feature sollte eine Animation der Datensätze in den Diagrammen über die Zeit implementiert werden, um Veränderungen einer gezielten Suche über einen längeren Zeitraum darstellen zu können.

Diese Aufzählung verschafft einen soliden Überblick, was in AVID drinsteckt. Nachdem alle Punkte mehr oder weniger gelungen abgearbeitet worden waren, hat sich der schematische Aufbau von AVID im Vergleich zum Vorschlag aus der Aufgabenstellung natürlich drastisch verändert:



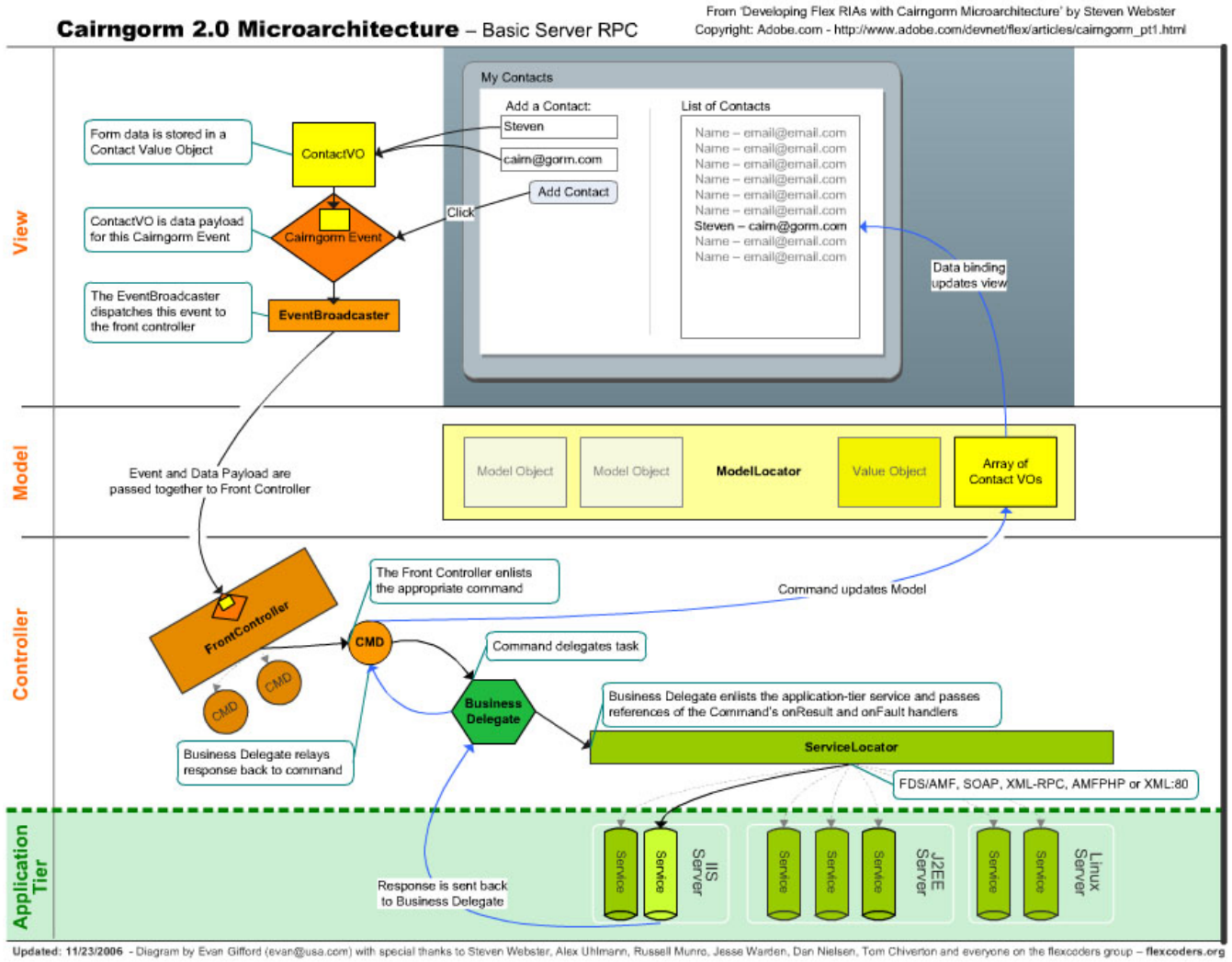
neues Layout

Der Anzeigebereich ist nun variabel und die Bedienung nach drei Wochen Auseinandersetzung mit dem Thema hoffentlich angenehm.

2.2 MVC mit Flex in AVID

Autoren: Markus Moormann & Matthias Sandbrink

Um bei größeren Projekten besser den Überblick behalten zu können benutzt man das MVC-Pattern[2]. Für Flex benutzt man hierzu hauptsächlich Cairngorm, welches eine Microarchitektur für dieses Pattern bereitstellt. Um das Verhalten und den Mechanismus besser erklären zu können sei an dieser Stelle zunächst auf folgendes Schaubild hingewiesen:



Cairngorm Technik

Hier ist zu erkennen, dass in der View, also auf der Oberfläche, ein Event geschieht, weil auf den Button "Add contact" geklickt wurde. Diesem Event wird eine Eigentschaft *contactVO* zugewiesen in welchem die eingegebenen Daten aus dem Formular beinhalten. Im weiteren wird das Event zum FrontController propagiert. Nun, auf der Controller-Ebene, 'weiß' der Frontcontroller welches Command es zu diesem Event ausführen

2 **M**odel, **V**iew, **C**ontrol

soll. Dazu implementieren alle Commands das *ICommand*-Interface und somit eine Funktion *execute*, welche vom Frontcontroller aufgerufen wird. Das aufgerufene Command kann nun wiederrum eine Verbindung zu einem externen Server aufbauen und zum Beispiel, wie in unserem Fall, eine Funktion im Tomcat aufrufen. Damit man die Antwort des Servers auf abfangen kann müsste das Command nun auch das *IResponder*-Interface implementieren, welches eine *result*- und eine *fault*-Methode bereitstellt. Die neuen Daten, welche man vom Server zurückbekommen hat schreibt man nun ins Model und über 'DataBinding' wird sofort die zugehörige Komponente aktualisiert, welches in diesem Fall die Liste mit den Kontakten darstellt.

MVC in AVID

Im folgenden wollen wir etwas genauer auf MVC in AVID eingehen, damit man den Ablauf des Programmes besser verstehen kann. Dazu brauchen wir wie oben ein VO[3], welches wir mit Daten füllen können. Da wir hauptsächlich vom füllen der Filter sprechen werden wurde dieses VO 'FilterVO' genannt.Hier ein kleiner Auszug aus der Klasse *FilterVO*:

```
1  [Bindable]
2      [RemoteClass(alias="cgp.vo.FilterVO")]
3      public class FilterVO
4      {
5
6          public var gender:String;          // null, "M", "W"
7          public var nationality:String;      // null, "LandXY"
8          public var state:String;           // null, "Deutschland"
9          public var county:String;          // null, "NDS"
10         public var faculty:String;         // null, "Mathematik"
11         public var superSubject:String;
12         public var firstStudy:String;      // null, "Info"
13         public var secondStudy:String;     // null, "Info"
14         public var degree:String;          // null, "Master"
15         public var age:Array;              // null, [15, 99]
16         public var semester:int;           // 19951
17         public var hzbTown:String;         // HZB-Ort
18         ...
19     }
```

Hier sieht man einige Eigenschaften, welche man über die Filter auswählen kann. Wie schon oben beschrieben wird dieses VO nun in ein Event gesteckt, welches wiederrum dispached wird. Anders als oben gibt es nun für jede ComboBox ein Event welches sie mit Inhalt füllt. Als Beispiel sei hier einmal das *GetNationalityListEvent*:

```
1  package cgprakt08.control.event.initial
2  {
3      import cgprakt08.control.event.ExtendedCairngormEvent;
4      import cgprakt08.model.vo.FilterVO;
5
6      import com.adobe.cairngorm.control.CairngormEvent;
7
8      public class GetNationalityListEvent extends ExtendedCairngormEvent
9      {
10         public static var GET:String = "query.GetNationalityList";
11         public var filterVO:FilterVO;
12
13        public function GetNationalityListEvent(filterVO:FilterVO, nextEvent:CairngormEvent
    = null)
14         {
15             super(GET);
16             this.filterVO = filterVO;
17             this.nextEvent = nextEvent;
18         }
19
20     }
21 }
```

Als Eigenschaften dieser Klasse sehen wir zunächst den String *GET* welcher für den Controller wichtig ist, damit er das Event einem Command zuordnenen kann. Dazu muss dieser String natürlich eindeutig sein. Als weitere Eigenschaft sieht man ein *nextEvent*, worauf wir später noch eingehen werden. Nachdem das jeweilige Event nun dispatched wurde nimmt es wieder den gewohnten Weg bis hin zum jeweiligen Command, in diesem Fall also das *GetNationalityListCommand*.

3 **V**alue**O**bject

```
package cgprakt08.control.command.initial
{
...
    public class GetNationalityListCommand extends SequenceCommand implements ICommand,
IResponder
    {
        override public function execute(event:CairngormEvent):void
        {
            var e:GetNationalityListEvent = event as GetNationalityListEvent;
            this.nextEvent    = e.nextEvent;
           ModelLocator.getInstance().preloader.downloadText.text = "Nationalitäten werden
 geladen";
            ModelLocator.getInstance().progressLabel = "Nationalitäten";
            var service:InitialDelegate = new InitialDelegate(this);
            service.getNationalityList(e.filterVO);
        }

        public function result(data:Object):void{

            var model:ModelLocator = ModelLocator.getInstance();
            ...
            [ArrayElementType("cgprakt08.model.vo.TupleVO")]
            var result:ArrayCollection = data.result as ArrayCollection;

            model.nationalityList = new ArrayCollection();

            model.nationalityList.addItemAt(TupleVO.getInstance(null, "alle"), 0);
            for(var i:int = 0; i < result.length; i++) {
                var t:TupleVO = result.getItemAt(i) as TupleVO
                if(t.value == "Deutschland") {
                    model.nationalityList.addItemAt(t, 1);
                    model.nationalityList.addItemAt(TupleVO.getInstance("seperator",
"------"), 2);
                }
                model.nationalityList.addItem(result.getItemAt(i));
                if(model.filterVO != null && model.filterVO.nationality == t.primaryKey)
                {
                    model.nationalityListSelectedIndex = model.nationalityList.length - 1;
                }
            }
            ...
            executeNextCommand();
        }

        ...

    }
}
```

Die *execute* Methode im Command wurde nun vom Controller aufgerufen. In dieser Funktion wird ein *InitialDelegate* erzeugt, welches sich um den Aufruf der ServerMethode kümmert. An diesem *InitialDelegate* rufen wir nun eine Funktion auf und übergeben ihr das *filterVO* aus dem übergebenen Event. Das *InitialDelegate* ruft nun seinerseits die Funktion am Server auf und übergibt das *FilterVO*-Objekt. Am Backend gibt es auch eine Klasse *FilterVO*; um das Mapping kümmert sich in diesem Fall BlazeDS. Ist diese Funktion am Server aufgerufen dauert es einige Zeit bis die Antwort vom Server zurückkommt und entweder die *result*-Methode bei erfolgreichem Aufruf oder die *fault*-Methode bei einem nicht erfolgreichen Aufruf aufgerufen wird.

In der Zwischenzeit wollen wir ein wenig näher auf die Bedeutung des oben genannten *nextEvent* eingehen. Da sich die ComboBoxen dynamisch ändern und sich bei jedem Ändern einer ComboBox alle anderen ändern sollen gibt es ein *SequenceCommand* welches das *GetNationalityListCommand* auch extended. Setzt man die Eigenschaft *nextEvent* in dem jeweiligen Command und ruft danach irgendwann die *executeNextCommand*-Methode auf, so wird automatisch das nächste Event dispached. Dies ermöglicht uns die Reihenfolge der Events schon vorher zu bestimmen. Da man dem Event als zweite Parameter im Konstruktor ein nextEvent mitgeben kann sieht ein verschachtelter Aufruf einer Event-Command-Kette beispielsweise so aus:

```
new GetNationalityListEvent(new FilterVO(),
    new GetStatesListEvent(new FilterVO(),
        new GetCountyListEvent(new FilterVO(),
            new GetHZBTownListEvent(new FilterVO(),
```

```
5                 new GetFacultyListEvent(new FilterVO(),
6                     new GetSuperSubjectListEvent(new FilterVO(),
7                         new GetDegreeListEvent(new FilterVO(),
8                             new GetFirstStudyListEvent(new FilterVO(),
9                                 new GetSecondStudyListEvent(new
  FilterVO()))))))))).dispatch();
```

Somit werden die Events von außen nach innen abgearbeitet.

Mittlerweile ist auch die Antwort des Servers da und es wurde die *result*-Methode aufgerufen. die Zeilen 23 - 31 des *GetNationalityCommand* wollen wir uns ein wenig genauer anschauen. Zunächst wird das result-Object, welches der Methode übergeben wurde in eine ArrayCollection gecasted. Zudem wird der ArrayCollection noch mitgeteilt von welchem Typ die enthaltenen Elemente sind. Dann wird eine neue ArrayCollection im model angelegt und diese mit Daten gefüllt. Dabei wird zunächst ein Element "alle" angelegt, welche der Standardauswahl entspricht. Danach wird die gesamte ArrayCollection durchlaufen und es wird in diesem Fall überprüft ob der Wert "Deutschland" vorkommt, damit diese noch einmal gesondert an oberer Stelle auftaucht. Nachdem nun die neuen Werte im Model stehen werden die graphischen Komponenten wieder mittels DataBinding geupdated, ohne dass wir etwas dazu tun müssen.

2.3 Erweiterte Komponenten

Autoren: Markus Moormann & Matthias Sandbrink

Schon nach kurzer Zeit wurde deutlich, dass einige der Standardkomponenten in Flex nicht unseren Ansprüchen genügten. Gerade der Slider zur Auswahl der Semester war einfach unpraktisch und nicht für unsere Zwecke verwendbar. Desweiteren waren die ComboBoxen teilweise zu schmal für den Text der in ihnen steht. Also mussten wir diese Komponenten anpassen, damit sie zu unseren Zwecken passten.

SemesterSlider

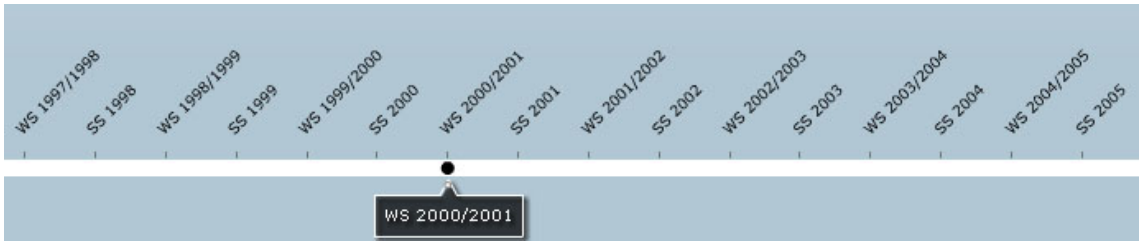
Zunächst möchten wir auf den Slider eingehen, der uns die meiste Arbeit abverlangt hat. Der einfach Slider besteht im wesentlich aus einem *Track*, dem Teil auf dem der *SliderThumb*, also der Button, den man ziehen kann sitzt.

StandardSlider

Da wir nun aber Semester an diesem Slider anzeigen wollen war dieser Slider einfach nicht praktikabel. Nach einiger Zeit der Suche im Internet fanden wir einem Blog einen Hinweis darauf, wie man das Problem lösen kann. Und zwar setzt man die Eigenschaft *sliderThumbClass* des Sliders auf eine selbsgeschrieben Klasse. Nun lässt sich der *SliderThumb* beliebig gestalten. In unserem Fall machten wir daraus ein runden Button mit einem Textfeld darunter in dem das aktuell ausgewählte Semester steht.

CustomSlider

Um die Übersicht bei der Auswahl des Semesters weiter zu verbessern sollte über den hier angezeigten Strichen, die die einzelnen Semester darstellen auch noch der Text des jeweiligen Semesters stehen. Um dies zu bewerkstelligen mussten wir eine komplett neue Komponente schreiben. Dieser *LabelTrack* wie wir ihn genannt haben besteht im erbt von der *UIComponent* und stellt damit den simpelsten Container für Grafikkomponenten dar. In diesen Container platzieren wir die Labels jeweils um 45° gedreht und erhalten folgendes abschließendes Bild für die Auswahl des Semesters:



Fertiger Slider

Um besser zu verstehen wie das platzieren der Labels funktioniert wollen wir einen kurz Blick auf den Quellcode werfen.

```
1  package cgprakt08.view.customComponents.slider
2  {
3      import cgprakt08.model.ModelLocator;
4      import cgprakt08.view.customComponents.slider.sprites.LabelSprite;
5
6      import mx.binding.utils.BindingUtils;
7      import mx.controls.Alert;
8      import mx.core.UIComponent;
9      import mx.events.ResizeEvent;
10
11     public class LabelTrack extends UIComponent
12     {
13         private var model:ModelLocator;
14
15         public function LabelTrack()
16         {
17             super();
18             model = ModelLocator.getInstance();
19             BindingUtils.bindSetter(drawLabel, model, "semesterListForSlider");
20             this.addEventListener(ResizeEvent.RESIZE, resize);
21         }
22
23         ...
24
25         public function resize(event:ResizeEvent):void
26         {
27             if(model.semesterList.length > 0)
28             {
29                 this.drawLabel(model.semesterList.toArray());
30             }
31         }
32
33         public function drawLabel(data:Array):void
34         {
35             while(this.numChildren > 0){
36                 this.removeChildAt(0);
37             }
38             if(data.length > 0) {
39                 var step:Number = this.unscaledWidth / (data.length - 1);
40                 for(var i:int = 0; i < data.length; i++) {
41                     var spr:LabelSprite = new LabelSprite(data[i]);
42                     spr.x = ((i - 0) * step) - 10;
43                     spr.y = 20;
44                     addChild(spr);
45                 }
46             }
47         }
48    ...
49
50     }
51 }
```

Hierbei gibt es zwei Klassen, die Klasse *LabelSprite* und die Klasse *LabelTrack*. Dabei stellt der *LabelTrack* praktisch die Pinnwand für die *LabelSprite*s dar. Die Funktion *drawLabel* bekommt ein Array mit den Daten der Semester übergeben. Nun wird dieses Array durchlaufen und für jeden Eintrag ein *LabelSprite* erzeugt und an die richtige Stelle gesetzt. Das *LabelSprite* an sich ist nur für den Text und seine Formatierung zuständig und ist eigentlich selbsterklärend.

```actionscript
package cgprakt08.view.customComponents.slider.sprites
{
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.text.TextFieldAutoSize;
    import flash.text.TextFormat;

    import mx.controls.sliderClasses.Slider;

    public class LabelSprite extends Sprite
    {
        public var slider:Slider;
        private var lbl:TextField = new TextField();
        [Embed(source="C:/WINDOWS/Fonts/VERDANA.TTF", fontFamily="Verdana")]
        private var verdana_string:String;
        private var _verdana_fmt:TextFormat;
        private var _text_txt:TextField;


        public function LabelSprite(text:String)
        {
            super();
            this._verdana_fmt = new TextFormat();
            this._verdana_fmt.font = "Verdana";
            this._verdana_fmt.size = 9;
            this._text_txt = new TextField();
            this._text_txt.rotation = -45;
            this._text_txt.embedFonts = true;
            this._text_txt.autoSize = TextFieldAutoSize.LEFT;
            this._text_txt.defaultTextFormat = this._verdana_fmt;
            this._text_txt.text = text;
            this.addChild(this._text_txt);
        }

    }
}
```

FilterSlider

Auch der Slider für das Alter, Fachsemester, Abschlussnote und die benötigten Semester entsprach nicht unseren Vorstellungen. Es sollte ein Wertebereich auswählbar sein, was mit der Standardkomponente zwar möglich ist, aber nicht wirklich übersichtlich ist, da die ausgewählten Werte nicht direkt angezeigt werden bzw. bleiben. Zu unseren Überlegungen gehörte zunächst einfach zwei Labels links und rechts neben den Slider zu legen um die aktuellen Werte darzustellen, was uns jedoch nicht genügte und wir somit nocheinmal einen neuen Slider anlegen mussten. Im Grunde genommen ähnelt der *FilterSlider* dem oben beschriebenen Slider. Der einzige Unterschied hier ist, dass im FilterSlider nun ein Dreieck gezeichnet wird, über welchem ein TextFeld mit dem aktuellen Wert angordnet wird. Zudem wird dieses Dreieck leicht durchsichtig gezeichnet falls der Slider deaktiviert ist, damit dieses auch wirklich deutlich wird.

FilterSlider für das Alter

ComboBoxen

Das Problem bei den ComboBoxen war, dass die festgelegte Breite der Boxen nicht dem Inhalt entsprach. D.h. das der Text der in den ComboBoxen steht teilweise einfach breiter als die ComboBox selbst war. Damit man nun nicht jeder ComboBox eine andere Breite zuordnet und damit das optische Bild doch sehr unruhig macht. Um dies zu realisieren mussten wir uns wieder eine eigene Komponente schreiben, die diesemal von *ComboBox* erbt.

```actionscript
package cgprakt08.view.customComponents
{
    import flash.events.Event;

    import mx.controls.Alert;
```

```
    import mx.controls.ComboBox;
    import mx.core.ClassFactory;
    import mx.core.IFactory;
    import mx.events.FlexEvent;

    public class ExtendedComboBox extends ComboBox
    {
        private var _ddFactory:IFactory = new ClassFactory(ExtendedList);

        public function ExtendedComboBox()
        {
            super();
        }

        override public function get dropdownFactory():IFactory
        {
            return _ddFactory;
        }

        override public function set dropdownFactory(factory:IFactory):void
        {
            _ddFactory = factory;
        }

        public function adjustDropDownWidth(event:Event=null, small:Boolean = false):void
        {
            this.removeEventListener(FlexEvent.VALUE_COMMIT,adjustDropDownWidth);

            if (this.dropdown == null)
            {
                callLater(adjustDropDownWidth);
            }
            else
            {
                var ddWidth:int =
this.dropdown.measureWidthOfItems(-1,this.dataProvider.length);
                if (this.dropdown.maxVerticalScrollPosition > 0)
                {
                    ddWidth += ExtendedList(dropdown).getScrollbarWidth();

                }
                if(!small)
                {
                    this.dropdownWidth = Math.max(ddWidth,this.width);
                }else {
                    this.dropdownWidth = this.width;
                }
            }
        }

        override protected function collectionChangeHandler(event:Event):void
        {
            super.collectionChangeHandler(event);
            this.addEventListener(FlexEvent.VALUE_COMMIT,adjustDropDownWidth);
        }


    }
}
```

Wie man sieht überschreiben wir die Funktion ***get** dropdownFactory*, damit wir hier unsere eigene *ExtendedList* verwenden:

```
package cgprakt08.view.customComponents
{
    import flash.events.MouseEvent;
    import flash.ui.Keyboard;

    import mx.controls.Alert;
    import mx.controls.List;
    import mx.controls.listClasses.IListItemRenderer;
    import mx.core.ClassFactory;

```

```
11
12      public class ExtendedList extends List
13      {
14          public function ExtendedList()
15          {
16              super();
17          }
18
19          public function getScrollbarWidth():int
20          {
21              var scrollbarWidth:int = 0;
22              if (this.verticalScrollBar != null)
23              {
24                  scrollbarWidth = this.verticalScrollBar.width;
25              }
26              return scrollbarWidth;
27          }
28
29
30          override protected function
    mouseEventToItemRenderer(event:MouseEvent):IListItemRenderer
31          {
32              var row:IListItemRenderer = super.mouseEventToItemRenderer(event);
33
34              if (row && row.data && row.data.primaryKey == "seperator")
35                  return null;
36              else return row;
37          }
38
39
40          override protected function moveSelectionVertically(code:uint, shiftKey:Boolean,
    ctrlKey:Boolean):void
41          {
42              super.moveSelectionVertically(code, shiftKey, ctrlKey);
43              if(code == Keyboard.DOWN && selectedItem.primaryKey == "seperator")
44              {
45                  caretIndex++;
46              }
47              if(code == Keyboard.UP && selectedItem.primaryKey == "seperator")
48              {
49                  caretIndex--;
50              }
51
52              finishKeySelection();
53          }
54
55
56      }
57  }
```

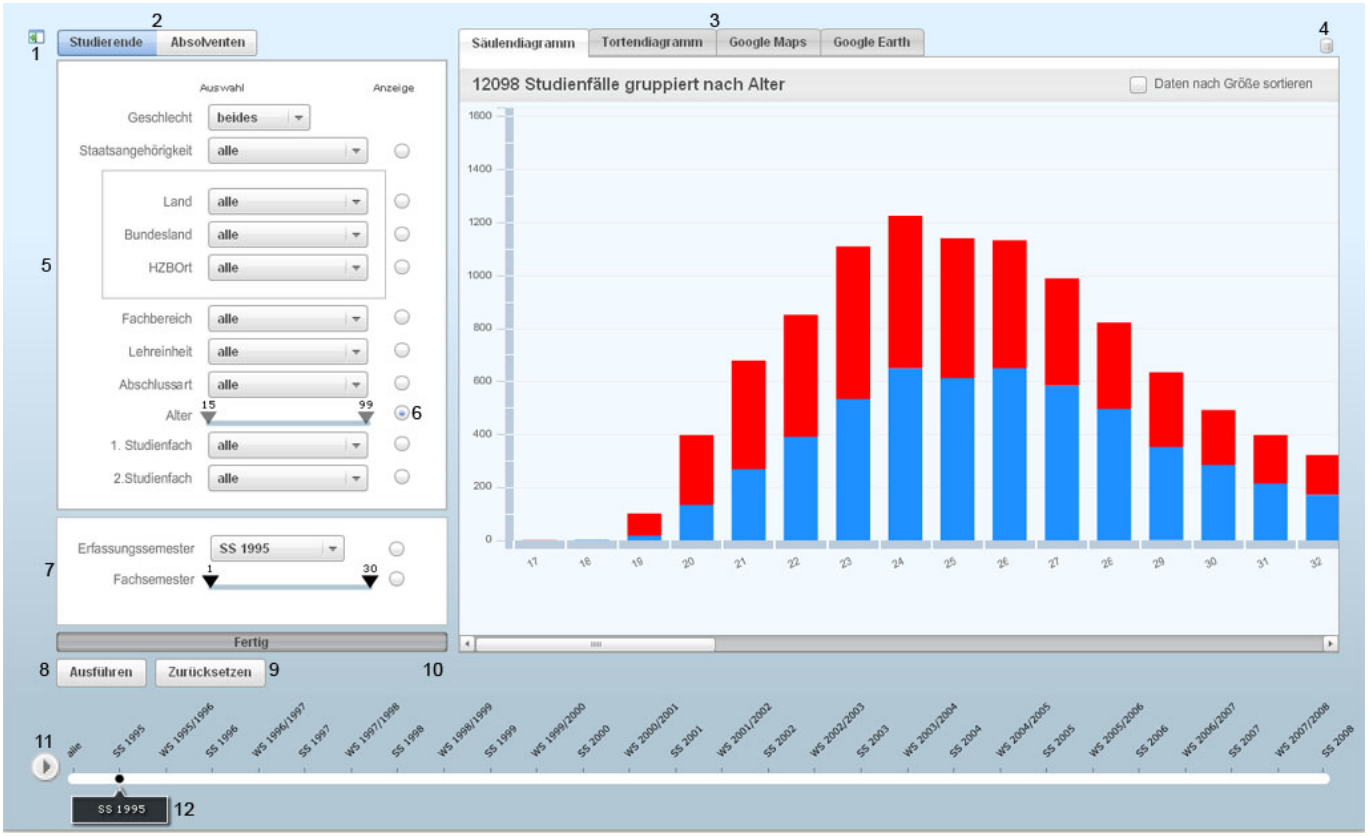
Die beiden überschriebenen Funktionen, die durch das Wort **override** gekennzeichnet sind, *moveSelectionVertically* und *mouseEventToItemRenderer* werden dazu benuzt zu verhindern, dass man den sogenannten *seperator* auswählen kann. Dieser stellt eine gestrichelte Linie dar, die beispielsweise die Auswahl "alle" und "Deutschland" in der Nationalitätenliste vom Rest abtrennt. Da es natürlich keinen Sinn macht diese auszuwählen muss geprüft werden ob das Objekt, auf welches wir gerade zeigen ein *seperator* ist. Die Funktion *getScrollbarWidth* ist die wichtige Funktion, die uns die Breite der Liste liefert. Wenn wir nocheinmal auf den Quellcode der *ExtendedComboBox* schauen so finden wir die Funktion *adjustDropDownWidth*, wo wir die Funktion *getScrollbarWidth* aufrufen und die *dropDownWidth* setzen. Fertig kompiliert und eingebunden sieht es so aus:

ExtendedComboxBox

2.4 Zusammenfassung

Autoren: Markus Moormann & Matthias Sandbrink

Zu guter letzt wollen wir nun nocheinmal kurz die wichtigsten Funktionen an einem Schaubild verdeutlichen:



AVID

1. Filter aus- oder einblenden
2. Filterart: Studenten oder Absolventen
3. AnzeigeTab: Säulen-, Tortendiagramm, GoogleMaps, GoogleEarth
4. Login zum Backend
5. allgemeine Filter
6. Gruppierung, in diesem Fall wird nach Alter gruppiert
7. spezielle Filter für Stundenten oder Absolventen

8. Ausführen: führt die Anfrage gemäß der Filter aus
9. Zurücksetzen: setzt alle Filter auf den Anfangswert zurück
10. Progressbar: zeigt den Ladestatus an
11. startet die Animation mit den gewählten Filtereinstellungen
12. SliderThumb: zeigt das aktuell gewählte Semester an

So war nach drei teilweise anstrengenden Wochen die Oberfläche von AVID fertiggestellt. Wir wünschen allen viel Spaß beim testen und spielen

3. Gruppe Statistik

von Benjamin Föcke und Quy-Manh Tsan

Diese Dokumentation zeigt die Überlegungen unserer Gruppe auf, die im Rahmen des Projektes "Visualisierung von Immatrikulationsdaten" mit der Teilkomponente Statistik betraut war, genauer der Visualisierung verschiedener studentischer Daten mittels Flex-Charting-Komponenten. Es werden zunächst allgemeine Vorüberlegungen aufgeführt, anschließend die aufgetretenen Schwierigkeiten bei der Umsetzung erläutert und die eingeschlagenen Lösungswege beschrieben, die letztendlich zum vorliegenden Endergebnis geführt haben.

Inhalt:

Allgemeine Vorüberlegungen

In den ersten Tagen galt es zunächst, sich einen Überblick über die anzuzeigenden Daten zu verschaffen und dabei zu überdenken, welche Flex-Charting-Komponenten zur Anzeige der entsprechenden Daten auf dem Bildschirm geeignet seien. Hierbei spielte die Überlegung, wie z.B. die Koordinatenachsen belegt werden sollten, eine große Rolle.
Wir wählten letztendlich zwei verschiedene Anzeigearten aus, und zwar:

- das Säulendiagramm,
- das Tortendiagramm.

Andere Statistiken wie z.B. das Blasendiagramm, das Liniendiagramm oder das Balkendiagramm schienen unzweckmäßig, vor allem weil schnell klar wurde, dass beim Einsatz aller Diagramme auf einer Achse stets die Anzahl der Studenten angezeigt werden sollte.
Nach dem Setzen von Filtern im Interface sollte dann noch eine Gruppierung gewählt werden können(z.B. Anzeige nach Semester), die die jeweils andere Achse repräsentieren sollte.
Die Darstellung eines zeitlichen Ablaufs per Liniendiagramm machte nur bei Gruppierung nach Alter Sinn. Auch wenn ein Balkendiagramm ebenso sinnvoll gewesen wäre wie das Säulendiagramm, so erschien es doch in der Ansicht zweckmäßiger, ein Säulendiagramm zu wählen, da Nutzer diese Art der Anzeige gewohnter sind und so an der y-Achse die Anzahl an Studenten angezeigt werden konnte, was uns intuitiver zu sein schien.
Das Tortendiagramm kam als alternative Anzeigemöglichkeit hinzu, da mit dieser Ansicht bestimmte Datenanzeigen plastisch und ästhetisch besser dargestellt werden können.
Unserer Meinung nach ergänzten sich die beiden Diagramme perfekt, der Nutzer könnte seine bevorzugte Anzeigeart wählen und zwischen diesen beliebig hin- und herschalten.

Die grundlegenden Aufbauten der beiden statistischen Anzeigearten wurden innerhalb der ersten Woche fertig gestellt. Hierbei mussten verschiedene Entscheidungen getroffen werden, wobei jeweils berücksichtigt werden musste, zum einen die Darstellung einfach zu halten und für den Standardnutzer verständlich zu belassen, zum anderen möglichst viele Informationen zu repräsentieren. Nachfolgende Beispiele sollen einige dieser Kompromisse und Überlegungen offen legen:

Zunächst war z.B. unklar, wie zwischen den beiden Diagramme gewechselt werden könnte. Anfangs hatte unsere Gruppe Statistik im Hauptinterface neben GoogleMaps und GoogleEarth einen eigenen Tab im Darstellungsfenster. Daher wurde der Wechsel zwischen den Diagrammen mittels eines Akkordeons implementiert. Von dieser Lösung wurde letztlich Abstand genommen, das Tortendiagramm und das Säulendiagramm erhielten jeweils eigene Tabs - dies verbesserte maßgeblich die Übersichtlichkeit und gleichsam die Bedienerfreundlichkeit.

Ein wichtiger Punkt war auch die Darstellung der Unterteilung der studentischen Daten in männlich und weiblich bzw. gesamt. Man entschied sich zunächst dafür, dass die Anzeige unerlässlich sei; daher sollten ab sofort nicht nur die Gesamttrefferdaten, sondern auch die Treffer unterteilt in männlich und weiblich an die View übergeben werden (einschließlich der x-Achse, für die ein eigenes Array übergeben wurde, sollten demzufolge nun 4 statt wie bislang 2 Arrays übergeben werden).
Die Anzeige zwischen männlichen und weiblichen bzw. Gesamtzahl an Studenten wurde besonders kompliziert bei der Darstellung im Säulendiagramm, da seitens Flex hier klare Grenzen gesetzt wurden. Mehr dazu soll im Abschnitt Säulendiagramm erläutert werden.
Beim Tortendiagramm war klar, dass eine Unterteilung eines Kuchenstücks in männliche und weibliche Anteile unsinnig war - die Darstellung wurde hier mittels ToolTip gelöst.

Zu guter Letzt war zu entscheiden, ob lediglich der absolute oder auch der relative Anteil angezeigt werden sollte.
Wir entschieden uns, in den Grafiken selbst den absoluten Anteil anzuzeigen, aber auf eine Anzeige des relativen Wertes mittels ToolTips nicht zu verzichten.

Weitere Probleme ergaben sich in der Programmierung der einzelnen Diagramme, da Flex oftmals unseren Wünschen Grenzen setzte. Diese Details werden in den nachfolgenden Abschnitten genauer dargestellt.

Tortendiagramm

Die Realisierung des Tortendiagramms war zunächst kein größeres Problem.
Flex berechnet und erstellt das Diagramm bei Vorgabe entsprechender Daten weitestgehend selbstständig und dynamisch. Um das Diagramm ansprechender zu gestalten, haben wir hier viel am Style gearbeitet. So wurden Farben, Linien und Styles definiert, angepasst und anschließend in ein CSS-File ausgelagert.
Der Code zeigt den Aufbau des Tortendiagramms in Flex.

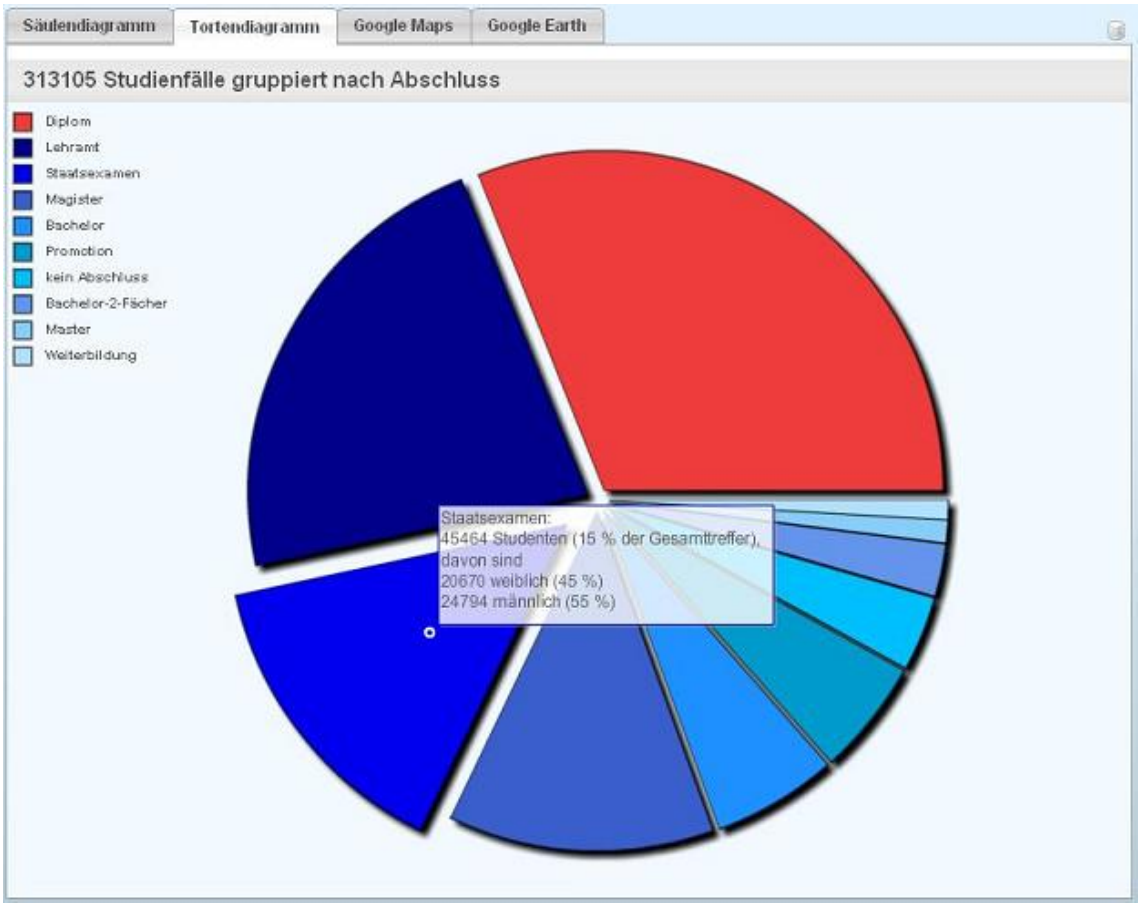
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        ...
    </mx:Script>
        ...
    <mx:Panel
        id="panel"
        width="100%" height="100%"
        layout="absolute">
        <mx:Legend
            id="pieChartLegend"
            dataProvider="{pieChart}"
            height="100%"/>

        <mx:PieChart
            id="pieChart"
             height="100%" width="95%"
             showDataTips="true"
               dataTipFunction="pieChart_dataTipFunction">
            <mx:series>
                <mx:PieSeries
                    id="pieSeries"
                    field="valueTotal"
                    nameField="Name"
                       showDataEffect="{effInitial}"
                       labelFunction="pieChart_labelFunction"
                    radialStroke="{radial}"
                    stroke="{pieborder}">
                </mx:PieSeries>
            </mx:series>
        </mx:PieChart>
    </mx:Panel>
    ...
</mx:Canvas>
```

Als besonderer Effekt entstand beispielsweise auch ein Initialeffekt, der bei Übergabe neuer Daten einmalig abgefahren wird und das Tortendiagramm entstehen lässt. Dieser Initialeffekt wird nur bei Übergabe völlig neuer Daten genutzt.

Bei Erneuerung der Daten sollte sich das Tortendiagramm durch Interpolation animierend verändern (siehe Punkt Animation).
Hierbei entstand das Problem, wie es umzusetzen sei, dass der Initialeffekt lediglich ein Mal abgefahren wird und sich danach das Diagramm nur noch per Interpolation verändert. Bei allen Versuchen, den Effekt abzuschalten und danach einen anderen zu nutzen, führten dazu, dass der ressourcentechnisch relativ anspruchsvolle Initialeffekt weiterhin im Hintergrund lief und bei jeder Mausbewegung über dem Tortendiagramm erneut gestartet wurde.
Dieses Problem wurde letztlich erfolgreich durch die Nutzung von so genannten States gelöst, wobei sich der Initialeffekt nach seiner ersten Ausführung im Initial-State quasi selbst eliminiert.
Bei der abschließenden Durchsicht entfernten wir uns dann von der Nutzung von States und fanden einen Weg, wie sich der Initialeffekt quasi selbst eliminierte.

Weiterhin wurde, da eine Unterteilung von männlich und weiblich im Tortendiagramm wie oben bereits erwähnt keinen Sinn macht, eine ToolTip-Funktion geschrieben, die, sobald man mit der Maus über ein Tortenstück fährt, die wichtigsten Daten anzeigt, darunter die Gesamtstudentenzahl sowie die Anzahl der männlichen und weiblichen Studenten. Der ToolTip zeigt sowohl absolute als auch relative Zahlen an.
Im statischen Modus wird nicht nur ein ToolTip mit näheren Informationen zum gewählten Datensatz angezeigt, sobald man die Maus über ein Tortenstück bewegt, sondern zusätzlich springt zur besseren Indikation des ausgewählten Elementes das jeweilige Kuchenstück um einen vorher festgelegten Faktor aus dem Tortendiagramm heraus.
Diese ToolTip-Funktion sollte zunächst nur im statischen Anzeigemodus aktiv, während der Animation jedoch ausgeschaltet sein.
Wir entschieden uns jedoch letzten Endes dafür, den ToolTip auch während der Animationen anzeigen zu lassen, damit der Nutzer sich stets Informationen über jedes Kuchenstück auch während der laufenden animierten Veränderung der Tortenstücke einholen könne.



ToolTip und Indikation des Tortenstücks

Ein Problem war anfangs die Indikation der ausgewählten Tortenstücke als auch der stilistisch ansprechendere Effekt des "Abrückens" aller Tortenstücke um einen gewissen Faktor aus der Mitte.

Es wurde nämlich klar, dass ein statischer Faktor bei unterschiedlichen Datensätzen unterschiedlich intensiv wirkte. Sah das Abrücken der Tortenstücke aus der Mitte um einige wenige Pixel bei 25 Datensätzen noch ästhetisch aus, da die Abstände zwischen den Stücken nur leicht angedeutet wurden, so wirkte dieselbe Zahl an Pixeln als Abstand bei 2 oder 3 Datensätzen schon übertrieben groß.
Wir mussten also einen Algorithmus entwerfen, der das "Herausspringen" und das "Abrücken" von Tortenstücken in Abhängigkeit der Anzahl der vorliegenden Daten vornahm.

Zu Beginn der Entwicklungsphase wurde das Tortendiagramm mit Labels bestückt, die nicht innerhalb der Torte, sondern links und rechts außen angezeigt wurden und mittels Callout-Linien auf das jeweils zugehörige Tortenstück verwiesen. Diese Labels zeigten die grundlegenden Daten an, die der x- und y-Achse im Säulendiagramm entsprechen, also Anzahl an Studenten und die jeweiligen Datengruppierung.

Es stellten sich hierbei mehrere Schwierigkeiten heraus, wie z.B. die Tatsache, dass sich die Labels nicht wie gewünscht formatieren ließen. Flex ließ nicht zu, dass die Labels eingerahmt werden konnten, zudem war eine Links-Ausrichtung nicht möglich - die Schrift in den Labels war stets zur Mitte ausgerichtet.

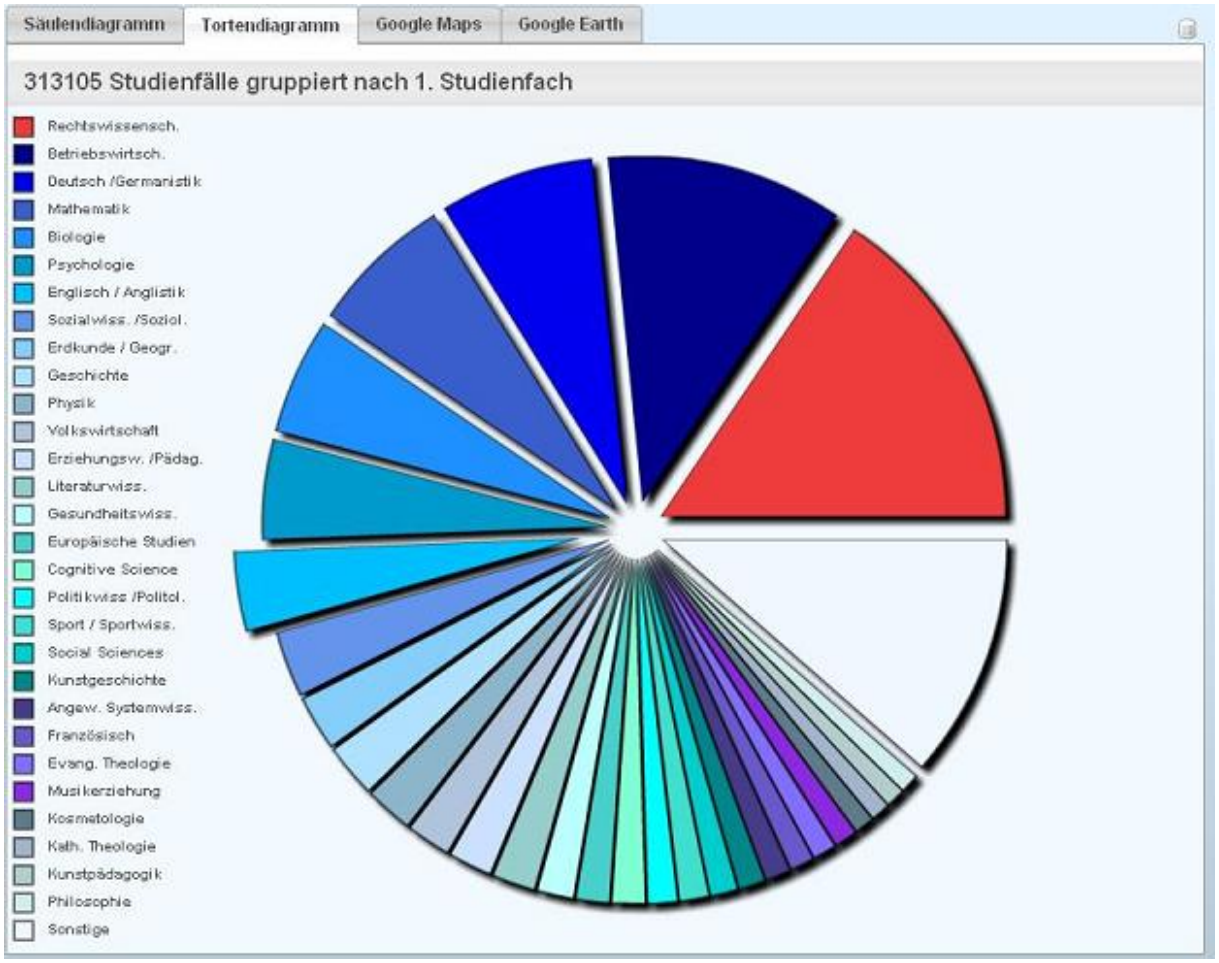
Die Ausrichtung der Schrift ließ sich durch das Auffüllen von Blanks in Abhängigkeit der Zeichenlänge aushilfsweise regeln, jedoch fand sich keine Lösung für das Einrahmen der Labels. Dies hätte bei wenig Datensätzen auch vernachlässigt werden können, bei mehr als 30 Datensätzen hätte dieser Umstand jedoch zu erheblicher Unübersichtlichkeit geführt.

Nichtsdestotrotz ergab sich in Bezug auf die Labels das größte Problem bei der Animation. Wir stellten fest, dass Flex die Labels während einer Animation verschwinden ließ, um sie erst wieder sichtbar zu machen, wenn das Tortendiagramm fertig aufgebaut war. Während unserer geplanten Animation hätte das dazu geführt, dass die Labels beim stetigen Durchlauf der Daten immer nur für den Bruchteil einer Sekunde zu sehen gewesen wären.

Man könnte demnach während der Animation zwar sehen, wie sich die einzelnen Datensätze durch Interpolation vergrößerten oder verkleinerten, wüsste aber nichts über die Daten, da weder Gruppierung noch Anzahl Studenten angezeigt würden.

Dies ließ uns letztlich den Entschluss fassen, auf die Labels vollständig zu verzichten und stattdessen eine Legende einzufügen und zusätzlich auch während der Animation ToolTips anzeigen zu lassen.
Wir formatierten die Legende so, dass diese in eine Spalte passte - dies war wichtig, damit sich die Legende nicht ggf. mit dem Kuchen überschnitt.

Weiterhin gelang es uns nach langer Arbeit und Tüftelei, eine Funktion und zugehörige Event-Handler zu schreiben, die, sobald man die Maus über ein Legenden-Item bewegte, das zugehörige Tortenstück herausspringen ließ. So war es nun möglich geworden, ein Tortenstück sowohl direkt zu indizieren als auch dieses über die Legende anzusprechen.



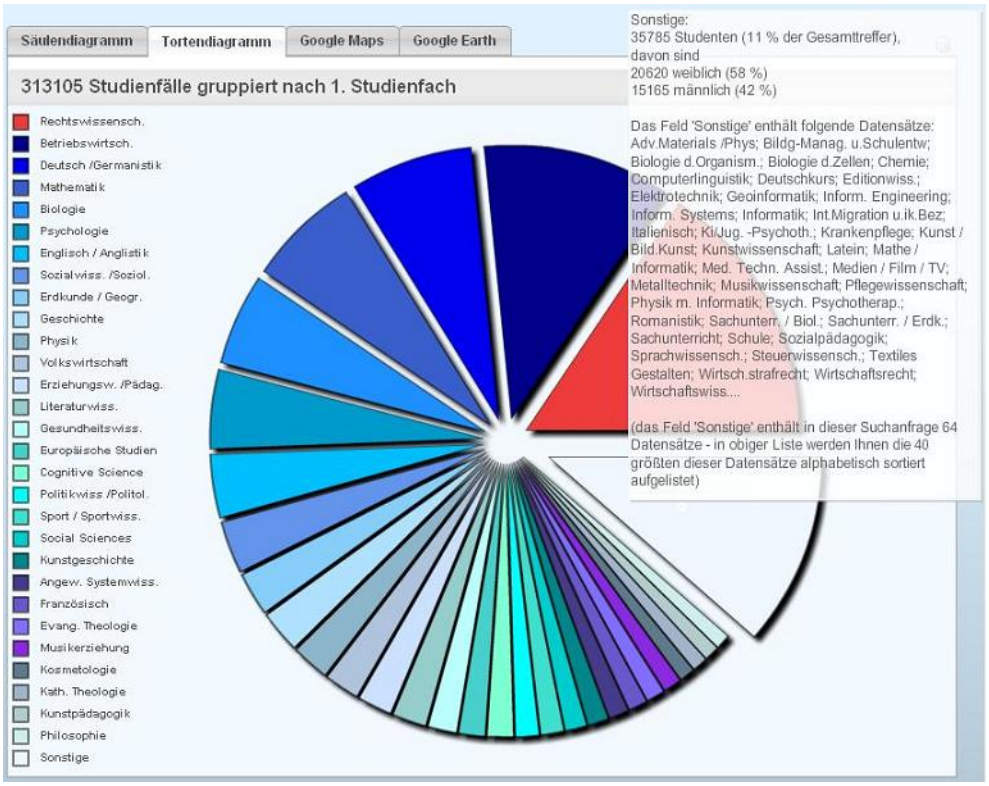
Legende und Legenden-EventHandler

Eine der größten Schwierigkeiten in Bezug auf die Anzeige wurde letztendlich die Anzahl der Daten. Nachdem die Daten aus der Datenbank verfügbar waren, wurde das Tortendiagramm gezwungen, bis zu 600 Datensätze (=Tortenstücke!) anzuzeigen. Dies machte keinen Sinn, so dass wir uns entschlossen, die Anzahl der Tortenstücke auf 30 zu begrenzen.

Damit gingen natürlich konsequenterweise 2 Folgen einher:
Zum einen mussten die anzuzeigenden Datensätze ausgewählt werden, zum anderen war klar, dass die Darstellung aller restlichen Daten in einem Tortenstück "Sonstige" aufgeführt werden mussten.

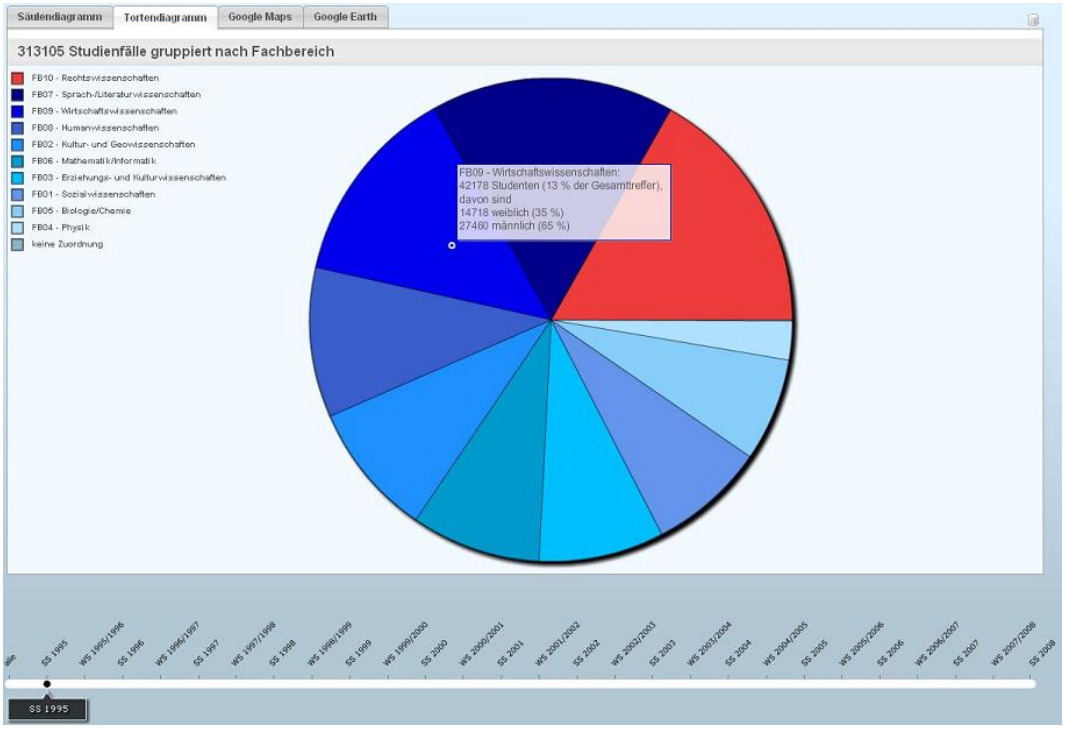
Bei der Wahl der anzuzeigenden Datensätze entschieden wir uns für die 29 größten - demnach relevantesten - Daten. In Feld 30 sollten dann die restlichen Daten zusammengefasst werden, was zu Additionsschleifen und weiteren ausgeklügelten Algorithmen führen sollte.

Da wir also die Daten stets der Größe nach benötigten, legten wir fest, im Tortendiagramm ausschließlich sortierte Datensätze zu nutzen. So wurden die ersten 29 angezeigt und der Rest im Feld Sonstige zusammengefasst, wobei wir für das Feld 30 einen eigenen ToolTip entwarfen, der angab, wie viele und welche Daten in diesem Feld zusammengefasst wurden.



ToolTip beim Feld Sonstige

Eine negative Konsequenz brachte die Sortierung jedoch mit sich: In der Animation wurden alle Datensätze stets neu sortiert - weniger erfreulich daran war, dass bei jedem Datentausch ein Datensatz eine neue Farbe und in der Legende eine neue Position bekam. Die Daten tauschten "im Laufe der Zeit" - also während der Animation - ihre Reihenfolge (je nach Größe der Datensätze). Obwohl wir aufgrund der Zeit dieses Problem nicht mehr lösen konnten, so kompensierten wir es doch mit dem ToolTip, den wir für die Animation integrierten, so dass ein Nutzer auch während der Animation stets prüfen konnte, wo welcher Datensatz angezeigt werden würde.



Tortendiagramm im Animationsmodus

Säulendiagramm

Die Flex-Charting-Komponenten bieten vier verschieden Typen der Säulendiagramme an. Für unsere Anforderungen haben wir uns erstmal auf zwei beschränkt: Stacked oder Clustered.

Unser erster Ansatz war es, dem Benutzer zwei verschiedene Ansichten zu bieten, zwischen die er wählen konnte. Die erste Ansicht zeigt die gesamte Studentenanzahl und die zweite Ansicht unterteilt diese nach dem Geschlecht. Im letzten Fall würden zwei Säulen angezeigt werden, die nebeneinander stehen. Diese Ansicht lässt sich mich Clustered-Säulendiagramm realisieren.

Der zweite Ansatz besteht darin, alle Informationen in einer Säule darzustellen, indem wir die Säulen, die jeweils die Studentinnen und Studenten repräsentieren, aufeinander stapeln und somit auch die Gesamtzahl angezeigt bekommen. Die Vor- und Nachteile der Säulendiagramme zeigt die folgende Tabelle:

Säulendiagramm	Vorteile	Nachteile
Clustered	-Vergleich der weiblichen und männlichen Studenten deutlicher zu sehen	-Der Wechsel der Ansichten wirkt verwirrend, weil die y-Achse sich der Säulen dynamisch anpasst. Das bedeutet, dass sich die Größe der Säulen nicht wesentlich ändert, sondern nur die Skalierung der y-Achse. Somit könnte es zu einem verfälschten Eindruck kommen -Implementation mit zwei ColumnChart-Komponenten
Stacked	-Alle Angaben in einer Säule -Implementation mit einer ColumnChart-Komponente	-Vergleich der weiblichen und männlichen Studenten nicht optimal

Da wir die Diagramme noch über die Zeit animieren wollten, haben wir uns für das "Stacked"-Säulendiagramm entschieden. Denn für die Animation müssen die Daten für alle Zeitpunkte im Voraus berechnet werden, um die Ladezeit zu verkürzen. Somit hätte man für das Clustered-Säulendiagramm zwei verschiedene Datensätze gebraucht, das mehr Ladezeit und Speicherplatz beanspruchen würde.
Der Code zeigt den Aufbau eines Balkendiagramms und die Initialisierung der Säulen und der Achsen.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <mx:Canvas xmlns:mx="http://www.adobe.com/2006/mxml">
3      <mx:Script>
4          ...
5      <mx:Script>
6          ...
7      <mx:HBox width="100%" height="100%" id="hbox">
8          <mx:ColumnChart
9              gutterBottom="100"
10             id="myChart"
11             showDataTips="true"
12             height="100%"
13             dataProvider="{dataSet}"
14             width="{(model.queryData.length*breite)+50}"
15             type="stacked">
16             <mx:horizontalAxis>
17                 <mx:CategoryAxis
18                     id="xAchse"
19                     displayName="{model.filterVO.groupBy}"
20                     categoryField="Name"
21                     />
22             </mx:horizontalAxis>
23             <mx:verticalAxis>
24                 <mx:LinearAxis
25                     id="yAchse"
26                     displayName="Studienfälle"
27                      minorInterval="100"/>
28             </mx:verticalAxis>
29             <mx:horizontalAxisRenderers>
30                 <mx:AxisRenderer
31                     labelRotation="25"
32                     axis="{xAchse}"
33                     />
34             </mx:horizontalAxisRenderers>
35             <mx:series>
36                 <mx:ColumnSeries
```

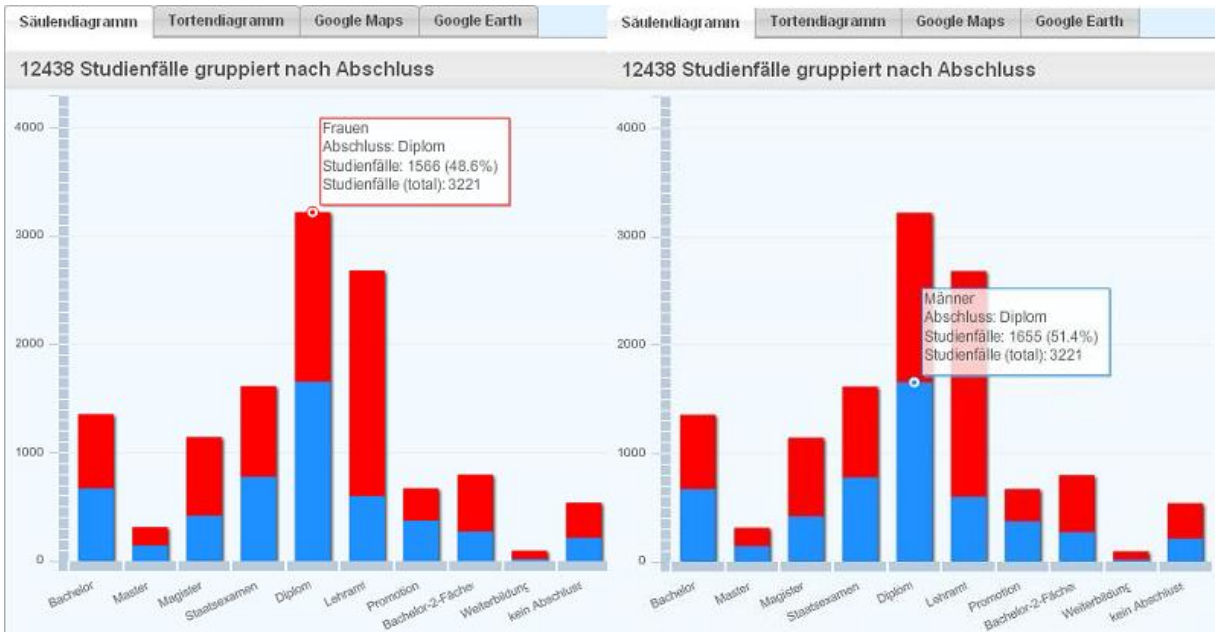
```
37                      id="mSeries"
38                      displayName="Männer"
39                      yField="valueMale"
40                      xField="Name"
41                      showDataEffect="{effIn}" >
42                          <mx:fill>
43                              <mx:SolidColor color="#1E90FF"/>
44                          </mx:fill>
45                  </mx:ColumnSeries>
46                  <mx:ColumnSeries
47                      id="fSeries"
48                      displayName="Frauen"
49                      yField="valueFemale"
50                      xField="Name"
51                      showDataEffect="{effIn}">
52                          <mx:fill>
53                              <mx:SolidColor color="#FF0000"/>
54                          </mx:fill>
55                  </mx:ColumnSeries>
56              </mx:series>
57          </mx:ColumnChart>
58      </mx:HBox>
59      ...
60  </mx:Canvas>
```

Konfiguration

Nachdem wir unsere Wahl getroffen haben, musste nun das Layout bestimmt werden. Die Breite der x-Achse wird durch die Anzahl der Gruppierungselemente dynamisch berechnet. Zusätzlich wird das Diagramm mit einer vertikalen Scrollbar versehen, falls es zu viele Gruppierungselemente gibt. Weiterhin haben wir uns entschieden, dass die Beschriftung der x-Achse etwas schräger angesetzt wird, um die Säulen besser zuzuordnen, weil die Beschreibungen der Gruppierungselemente unregelmäßig lang bzw. kurz sind. Der Text über dem Diagramm gibt dem Benutzer die Auskunft, wie viele Studenten insgesamt nach dem Filtern gefunden wurden und wonach gruppiert wird.

ToolTip

Um den Nachteil des Stacked-Säulendiagrammes aufzuheben, sollten im Tooltip neben den absoluten Zahlen auch die prozentualen Zahlen stehen. Das generiert Flex automatisch. Weiterhin werden die Gruppierung und die Zuordnung der Säule angegeben. Die Randfarbe des Tooltips passt sich hier automatisch dem der Säule an, so dass es eindeutig bleibt.
Das Bild zeigt die Realisierung unserer Ideen.



Säulendiagramm im Erfassungssemester WS 05/06

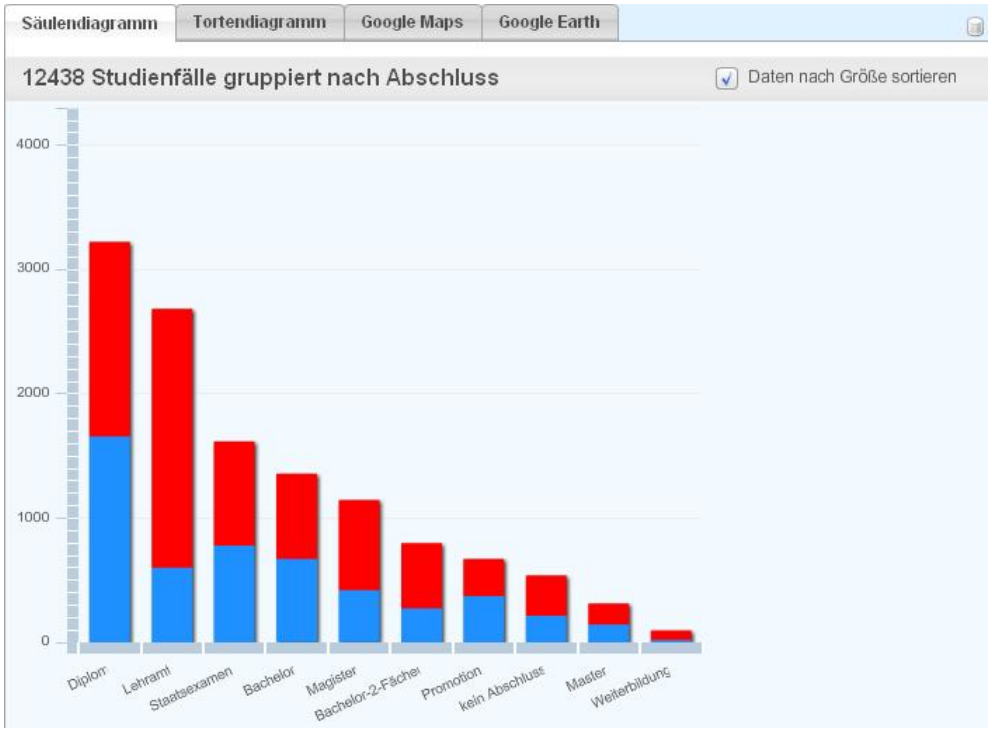
Sortierung der Daten

Weitere Überlegungen, die sich im Laufe des Projektes ergeben haben, entstanden dem Wunsch, innerhalb der Statistikanzeige die Daten nach dem Attribut Anzahl Studenten sortieren zu können.

Die Daten sollten mittels einer CheckBox, die am oberen Rand des Statistikvisualisierungsfensters platziert werden sollte, auf Mausklick sortiert angezeigt werden können. Es ermöglicht dem Benutzer eine bessere Ansicht, falls dieser sich z.B. für die Studiengänge interessiert, in denen sich die meisten Studenten befinden. Dieses Feature sollte nur dem Säulendiagramm gegeben werden, da sich im Tortendiagramm nur die geordneten Datensätze befinden(s.Tortendiagramm).

Die Implementation einer Sortierungsfunktion im Programmcode stellte soweit kein Problem dar. Es wurde darauf verzichtet, eine Sortierung von männlichen oder weiblichen Zahlen vornehmen zu können, die Sortierung beschränkt sich demnach auf die Gesamtzahl an Studenten.

Der Übergang zwischen den beiden Status wurde wie bei der Animation (siehe dort) mittels Interpolation geregelt und kann in Echtzeit und unmittelbar ohne Verzögerung erfolgen, weil alle Daten nicht noch mal übergeben werden mussten, sondern in der View gleich zwei mal angelegt wurden, wobei die eine sortiert wird.



Sortiertes Säulendiagramm im WS 05/06

Animation

Im Laufe der zweiten Woche des Computergrafikpraktikums entstand die Idee, die einzelnen Datensätze in ihrer zeitlichen Abfolge animiert einsehen zu können, und zwar in der Art, dass eine Animation mittels eines Play-Buttons entweder über alle Winter- oder aber über alle Sommersemester gestartet werden kann.
Die Idee dabei war, alle Semester nacheinander einsehen zu können, wobei sich die Daten in den Statistiken animiert ändern sollen, d.h. die Daten des jeweiligen Semesters verlaufen fließend in die neuen Daten des übernächsten Semesters.

Beim Säulendiagramm bedeutet dies, dass die Säulen entweder anwachsen oder abfallen, beim Tortendiagramm werden die Tortenstücke entsprechend größer oder kleiner.

Um die Animation starten zu können, mussten wir beim Säulendiagramm die jeweiligen Achsen festsetzen. Hätte wir die y-Achse nicht durch das Maximum aller Säulen gesetzt, so hätte diese sich während der Animation dynamisch verändert und die Tendenzen verfälscht. Weiterhin mussten alle Gruppierungselemente der View übergeben werden, da entweder Flex automatisch die Elemente nicht zeigt, dessen Quantität gleich null ist oder es Gruppierungselemente erst zu einem späteren Zeitpunkt gibt. Als Beispiel hätten wir den Abschluss Bachelor bzw. Master, der erst seit dem Wintersemester 1998/1999 existiert.
Der Code zeigt die Methode, die bei der Animation der Säulen aufgerufen wird.

```
1 private function animate(target:String):void
2             {
3                  this.yAchse.maximum=model.yMax;
4                  this.xAchse.dataProvider=model.abscissas;
5             }
```

Somit können wir das Anwachsen und das Abfallen der Torten und Säulen mit dem Effekt der Interpolation realisieren und die Animation kann beginnen.

Fazit

Die Flex Charting Komponenten sind dynamisch und das Programmieren der Diagramme ist recht einfach und lässt viele Spielereien zu. Ein noch offenes Problem ergibt sich, sobald man auf die Dynamik verzichten will. In unserem Fall mussten wir für die Animation die x-Achse festsetzen. Das führte aber zu einem nicht gewünschten Nebeneffekt. Flex interpolierte die Säulen nicht immer, sondern verschob sie. Dies ist wahrscheinlich darin begründet, dass Flex manchmal "faul" ist und erkennt, dass das Schieben der Säulen mit weniger Aufwand verbunden ist als das Interpolieren der Säulen. Zusammengefasst sind die Komponenten in ihrer Handhabung einfach, solange man ihre Einstellungen übernehmen will. Die Dynamik kann hier aber auch zu Problemen führen, falls man teilweise auf sie verzichten möchte.

4. Gruppe Google Maps

Diese Dokumentation gibt einen Überblick über die Programmierung der ***Google Maps API for Flash***-Komponenten, welche in die Anwendung zur Visualisierung von Immatrikulationsdaten des Computergrafikpraktikums 2008 integriert wurden. Sie geht auf Schwierigkeiten während der Programmierung ein und erklärt die Lösungswege die eingeschlagen wurden.

Inhaltsverzeichnis

Aufbau Oberfläche

Autor: Andreas Wichmann & Florian Hillen

Oberfläche

Unsere Idee für die Benutzeroberfläche der Google Maps Applikation beruht darauf sie möglichst einfach und intuitiv für den Benutzer zu gestalten, ihm aber dennoch die Möglichkeit zu geben auf bekannte Interaktionsmöglichkeiten zurückzugreifen. Aus diesem Grund entschieden wir uns beispielsweise dazu die Auswahl der Map-Types herauszunehmen, da die physikalische Karte zur Anzeige der Daten am besten geeignet war. Zur Interaktion mit der Karte fügten wir eine PositionControl und ZoomControl hinzu, so dass der Benutzer sich auf der Karte bewegen und die Zoomstufe frei bestimmen kann. Jedoch beschränkten wir die Zoomstufen, da ein weiteres Herauszoomen die Darstellung der Treffer nur undeutlich gemacht und ein weiteres Hineinzoomen auf Grund der Übersichtlichkeit wenig Sinn gemacht hätte.

Desweiteren entschieden wir uns dem Benutzer die Möglichkeit zu geben auch gezielt nach Städten suchen zu lassen, indem wir im linken unteren Bereich auf Flex basierende Elemente zurückgriffen. Über die von Google Maps bereitgestellte Möglichkeit des Geocodings findet so eine automatische Positionierung der Karte auf die gewünschte Stadt statt.

Idee der Strukturierung

Autor: Andreas Wichmann & Florian Hillen

Bei Bearbeitung der uns gestellten Aufgabe zur Visualisierung von Immatrikulationsdaten in Google Maps unterteilten wir diese Aufgabe in drei Teilbereiche: Bundesländer, Marker und Infofenster. Dies erschien uns als sinnvoll, da die von Google Maps übergebene Karte für unsere Zwecke graphisch angepasst werden musste. Auf Grund der Raumbezogenheit der Daten, erschien uns der Einsatz von Markern als beste Möglichkeit zur Visualisierung. Um einen größeren Bereich der Informationen nutzen zu können und die Darstellung nicht nur auf die Trefferanzahl in den Markern zu reduzieren, sahen wir die Infofenster als beste Anzeigeform an.

Bundesländer

Autor: Andreas Wichmann & Florian Hillen

Beim Betrachten der Google-Maps-Karte auf minimaler Zoomstufe war zu erkennen, dass keines der Bundesländer mit Grenzen versehen war. Erst bei höheren Zoomstufen wurden dünne, gestrichelte Landesgrenzen sichtbar. Wir entschieden uns die Grenzen deutlicher hervorzuheben, um so einen besseren visuellen Eindruck der angezeigten Treffer zu erhalten.

Obige Bilder zeigen die Veränderung der Deutschlandkarte vor und nach dem Einfügen der Bundesländergrenzen.

Zunächst informierten wir uns die von Google bereitgestellte Sprache KML[4], welche die Möglichkeit bietet, graphische Elemente über die Karte zu legen. Nach einigen Recherchen mussten wir feststellen, dass die KML Überlagerung in der noch jungen Google Maps API for Flash nicht möglich ist.

Unser nächster Ansatz nutzte die von Google Maps bereitgestellte Polyline, welche eine Aneinanderreihung von einzelnen Linien darstellt. Bei näheren Überlegungen erschien es uns sinnvoller Polygone zu nutzen, da wir so nicht nur die Grenzen der Bundesländer anzeigen konnten, sondern zudem auch noch die Möglichkeit hatten mit dem Polygon bzw. Bundesland zu interagieren. Auf Grund der hohen Zoomstufen von Google Maps

4 http://de.wikipedia.org/wiki/Keyhole_Markup_Language

benötigten wir relativ genaue Koordinaten der Grenzen. Ein Tool der Geoinformatik (OpenJUMP[5] Version 1.2.0) half bei der Beschaffung dieser Daten. Anfängliche Befürchtungen, dass die ca. 30.000 Koordinatenpaare die Applikation stark verlangsamen würden, veranlassten uns dazu die Genauigkeit der Grenzen herunterzusetzen, indem die Bundesländer von Hand nachdigitalisiert wurden. Dies verringerte die Anzahl der Koordinatenpaare um den Faktor 10. Nach den ersten erfolgreichen Testläufen mit dem verkleinerten Koordinaten-Datensatz entschieden wir uns, das Land Niedersachen mit voller Koordinatenanzahl in die Applikation einzubinden. Die Ladegeschwindigkeit blieb weiterhin konstant, so dass wir auch die fehlenden Bundesländer in vollem Umfang hinzufügten. Ein selbstgeschriebenes Skript überführte die Rohdaten von OpenJUMP in ein von Google Maps interpretierbares Format. Nach vollständiger Einbindung aller Polygone stellten sich unsere anfänglichen Befürchtungen als unbegründet heraus, da die Performance der Applikation nicht unter der hohen Koordinatenanzahl litt.

Hover-Effekt am Beispiel Sachsen-Anhalt

Durch den Einsatz von Polygonen eröffneten sich mittels dem mitgelieferten MapMouseEvent neue Interaktionsmöglichkeiten mit der Karte. Bei Anklicken eines Bundeslandes auf minimaler Zoomstufe wird die Karte passend auf das Bundesland ausgerichtet, wobei auch die Zoomstufe entsprechend angepasst wird. Ein weiteres Anklicken ist hier nicht mehr möglich. Wir entschlossen uns gegen die Möglichkeit, den von Google bereitgestellten Geocoder für die Ausrichtung der Karte zu benutzen, da dieser mit hohem Aufwand verbunden gewesen wäre. Mittels OpenJUMP definierten wir unsere Ausrichtungspunkte selbstständig und übertrugen die Koordinaten direkt in den Quelltext.

Ein weiterer Effekt den die Polygone mit sich bringen ist der Mouse-Over-Effekt, welcher sich ebenfalls über zwei Bestandteile des MapMouseEvents realisieren lässt. Zur besseren Orientierung werden daher die Bundesländer beim Überfahren der Karte mit der Maus durch einen leichten Farbeffekt wie auf dem Bild rechts zu sehen hervorgehoben.

Um bereits auf der minimalen Zoomstufe einen optischen Eindruck über die Verteilung der Studentenzahlen zu erhalten, entschieden wir uns dazu die Hintergrundfarbe in Abhängigkeit der Trefferanzahl einzufärben. Wir stellten fest, dass viele verschiedene Farben hierfür zur Irritation des Benutzers führen. Aus diesem Grund realisierten wir die Einfärbung über den Alpha-Wert, der die Durchsichtigkeit der Blautöne definiert. Da dieser Effekte nur auf minimaler Zoomstufe angezeigt werden soll, mussten wir eine Überprüfung anhand der Zoomstufe vornehmen. Leider bot die Google Maps API for Flash in der Version 1.5 keine Möglichkeit auf Ereignisse der ZoomControl zu reagieren. Aus diesem Grund versuchten wir das Problem zu umgehen, indem wir unsichtbare Buttons über die ZoomControl legten, welche auf Anklicken separat reagieren konnten.

Mit dem Update der Google Maps API for Flash auf die Version 1.6 wurde ein neues Event implementiert, welches eine Reaktion auf die von der ZoomControl geworfenen Events ermöglichte. Das MapZoomEvent liefert uns also unter Anderem die Möglichkeit auf Änderung des Zooms zu reagieren und so den Effekt nur auf die minimale Zoomstufe zu beschränken.

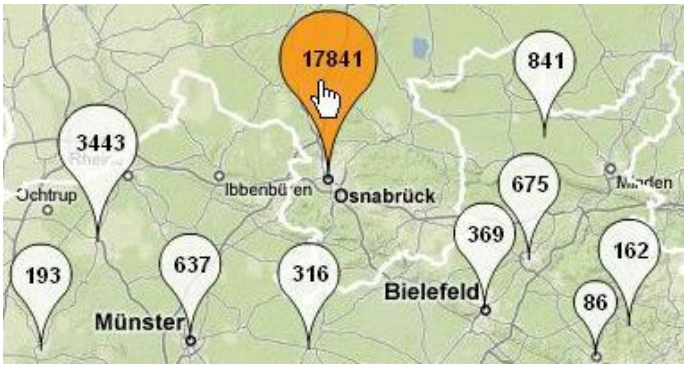
5 http://openjump.org

Ein letztes Problem bezieht sich auf die Überlappungen der Bundesländer in den Bereichen der Stadtstaaten. Hier wären Löcher in den Polygonen von Nöten gewesen, welche zur Zeit jedoch nicht in Google Maps implementiert sind. So kann es beim direkten Anklicken einzelner Stadtstaaten zur Ausrichtung der Karte auf das umgebende Bundesland kommen. Dieses Problem lässt sich zur Zeit nicht beheben.

Bundesländerpolygone: Sourcecode (PolygonOptions)

Marker

Autor: Andreas Wichmann & Florian Hillen



Markerdarstellung

Nebenstehendes Bild zeigt die Darstellung der Marker nach der graphischen Anpassung an die Applikation. Die Marker unterscheiden sich dabei hinsichtlich der Größe und der Farbe. Die Größe ändert sich in Relation zur Trefferanzahl, die Farbe dient zur Verdeutlichung der Selektion, indem sie sich beim Mouse-Over von weiß zu orange ändert. Beim Anklicken eines Markers wird die Mouse-Over-Farbe fixiert und das Infofenster geöffnet. Erst beim Schließen des Infofensters nimmt der Marker wieder seine ursprüngliche Gestalt an.

Da bei voller Farbfülle unterliegende Informationen wie zum Beispiel Städtenamen verdeckt werden können, entschieden wir uns die Marker durch die Anpassung des Alpha-Wertes eine leichte Durchsichtigkeit zu geben. Dies erkennt man am nebenstehenden Bild besonders gut im linken Teil am Beispiel der Stadt Rheine.

Marker: Sourcecode

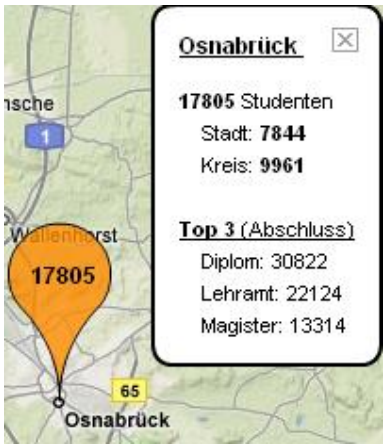
Bevor wir uns jedoch für die oben beschriebene Markerdarstellung entschieden haben, war unsere erste Idee die Marker als Kreis darzustellen. Die Durchführung dieser Idee stieß jedoch bereits nach kurzer Zeit auf unlösbare Probleme. Zwar war es möglich mit Hilfe der Polygone einen Kreis zu definieren und dessen Farbe beliebig anzupassen, jedoch fehlt einem Overlay generell die Fähigkeit ein Label zu besitzen, so dass wir die Trefferanzahl nicht direkt darstellen konnten. Desweiteren war es nicht möglich die Größe des Polygons an die jeweiligen Zoomstufe anzupassen. Dies ließ sich auch nicht damit umgehen, dass wir bereits vordefinierte PNG's zur Darstellung nutzten.

Aus diesem Grund fiel unsere Entscheidung auf den Standardmarker, der unter anderem den positiven Effekt mit sich brachte, dass die exakte Position auf Grund der Georeferenzierung besser zu erkennen war, da die Spitze direkt auf dem entsprechenden Punkt in der Karte aufsetzt. Da die bereits vorhandenen Marker beim erneuten Setzen nicht automatisch überschrieben werden können, müssen alle Overlays gelöscht und mit den neuen Daten nochmals erstellt werden. Eine komfortable Möglichkeit des Löschens aller Overlays wurde erst mit dem Update der API auf die Version 1.6 ermöglicht.

Weitere Überlegungen beschäftigten sich damit nicht nur die Größe, sondern auch die Farbe der Marker in Abhängigkeit der Trefferanzahl zu verändern. Dies führte zu dem Problem, dass leichte Farbabstufungen nicht deutlich sichtbar wurden. Auch eine Aufteilung in mehrere möglichst gut unterscheidbare Farben verwirrte den Benutzer mehr als dass es zur Übersichtlichkeit der Karte beitrug.

Infofenster

Autor: Andreas Wichmann & Florian Hillen



Infofenster mit unterschiedlichen Top 3 Gruppierungen

Nebenstehende Bilder zeigen die Infofensterdarstellung nach der graphischen Anpassung an die Applikation. Das Fenster öffnet sich beim Klick auf einen Marker und bleibt bis zur Auswahl eines anderen Markers oder bis zum Schließen aktiv. Der Inhalt liefert eine informelle Übersicht über die ausgewählte Stadt / Region. Um dem Benutzer die Möglichkeit zu geben mehr Informationen über die gewählte Stadt / Region zu erhalten, entschieden wir uns dazu ein Direktlink zu Wikipedia zu realisieren. Dies stellte uns jedoch vor ein Problem, da die Städtenamen in der Datenbank oft nicht wikipediakonform angelegt waren. So mussten Abkürzungen wie "a. d." oder Städte wie "Wesermünde", die seit 1947 einen anderen Namen haben, von Hand geändert werden. Über einen Klick auf den Städtenamen im Infofenster gelangt man nun zur passenden Wikipedia-Seite und kann dort weitere Informationen einsehen.

Der eigentliche Inhalt des Infofensters bezieht sich auf die statistischen Werte die durch die Filter selektiert wurden. Im oberen Teil werden die absoluten Studenten bzw. Studienfälle angezeigt, welche nochmals in Stadt und Kreis unterteilt werden, falls ein Kreis zur ausgewählten Stadt existiert. Sollte nur ein Kreis, ohne zugehörige Stadt, in der Trefferliste vorhanden sein, so wird diese Funktion übersprungen. Im unteren Teil befindet sich eine "Top 3"-Statistik, die in Abhängigkeit der durch die Filter einstellbaren Gruppierung gefüllt ist. Dies ist an den beiden obigen Bildern erkennbar, welche die Gruppierung nach dem 1. Studienfach und den Abschlüssen zeigen.

Infofenster: Sourcecode

Das Infofenster lässt sie über die Google Maps API for Flash mit einem Befehl implementieren. Jedoch biete diese Standardversion wenige Formatierungsmöglichkeiten. Im Folgenden wurden Versuche angestellt mit dem Attribut contentHTML den Inhalt über HTML-Tags zu formatieren. Hierzu konnten CSS[6] Attribute definiert werden, welche jedoch nur teilweise von der API unterstützt wurden. Nach direkten Anfragen bei Google wurden wir auf das in Version 1.6 neu implementierte Attribut customContent verwiesen, welcher das Infofenster als Bitmap vorhält und so jegliche Anpassungen ermöglicht.

Ein weiteres Problem stellte die Darstellung der Top 3 - Statistiken dar, welche aus Performancegründen erst beim Öffnen des Infofensters aus der Datenbank ausgelesen werden sollten. So mussten eigene Events für die Darstellung der Infofensterinhalte implementiert werden. Es war daher von Nöten, dass der Marker einige, wichtige Attribute übergeben bekommt, um die Infofenster individuell an die Marker zu binden. Um die teilweise erheblichen Ladezeiten zu überbrücken, wird direkt nach Anklicken eines Markers ein weiteres, temporäres Infofenster geöffnet, welches den Benutzer um eine kurze Wartezeit bittet.

Während der Entwicklung mit der älteren API Version 1.5 fiel uns auf, dass Infofenster, die am Rand der Kartendarstellung geöffnet wurden, teilweise gänzlich außerhalb des Anzeigebereichs lagen. Zunächst versuchten wir dieses Problem zu umgehen, indem wir bei Klick auf einen Marker, den gesamten Bildausschnitt auf den Marker zentrieren. Diese Vorgehensweise brachte jedoch viel Unruhe in die Karte, da selbst bei kleinen

6 http://de.wikipedia.org/wiki/Cascading_Style_Sheets

Abweichungen die gesamte Karte verschoben wurde. Zudem konnte es sein, dass man mit einem Klick gänzlich den vorherigen Fokus (zum Beispiel auf ein selektiertes Bundesland) verliert. Aus diesen Gründen entschlossen wir uns dazu, eine eigene Funktion zu implementieren, welche die Karte entsprechend der Position des Markers verschieben sollte. Dazu definierten wir Korridore in denen keine Verschiebung stattfinden sollte. Außerhalb dieser Korridore wurde je nach Fall unterschiedlich, ähnlich der Regioncodes von Cohen & Sutherland, entschieden in welche Richtung die Karte verschoben werden sollte. Mit Erscheinen der neuen API Version 1.6 wurde diese Funktion automatisch von Google Maps übernommen, so dass unsere Funktion überflüssig wurde.

Fazit

Autor: Andreas Wichmann & Florian Hillen

Auf Grund der jungen API für die Flashversion von Google Maps hatten wir mit einigen Problemen zu kämpfen, welche einen engen Kontakt zu einigen Google Mitarbeitern notwendig machten. Diese reagierten schnell mit Tutorials und Hilfestellungen auf gestellte Anfragen und Hinweise.
Trotz alledem verhalf uns die sehr gute Dokumentation der API schnell zu ersten Erfolgen. Wir konnten so mit den vorgegeben "Pflichtaufgaben" zügig abschließen und hatten lange Zeit um unsere selbst gesteckten Ziele zu verwirklichen und zu verfeinern.
Alles in allem lässt sich sagen, dass die Google Maps API for Flash für die kurze Zeit schon umfangreiche Anpassungsmöglichkeiten bietet, jedoch mit der normalen Google Maps API in Umfang und Funktionalität für Großprojekte noch nicht mithalten kann. Wir sind jedoch der Meinung, dass sich dieser Zustand, auf Grund der motivierten Google Mitarbeiter, schnell ändern wird und so die Flashversion von Google Maps zu einer echten Alternative werden kann.

Autoren

Andreas Wichmann

Florian Hillen

Peer Wagner

5. Gruppe Google Earth

Marc Ermshaus, JI Yue Ming

Einleitung

Im Rahmen des Computergrafikpraktikums 2008 "Visualisierung von Immatrikulationsdaten" ist die Google Earth-Komponente eine der vier Darstellungsformen. In Abgrenzung zum Säulendiagramm, dem Tortendiagramm und der Google Maps-Karte ist es Aufgabe der Google Earth-Komponente, einen geographischen Überblick darüber zu geben, aus welchen Ländern der Welt Studierende den Weg an die Universität Osnabrück gefunden haben.

Zur Präsentation der Daten dient die direkt im Browser als Plugin ausführbare Version der bekannten Google Earth-Software, die einen frei dreh- und zoombaren Globus zur Verfügung stellt.

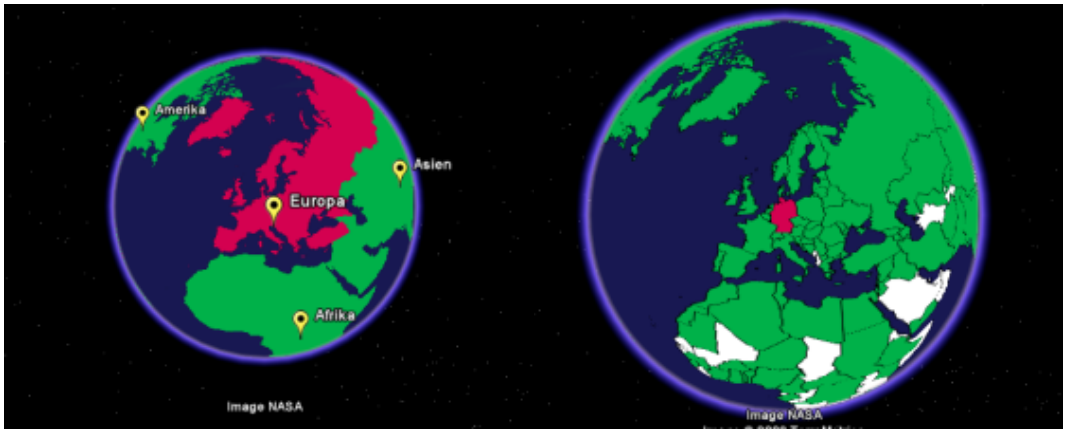
Das Plugin wurde am 2. Juni 2008 veröffentlicht und ist derzeit nur für Windows verfügbar. Es kann über die Google Earth API-Seite[7] heruntergeladen werden.

Benutzerdokumentation

Auf dem Google Earth-Globus wird das Resultat der aktuellen Filtereinstellungen immer so dargestellt, als wäre nach dem Herkunftsland gruppiert worden.

Steuerung und Zoomstufen

Die Verwendung der Google Earth-Komponente ist äußerst unkompliziert. Beim ersten Aufruf des zugehörigen Registerreiters wird das Plugin gestartet und es beginnt eine kurze Kamerafahrt auf Deutschland zu. Während dieses Flugs sind erst die Kontinente zu sehen, beim weiteren Absinken geht die Darstellung in Landesgrenzen über.



Zoomstufe: sehr weit Zoomstufe: weit

Die Anwendung wechselt abhängig von der Höhe des Betrachters den Detailgrad der angezeigten Informationen. Je geringer der Abstand zur Erde, desto mehr Details.

7 http://code.google.com/apis/earth/

Zoomstufe: nah

Zoomstufe: sehr nah

Drehen am Mausrad ist der einfachste Weg, die Zoomstufe zu verändern. Doch auch über die Tasten Plus/Minus und Bild auf/Bild ab kann die Höhe angepasst werden.

Zur direkten Anwahl eines bestimmten Detaillevels dient die Auswahlbox rechts unten im Benutzerinterface. Es gibt vier Stufen, die von "sehr weit" nach "sehr nah" folgende Details anzeigen:

sehr weit
: Statt Ländern werden "Kontinente" angezeigt. Die Zahl der Studierenden eines Kontinents ist die Summe aller Studierenden der Länder dieses Kontinents.

weit
: Hier werden nur die Landesgrenzen angezeigt. Die geeignete Stufe für den globalen Länderüberblick.

nah
: Zusätzlich zu den Landesgrenzen erscheinen (anklickbare) Landesflaggen und ein Kürzel des Landes.

sehr nah
: Das Kürzel wird zum vollen Landesnamen, die Anzahl der Studierenden wird dahinter in Klammern gesetzt.

Das Google Earth-Plugin reagiert übrigens erst dann auf einen neuen Befehl zur Änderung der Kameraposition, wenn sich die Kamera nicht mehr bewegt.

Nach dem Zoomen wird es Zeit für einen Rundflug. Mit der Box links unten im Benutzerinterface kann die Kamera auf ein Land zentriert werden.

Infofenster

Beim Klick auf eine Landesflagge erscheint ein Infofenster mit weiteren Informationen über das entsprechende Land.

Infofenster: Großbritannien

Farbsymbolik

Die Einfärbung der Landesfläche ist ein schneller Hinweis darauf, ob verhältnismäßig viele oder wenige Studierende aus einem Land stammen.

Die Auswahl der Farben funktioniert so:

- Für jede neue Abfrage wird zuerst das Land ermittelt, aus dem die meisten Studierenden kommen. Dieses Land bildet den Maximalwert und steht für 100 %.
- Für jedes weitere Land wird nun die jeweilige Studierendenzahl in einen Prozentanteil dieses Maximalwerts umgerechnet. Ein Beispiel: Aus Deutschland kommen die maximale Anzahl von 100 Studierenden (= 100 %). Aus Österreich kommen 25 Studierende, was 25 % entspricht.
- Dieser Prozentwert wird schließlich einem Farbbereich auf der Farbskala zugewiesen. 0 % befindet sich ganz links im grünen Bereich, 100 % ganz rechts im rot-violetten. Die übrigen Prozentzahlen werden gleichmäßig dazwischen verteilt.


Farbschema für die Einfärbung der Länderpolygone

Kommen aus einem Land für eine Auswahl gar keine Studierenden, verbleibt es als weißer Fleck auf der Landkarte.

Beschreibung des Entwicklungsablaufs

Beschränkung der Darstellung auf wesentliche Informationen

Beim Ausprobieren und Abwägen verschiedener Wege, Studierendendaten auf dem Globus darzustellen, wurde deutlich, dass es für eine übersichtliche Datenvisualisierung nötig sein würde, auf einen großen Teil der Möglichkeiten von Google Earth zu verzichten und zu versuchen, lediglich das notwendige Minimum relevanter Informationen abzubilden. Deshalb werden in der vorliegenden Version der Anwendung gar keine Standarddaten von Google Earth mehr auf dem Globus dargestellt, sondern nur noch die eigenen, bei Initialisierung hinzugefügten Länderpolygone vor einer dunkelblauen Hintergrundgrafik. Außerdem wurde eine Untergrenze für die Zoomstufe definiert, da sich ab einer gewissen Kamerahöhe beim weiteren Hereinzoomen kein Mehrwert an Übersicht oder Information mehr ergeben kann.

Dieser Ansatz der größtmöglichen Vereinfachung findet sich auch in der Gestaltung der Flex-GUI. Sie stellt lediglich die Möglichkeiten bereit, den Globus zu einem Land zu drehen und eine bestimmte Zoomstufe direkt auszuwählen.

Es war keine leichte Entscheidung, zum Wohle der Übersicht das Google Earth-Plugin darauf zu beschränken, eine dreh- und zoombare blaue Kugel mit ein paar Polygonen zu zeichnen, statt hochauflösender, dreidimensionaler Gebirgszüge, australischer Wüsten und dem Amazonas-Delta, aber es war im Sinne der Projektkonzeption richtig, völlig auf diese für die Visualisierung wenig bedeutsamen Elemente zu verzichten.

Dasselbe Gleichgewicht zwischen Informationsmenge und Übersichtlichkeit musste auch bei Darstellung der Länder gefunden werden. Während auf dem afrikanischen Kontinent, auf dem die Staaten groß und relativ gleichmäßig verteilt sind, weitere Angaben auf dem Globus Platz gefunden hätten, wurde dieselbe Informationsmenge bei den flächenmäßig kleinen europäischen Staaten, die aber für die Statistik wesentlich interessanter sind, schnell unübersichtlich.

Die Darstellungsweise der Studierendenzahlen war kein Anlass zu längerer Diskussion. Die Daten werden in Prozentanteile (an dem Land mit der höchsten Studierendenzahl der aktiven Auswahl) umgerechnet, die bereichsweise über eine Skala von 15 Farben kodiert werden. Die jeweilige farbliche Entsprechung des Prozentanteils eines Landes wird dann den zu diesem Land gehörenden Polygonen als Hintergrundfarbe zugewiesen. So lässt sich über die Farbgebung immer ein Eindruck gewinnen, aus welchen Ländern verhältnismäßig viele oder verhältnismäßig wenige Menschen in Osnabrück studierten oder studieren. Größe oder Einwohnerzahl eines Landes spielen dabei keine Rolle, es werden lediglich die absoluten Zahlenwerte zueinander in Beziehung gesetzt.

Ermittlung und Verarbeitung benötigter Daten

Bereits früh im Entwicklungsprozess wurde entschieden, die Liste der in den Stammdaten enthaltenen Staaten mit einer neu zu schaffenden Liste von Länderpolygonen zu verknüpfen, um eine Schnittstelle für die Präsentation der Datenbankdaten auf dem Globus zu schaffen. Daraus ergaben sich die beiden Aufgaben, eine geeignete Liste vorgefertigter Definitionsdaten für Ländergrenzen zu finden und diese den Staatseinträgen aus den Stammdaten zuzuweisen.

Die zur Erfüllung der ersten Aufgabe nötige Suche gestaltete sich wie erwartet als unbequem, aber dann als überraschend erfolgreich. Die Polygondaten der Landesgrenzen sollten aktuell und detailliert genug für eine ordentliche Darstellung sein, dabei aber ausreichend schnell vom Server an den Client übertragbar sein. Da die Google Earth-Software als eigenständige Anwendung schon seit einiger Zeit existiert und das Hinzufügen eigener geographischer Daten in einem maschinenlesbaren Format (KML, siehe unten) erlaubt, das auch vom Google Earth-Plugin gelesen werden kann, konnte auf Definitionsdaten des Projekts "Thematic Mapping"[8] zurückgegriffen werden, die die gewünschten Bedingungen erfüllen und die frei verwendet werden dürfen. (Mehr dazu im Abschnitt "Datenquellen".)

Die zweite Aufgabe, die neu gefundenen Daten mit der Staatenliste aus den Stammdaten zu verknüpfen, erforderte nicht nur an dieser Stelle eine Menge Handarbeit. Die Stammdaten verfügen über einen Länderschlüssel, der in den Kommentaren zur Stammdatenbank als "interne ID des Staates" bezeichnet wird und der keinem bei längerer Suche entdeckten Kodierungsstandard gut genug entspricht, um eine automatische Weiterverarbeitung zuzulassen. Bei den Polygondaten der Ländergrenzen war eine Zuordnung von Hand ohnehin erforderlich, da sie keinen brauchbaren Schlüssel enthielten. Aber auch später, als es darum ging, die Staatenliste um eine weitere Spalte mit einem anderen Kodierungsschema zu ergänzen, konnten die Zuweisungen nicht automatisch erfolgen.

Weiterhin enthält die Stammdatentabelle (DIM_STAATEN) etliche Ländereinträge, die definitiv veraltet sind und weiterhin ein halbes Dutzend Einträge, die auf unterschiedliche Weise ausdrücken, dass die Herkunft eines Studierenden unbekannt ist oder keinem anderen Eintrag sinnvoll zugeordnet werden kann. Deshalb wurden alle Einträge dieser Stammdatentabelle mit Einträgen aus einer neu erstellten, bereinigten Tabelle assoziiert, die nur die aktuell existierenden Ländereinträge mit Verknüpfungen zu zugehörigen Polygondaten enthält. Diese sind es, die auf dem Google Earth-Globus dargestellt werden sollen.

8 http://thematicmapping.org/

Um eine bessere Verwaltbarkeit der Daten sowie höhere Flexibilität bei sich verändernden Landesgrenzen zu erreichen und weil sich die gefundenen Polygondaten dazu anboten, wurde eine zusätzliche Tabelle erstellt, die eine Liste von Gebieten und den zugehörigen Polygondaten enthält. Jedes Gebiet ist mit einem Eintrag der Ländertabelle verbunden und wird auf dem Globus als Teil dieses Landes dargestellt. Jedem Land können auf diese Weise mehr als ein Gebiet zugeordnet werden. Ob diese Trennung letztlich wirklich hilfreich ist, wird sich zeigen.

Die Idee, eine bereinigte Ländertabelle mit genau den aktuell enthaltenen Einträgen (alle aktiven Länder, ein Eintrag "Sonstige" pro Kontinent, ein Eintrag "Sonstige" für die Gesamtwelt) zu schaffen, wurde nicht sofort vollständig umgesetzt. Viele Schwierigkeiten wurden erst spät offenbar, und es wirkte anfangs intuitiver, die neu hinzugefügten Polygon- und Länderdaten über den bereits bestehenden Primärschlüssel (DIM_STAATEN.STA_ID) der Stammdatentabelle zuzuordnen und nicht umgekehrt jede Zeile der Stammdatentabelle mit einer Zeile einer neu hinzugefügten Ländertabelle zu verknüpfen.

Nachdem die neue Ländertabelle schließlich in Gebrauch war, wurde von Hand eine Spalte hinzugefügt, die die Landeskennung jedes Eintrags in einem international standardisierten Kodierungsschema enthält. Anhand dieser Spalte wurde es verhältnismäßig einfach, Metadaten wie die Hauptstädte oder Bevölkerungszahlen einzutragen. (Siehe Abschnitt "Datenpflege".)

Integration des Google Earth-Plugins in die Flex-Anwendung

Beim Versuch, die Google Earth-Komponente wie andere Komponenten als Registerreiter in die Flex-Anwendung zu integrieren, traten einige Schwierigkeiten auf. Da es sich beim Google Earth-Plugin um eine noch neue Technologie handelt, existierte während der Entwicklungszeit keine direkte Schnittstelle zwischen Flex und der Plugin-API. Das Plugin selbst konnte deshalb nicht direkt innerhalb der Flex-GUI dargestellt werden, sondern musste in einem HTML-Layer an der richtigen Stelle über das Flash-Plugin gelegt werden, in dem die Flex-GUI ausgeführt wird. Das verwendete Grundgerüst für diese Umsetzung entstammt dem Blogeintrag "Google Earth + Flex Source Code"[9] von Andrew Trice in der Beispielumsetzung von Johannes Emden. Obwohl diese Form der Umsetzung zum Entwicklungszeitpunkt die einzig praktikable zu sein schien, galt es weitere Probleme zu beheben.

Zwar wurde das Plugin im Rahmen der HTML-Seite oberhalb der Flex-GUI dargestellt, der Inhalt der Infofenster, die beim Klicken auf die Landesflaggen erscheinen, jedoch unterhalb der Flex-GUI. Die Infofenster enthielten also anfangs keine Informationen über das angeklickte Land, sondern schnitten gewissermaßen ein Loch in die Darstellung des Google Earth-Plugins und zeigten den dahinterliegenden Teil der Flex-GUI. Es dauerte einige Zeit, das Problem überhaupt korrekt zu beschreiben. Die Abfolge der Ebenenüberdeckungen ist nicht unbedingt logisch und der hinter dem Plugin liegende Teil der Flex-GUI war zudem durchgängig schwarz gefärbt. Deshalb wurde zuerst nach einer fehlerhaften oder nicht gesetzten Einstellung in der Google Earth-API gesucht und erst danach, um es ausschließen zu können, die Möglichkeit der Überlagerung des HTML-Inhalts der Infofenster durch das Flash-Plugin überprüft. Die seltsame Lösung bestand schließlich unter anderem darin, die Hintergrundeigenschaften des eingebetteten Flash-Plugins auf HTML-Ebene auf "durchsichtig" zu setzen. Dazu musste die (undurchsichtige) Hintergrundgestaltung der Flex-Anwendung in den `<body>`-Tag der HTML-Seite verschoben werden, was unter anderem zur Entstehung des Designs des initialen Ladebildschirms führte.

Eine ähnliche, jedoch ungleich schneller gelöste Schwierigkeit bestand darin, das Google Earth-Plugin über die Flex-GUI ein- und ausblenden zu können. Die Ansätze, den das Plugin beherbergenden HTML-Layer nicht darstellen zu lassen (CSS-Eigenschaft `display`) oder auf der Z-Achse hinter das Flash-Plugin zu legen (CSS-Eigenschaft `z-index`), hatten keinen Auswirkungen oder führten zum Absturz des Plugins. Durch Setzen der Breite des HTML-Layers auf den Wert '0' konnte die gewünschte Funktion zur Ausblendung des Plugins jedoch realisiert werden.

Steuerung des Google Earth-Plugins

Parallel zu den übrigen Aufgaben fand über den Gesamtzeitraum eine stetige Weiterentwicklung derjenigen Projektteile statt, die die Steuerung des Google Earth-Plugins auf Clientseite übernehmen. Insgesamt wurde in diesem Bereich die meiste Zeit investiert, da mit ActionScript/Flex, JavaScript, der Google Earth-API, dem

9 http://www.cynergysystems.com/blogs/page/andrewtrice?entry=google_eath_flex_source_code

KML-Format und (bei seltenen Problemen) HTML/CSS einige Technologien zum Einsatz kamen, die teilweise erlernt werden mussten und deren Gewichtung nicht immer auf Anhieb korrekt einzuschätzen war.

Ohne ins Detail zu gehen, sollen an dieser Stelle die für die Entwicklung der Google Earth-Komponente spezifischen Technologien vorgestellt werden:

JavaScript[10]
: Bei JavaScript handelt es sich um eine Java sehr ähnliche Scriptsprache, die vor allem innerhalb von Webbrowsern zur dynamischen Veränderung des angezeigten (X)HTML-Dokuments eingesetzt wird (Document Object Model Scripting). Die Google Earth-API ist derzeit nur über JavaScript ansprechbar, stellt jedoch ebenfalls eine Art hierarchisches Document Object Model bereit. JavaScript verbindet die Flex-GUI und das Google Earth-Plugin und erfüllt in der Google Earth-Komponente eine wichtige Funktion.

Google Earth-API[11]
: Die von Google bereitgestellte Programmschnittstelle zur Beeinflussung der Darstellung des Google Earth-Plugins. Sie verwaltet die auf dem Globus darzustellenden Objekte und deren Eigenschaften in einer hierarchischen Struktur, die in weiten Teilen eine objektorientierte Nachbildung der Struktur eines KML-Dokuments ist. Änderungen dieser Objekthierarchie wirken sich in der Regel unverzüglich auf die Darstellung des Globus aus.

Keyhole Markup Language (KML)[12]
: Ein im Rahmen der Entwicklung von Google Earth entstandenes XML-Schema zur hierarchisierbaren Speicherung (Ordner-Konzept) von Angaben zu geographischen Orten und von geographischen Visualisierungen in Form von Grafiken oder Polygonen. Daten im KML-Format können vom Google Earth-Plugin direkt geladen und der bestehenden Objektstruktur hinzugefügt werden. Das KML-Format ist der geeignete Weg, größere Datenmengen ans Google Earth-Plugin zu übergeben.

Die Arbeit mit der Google Earth-API benötigte einige Einarbeitungszeit, da die von Google bereitgestellte API-Referenz[13] in weiten Teilen zu knapp formuliert ist und beim Lesen vieler Einträge kaum Rückschlüsse auf den größeren Zusammenhang möglich sind. Ein Beispiel für die Menge an Informationen, die üblicherweise im Abschnitt "Detailed Description" zu finden sind:

> The KmlPlacemark is a feature with associated Geometry. (Google Earth-API, KmlPlacemark[14])

Als beste Herangehensweise im Umgang mit der Objektstruktur stellte sich deshalb schnell das Probieren heraus. Da JavaScript im Umgang mit Datentypen wenig strikt ist und die Google Earth-API-Referenz in der Regel nur den Interface-Typ einer Methodenrückgabe auflistet, wurde es häufig notwendig, die `getType()`-Methode eines erhaltenen Objekts aufzurufen, um herauszubekommen, ob eine JavaScript-Funktion überhaupt den richtigen Objekttyp liefert und an der richtigen Stelle der Objektstruktur arbeitet.

Beim Hinzufügen der Polygondaten und Ländermarkierungen aus der Datenbank stellte sich der Weg über die Objektstruktur schließlich als zu langsam heraus. Die Objekte konnten nicht im JavaScript-Model erzeugt und dann in die Objektstruktur des Google Earth-Plugins übertragen werden. Dieser Ansatz hatte zwar den Vorteil, dass eine Referenz auf das dem Plugin hinzugefügte Objekt im JavaScript behalten werden konnte, die für das Ändern der Polygonfarbe oder das Ein- und Ausblenden von Objekten bei Zoomänderungen gebraucht wird. Doch ab einer gewissen, sehr geringen Anzahl gleichzeitig hinzugefügter Objekte stellte das Plugin nicht mehr alle auf dem Globus dar. Es wurde daher nötig, die Polygondaten und sonstigen Marker ins KML-Format umzuformen und diese KML-Zeichenketten insgesamt ans Plugin zu übergeben. Die entsprechenden Methoden zur Umformung wurden in den Flex-Teil ausgelagert, sodass im JavaScript-Teil nur noch die fertige Zeichenkette ans Plugin weitergereicht werden muss. Allerdings gelingt es auf diese Weise nicht, im JavaScript-Model die Referenzen auf die dargestellten Objekte zu erhalten. Diese müssen bei Datenübergabe im KML-Format nachträglich aus der Objektstruktur des Google Earth-Models gewonnen werden. (Die entsprechende Methode befindet sich als Beispiel im Anhang.)

Da die vom JavaScript-Teil zu leistenden Aufgaben mit der Zeit wuchsen, wurde schließlich die bisherige Sammlung von Einzelfunktionen aufgegeben und zur Steuerung des Google Earth-Plugins im JavaScript eine eigene Model-Controller-Objektstruktur aufgebaut. Die aktuelle Version des Controllers verarbeitet zehn

10 http://de.wikipedia.org/wiki/JavaScript
11 http://code.google.com/apis/earth/
12 http://code.google.com/apis/kml/documentation/
13 http://code.google.com/apis/earth/documentation/reference/index.html
14 http://code.google.com/apis/earth/documentation/reference/interface_kml_placemark.html

verschiedene aus dem Flex-Teil abgeschickte Events und fünf verschiedene Events, die vom Google Earth-Plugin über Callbacks ausgelöst werden. Im Model werden alle zwischen Flex und dem Google Earth-Plugin zu synchronisierenden allgemeinen Statusdaten (etwa die Zoomstufe) gehalten und außerdem eine Liste von Länderobjekten, die zwischengespeicherte, gerade nicht dargestellte Daten jedes Landes enthalten sowie Referenzen auf alle zu diesem Land gehörenden auf dem Globus angezeigten Polygone und Markierungen.

Datenbank-Backend

Datenbankaufbau

Tabelle GE_COUNTRYINFO

ID
: Eindeutiger Primärschlüssel. Jeder Eintrag aus der Stammdatentabelle DIM_STAATEN soll auf eine ID aus dieser Tabelle verweisen. Dieser Wert wird auch dazu verwendet, im JavaScript-Model das entsprechende Land zu referenzieren. Um dabei eine konstante Lookup-Zeit beim Ermitteln eines Landobjekts anhand seiner ID zu gewährleisten, wird der Wert des ID-Feldes als Index im Lookup-Array gesetzt. Um den Speicherverbrauch des Lookup-Arrays gering zu halten, sollten daher große ID-Werte vermieden werden.

ISO3166A2
: Hilfsspalte zur Ermittlung von Landesinformationen nach dem ISO 3166-Standard (ALPHA-2)[15], siehe Beschreibung der Spalte FIPS10. Die Werte dieser Spalte werden in der entsprechenden Zoomstufe als Landeskürzel auf dem Globus dargestellt.

FIPS10
: Hilfsspalte zur Ermittlung von Landesinformationen nach dem FIPS 10-Standard[16]. Tabellen mit Informationen über Länder enthalten in der Regel eine Indexspalte, die ein Land einem Schlüssel eines bestimmten Kodierungsschemas (z. B. ISO 3166[17]) zuordnet. Indem in der hier vorliegenden Tabelle mit jedem Land die entsprechenden Schlüssel verschiedener Kodierungsschemen assoziiert werden, kann das Einlesen von Landesdaten in die Datenbank automatisiert werden. Das hilfreichste Kodierungsschema ist das im World Factbook verwendete FIPS 10-Schema, für das eine Tabelle mit Entsprechungen in anderen Schemen[18] existiert. Demnach besteht die entscheidende, nicht automatisierbare Zuordnung in der Verknüpfung der ID-Spalte mit der FIPS10-Spalte. Die Inhalte aller weiteren Spalten können sukzessive anhand der FIPS 10-Kennung ermittelt werden.

STA_ID
: Vermutlich obsolet, aber vorsichtshalber noch nicht entfernt. Die Zuordnung von Länderkennungen aus den Stammdaten zu Einträgen in dieser Tabelle findet in Tabelle DIM_STAATEN statt. Der Inhalt dieser Spalte sollte nicht verwendet werden.

STA_NID
: Von der Datenbank-Gruppe zur Beschleunigung von SQL-Queries angelegt. (Bei einem Datenupdate muss sicherlich auch diese Spalte an den entsprechenden Stellen angepasst werden, falls sich in DIM_STAATEN.STA_AIKZ etwas ändert. Die Spalte sollte jedoch im Zweifel auch problemlos neu zu erstellen sein, wenn jedem Eintrag aus GE_COUNTRYINFO genau ein aktiver Eintrag aus DIM_STAATEN zugewiesen ist. Siehe auch Abschnitt "Datenupdate".)

IS_COUNTRY
: Bestimmt, ob eine Zeile als Land aufgefasst und dargestellt werden soll. Da die Länderangaben aus den Stammdaten nicht immer eindeutig auf ein Land verweisen, enthält die Tabelle GE_COUNTRYINFO sechs Einträge für Angaben, die entweder einem Kontinent oder nur der Welt insgesamt zugeordnet werden können. Für diese Fälle ist der Wert dieser Spalte '0', für alle anderen '1'. Allgemein gilt: Steht für einen Eintrag in dieser Spalte der Wert '1', kann der Eintrag als Land mit Flagge und mindestens einer Region auf dem Globus angezeigt werden.

NAME
: Der auf dem Globus darzustellende Landesname. Diese Spalte enthält die gebräuchlichen deutschen Landesbezeichnungen (z. B. "Großbritannien" statt "Vereinigtes Königreich").

NAME_US

15 http://de.wikipedia.org/wiki/ISO_3166
16 http://de.wikipedia.org/wiki/FIPS_10
17 http://de.wikipedia.org/wiki/ISO_3166
18 https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-d.html

Hilfsspalte zur Ermittlung von Landesinformationen, siehe Beschreibung der Spalte FIPS10. Im World Factbook wird in Tabellen zumeist nur der Landesname, nicht die FIPS 10-Kennung verwendet. Diese Spalte lässt sich jedoch aus der FIPS10-Spalte ableiten[19].

CONTINENT_ID
: Ordnet ein Land einem Kontinent (GE_CONTINENTINFO.ID) zu. Die Werte wurden aus den Stammdaten übernommen.

CENTER
: Definiert den "Mittelpunkt" (die Position, an der die Landesflagge auf dem Globus gezeichnet werden soll) eines Landes. (KmlCoordinate)

CAPITAL
: Die Landeshauptstadt.

POPULATION
: Die Einwohnerzahl des Landes.

AREA
: Die Landesfläche in km^2.

GDP_PER_CAPITA
: Das Bruttoinlandsprodukt pro Einwohner.

OFFICIAL_LANGUAGES
: Offizielle Landessprachen.

Tabelle GE_CONTINENTINFO

ID
: Eindeutiger Primärschlüssel. Hier dürfen nur die Zahlen 1 bis 5 verwendet werden, da die ID im Quellcode als Arrayindex verwendet wird.

STA_ERDTEIL
: Der diesem Erdteil entsprechende Kontinentschlüssel aus der Stammdatentabelle DIM_STAATEN.

NAME
: Der Name des Kontinents.

CENTER
: Der "Mittelpunkt" des Kontinents (derjenige Punkt, an dem das Placemark auf dem Globus platziert wird). (KmlCoordinate)

Tabelle GE_MAPREGIONS

ID
: Eindeutiger Primärschlüssel, wird gegenwärtig von keiner anderen Tabelle referenziert.

COUNTRY_ID
: Das Land, dem das Gebiet angehört.

BOUNDARIES
: Die Grenzen des Gebiets. (KmlPolygonList)

COMMENT
: Da die Polygondaten allein wenig Aussagekraft besitzen, kann in dieses Feld etwa der Name des Gebiets eingetragen werden. Das Feld dient ausschließlich zu Informationszwecken und hat für die Funktionalität keine Bedeutung. (Da die aktuellen Inhalte dieser Spalte ebenfalls aus der unter "Datenquellen" angegebenen KML-Datei stammen, ist es zufällig möglich, die Spalte BOUNDARIES über die Inhalte der Spalte COMMENT zu füllen. Dies könnte sich unter *gewissen* Umständen bei einem späteren Update als nützlich erweisen: Falls sich in einer kommenden Version des World Borders Dataset die bestehenden Gebietsbezeichnungen nicht ändern, sondern lediglich neue hinzukommen, könnte der Datenbankinhalt automatisch aktualisiert werden. Es könnte sich also lohnen, die Inhalte dieser Spalte nicht zu verändern.)

Speicherformate

Bei jeder Initialisierung der Anwendung werden die Definitionsdaten aller Landesgrenzen vom Server an den Client geschickt. Um das Datenvolumen zu reduzieren, werden die Polygondaten in der Datenbank nicht im KML-Format gespeichert, sondern in einem optimierten Mikroformat, das in diesem Abschnitt definiert wird. Nach der Übertragung werden die Daten im Client wieder ins KML-Format umgeformt.

Die angegebenen Formatangaben müssen exakt so verwendet werden, wie es hier beschrieben wird. Falsch gesetzte oder überzählige Leerzeichen, Semikolons oder Pipes (|) führen fast sicher zu Fehlern, Zeilenumbrüche

19 https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-d.html

sind generell nicht vorgesehen. Vor allem Felder vom Typ KmlPolygonList sollten daher möglichst nicht von Hand editiert werden.

Die Angabe von Koordinatenpaaren richtet sich durchgehend nach der im KML-Format üblichen Abfolge `longitude,latitude` (Längengrad, Breitengrad).

Mit der Java-Klasse `cgp.helper.googleearth.KmlParser` können wohlgeformte KML-Dokumente passender Struktur ins Datenbankformat überführt werden.

KmlCoordinate

Eine geographische Koordinate im Format `longitude,latitude` (Längengrad, Breitengrad).

Formal:

```
c := "long,lat"
```

Beispiel:

```
0.0,5.0
```

KmlLinearRing

Ein geschlossener Ring aus durch jeweils genau ein Leerzeichen voneinander abgetrennten Einträgen im Format KmlCoordinate. Der Ring wird geschlossen, indem als letztes Koordinatenpaar wiederum das erste hinzugefügt wird.

Formal:

```
r := "c1 c2 ... cn c1"
```

Beispiel:

```
0.0,0.0 0.0,5.0 5.0,5.0 0.0,0.0
```

KmlPolygon

Polygone bestehen aus einem äußeren KmlLinearRing und beliebig vielen (auch keinen) inneren Ringen, die Aussparungen in der Polygonfläche definieren. Die Aussparungen werden für Länder benötigt, die andere Länder komplett umschließen (etwa im Fall von Südafrika und Lesotho). Innere Polygone werden durch das Präfix "i:" gekennzeichnet und durch Semikolons vom äußeren Ring und voneinander abgregrenzt.

Formal:

```
p := "r1[;i:r2;...;i:rn]"
```

Beispiel:

```
0.0,0.0 0.0,5.0 5.0,5.0 0.0,0.0;i:1.0,2.0 2.0,3.0 1.0,3.0 1.0,2.0
```

KmlPolygonList

Eine Liste von durch Pipes (|) voneinander abgegrenzten Einträgen im Format KmlPolygon.

Formal:

```
l := "p1[|p2|...|pn]"
```

Beispiel:

```
0.0,0.0 0.0,5.0 5.0,5.0 0.0,0.0|6.0,6.0 8.0,6.0 8.0,8.0 6.0,6.0
```

Datenupdate und Datenpflege

Der einzige Verbindungspunkt zwischen den Stammdaten und den Google Earth-Daten besteht über die Spalte DIM_STAATEN.STA_GE_COUNTRYINFO_ID, die den Stammdaten hinzugefügt wurde, um jede dort enthaltene Staaten-ID einem Eintrag aus der Tabelle GE_COUNTRYINFO zuzuordnen. Falls die Tabelle DIM_STAATEN

während des Updates überschrieben wird, muss die Assoziation zwischen den Spalten STA_ID und STA_GE_COUNTRYINFO_ID vorher gesichert und nachher wieder hinzugefügt werden.

(Siehe außerdem die Erläuterungen zur Spalte GE_COUNTRYINFO.STA_NID.)

Darüber hinaus ist nach einem Datenupdate lediglich darauf zu achten, dass die folgenden Bedingungen erfüllt sind:

Hinweis: Ein Eintrag aus DIM_STAATEN ist dann aktiv, wenn das Feld STA_AIKZ auf 'A' steht.

- Jeder Eintrag der Tabelle DIM_STAATEN soll einem Eintrag aus GE_COUNTRYINFO zugewiesen sein.
- Jedem Eintrag der Tabelle GE_COUNTRYINFO (mit Ausnahme der Sammeleinträge für "sonstige" Fälle) soll genau ein in DIM_STAATEN als aktiv gekennzeichneter Eintrag zugewiesen sein. (Das bedeutet anschaulich, dass so viele Länder auf dem Globus dargestellt werden sollen, wie in DIM_STAATEN als aktiv gekennzeichnet sind.)

Von diesen Regeln abweichende Einträge lassen sich über SQL-Abfragen finden und von Hand bearbeiten. Zwei Fallbeispiele:

Angenommen, die Demokratische Arabische Republik Sahara[20] ist nach einem Datenupdate in der Tabelle DIM_STAATEN als aktiver Eintrag enthalten. Dieser Eintrag muss nun einem Eintrag aus GE_COUNTRYINFO zugewiesen werden, um die eben aufgestellten Bedingungen zu erfüllen. Dazu muss dort ein neuer Eintrag eingefügt werden, da bereits jeder Eintrag aus GE_COUNTRYINFO mit einem aktiven Eintrag aus DIM_STAATEN assoziiert ist. Das Hinzufügen des neuen Westsahara-Eintrags ist glücklicherweise verhältnismäßig einfach, da ein passender eigenständiger Gebietseintrag in GE_MAPREGIONS existiert, dessen COUNTRY_ID lediglich von "Marokko" auf die neue "Westsahara"-ID geändert werden muss. (Zudem muss eine entsprechende Flagge im Ordner "./googleEarth/flags/" der Client-Anwendung hinzugefügt werden. Der Dateiname entspricht der FIPS 10-ID des Landes. Siehe Abschnitt "Datenquellen".)

Wenn sich der Inselstaat Nauru[21] an Neuseeland angliedert, steht er nach dem nächsten Datenupdate als inaktiv in DIM_STAATEN. Eine passende Abfrage würde ergeben, dass dem GE_COUNTRYINFO-Eintrag für "Nauru" kein aktiver Eintrag aus DIM_STAATEN mehr zugeordnet ist. Die Verknüpfungen der Nauru-Einträge in DIM_STAATEN und GE_MAPREGIONS müssten nun auf "Neuseeland" geändert werden, die Nauru-Zeile aus GE_COUNTRYINFO müsste entfernt werden. (Zum Aktualisieren der Landesflaggen siehe erstes Beispiel.)

Sollten die vorhandenen Polygondaten in GE_MAPREGIONS nicht mehr ausreichen, um eine Änderung der poltischen Grenzen nachzuvollziehen (falls sich etwa Bayern[22] für unabhängig erklärt), müssen die Polygondaten entweder von Hand bearbeitet oder es muss ein aktualisierter Satz gefunden werden. Letzteres ist potentiell problematisch, da die Zuordnungen der Gebiete zu genau den Ländern, die in GE_COUNTRYINFO enthalten sind, nicht automatisiert werden können, wenn die Polygondaten nicht mit verwertbaren Schlüsseln versehen sind. (Siehe die Erläuterungen zur Spalte GE_MAPREGIONS.COMMENT für weitere Gedanken.)

Die Landesinformationen können problemlos anhand der Kodierungsschlüssel in der Tabelle GE_COUNTRYINFO aktualisiert werden. Im Serverprojekt liegen etliche Hilfsklassen, die als Beispiel dienen. Vor allem die "Rank Order Pages"[23] aus dem World Factbook sind eine ergiebige, leicht zu verarbeitende Quelle. (Der Schlüssel GE_COUNTRYINFO.NAME_US ist besonders hilfreich.)

Datenquellen

World Borders Dataset[24], Bjørn Sandvik

20 http://de.wikipedia.org/wiki/Demokratische_Arabische_Republik_Sahara
21 http://de.wikipedia.org/wiki/Nauru
22 http://de.wikipedia.org/wiki/K%C3%B6nigreich_Bayern
23 https://www.cia.gov/library/publications/the-world-factbook/docs/rankorderguide.html
24 http://thematicmapping.org/downloads/world_borders.php

Die geographischen Koordinaten zur Definition der Landesgrenzen wurden durch Weiterverarbeiten der Ausgabe der Thematic Mapping Engine[25] gewonnen. Diese Ausgabe basiert auf dem World Borders Dataset, das von Bjørn Sandvik auf Grundlage frei zugänglicher Daten definiert wurde. Die Verwendung des World Borders Dataset unterliegt einer Creative Commons-Lizenz (CC-BY-SA, v3.0[26]). Da die Daten nicht modifiziert, sondern lediglich umformatiert wurden, wird auf erneute Bereitstellung im Sinne dieser Lizenz verzichtet.

The World Factbook[27], Central Intelligence Agency

Der Großteil der dargestellten Landesinformationen sowie die Landesflaggen sind der Onlineversion des von der US-amerikanischen Central Intelligence Agency herausgegebenen World Factbook entnommen. Die Inhalte dürfen frei verwendet werden (public domain[28]).

Wikipedia[29], Wikimedia Foundation

Die dargestellten Landesnamen, Hauptstädte und Landessprachen entstammen der deutschsprachigen Version der Wikipedia.

Anhang

GEController.prototype.flexOnKmlReceived

```
/**
 * Flex-Callback. Nimmt eine Zeichenkette im KML-Format entgegen und laesst
 * diese vom Google Earth-Plugin verarbeiten. Erzeugt daraufhin Entsprechungen
 * der hinzugefuegten Objekte im JavaScript-Model.
 *
 * Vgl. ./src/cgprakt08/util/googleEarth/KmlBuilder.as fuer genaues Format der
 * zu uebergebenden Zeichenkette.
 *
 * @param data Zeichenkette im KML-Format
 */
GEController.prototype.flexOnKmlReceived = function(data)
{
    // m_ge   : Instanz des Google Earth-Plugins
    // folder : Root-Element der uebergebenen KML-Daten
    var folder = this.m_ge.parseKml(data);
    var i = 0;

    if (folder.getId() == "countryObjects")
    {
        // In diesem Paket sind Laenderdaten uebergeben worden.

        var placemarks = folder.getFeatures().getChildNodes();

        // Ein solches Paket enthaelt Zweiergruppen von Landesmarker (Flagge)
        // und zugehoerigem Landespolygon. Beide Objekte sollen von derselben
        // GEPolygonData()-Instanz referenziert werden.
        var marker = true;

        var countryObject;

        // Fuer jeden Eintrag innerhalb der Folder...
        for (i = 0; i < placemarks.getLength(); i++)
        {
            var placemark = placemarks.item(i);

            if (marker)
            {
                countryObject = new GEPolygonData();

                // Definiert Ereignisse fuer den Landesmarker. Die angegebenen
                // Callback-Funktionen rufen wiederum eine Funktion dieses
                // Controllers auf. (Instanzmethoden lassen sich nicht direkt
                // als Callbacks verwenden.)
                google.earth.addEventListener(placemark, 'mouseover',
                        geGoogleOnPlacemarkMouseOver);
```

25 http://thematicmapping.org/engine/
26 http://creativecommons.org/licenses/by-sa/3.0/
27 https://www.cia.gov/library/publications/the-world-factbook/
28 https://www.cia.gov/library/publications/the-world-factbook/docs/contributor_copyright.html
29 http://de.wikipedia.org/

```
46                  google.earth.addEventListener(placemark, 'mouseout',
47                          geGoogleOnPlacemarkMouseOut);
48
49                  countryObject.setReferencePlacemark(placemark);
50              }
51              else
52              {
53                  // Das Description-Feld des Polygons wird zweckentfremdet, um
54                  // zusaetzliche Daten zu uebergeben.
55                  var parts = placemark.getDescription().split(";");
56                  placemark.setDescription("");
57
58                  countryObject.setId(placemark.getId());
59                  countryObject.setReference(placemark);
60                  countryObject.setContinentId(parts[0]);
61                  countryObject.setName(parts[1]);
62                  countryObject.setShortName(parts[2]);
63
64                  this.m_model.polygonData.addItem(countryObject);
65              }
66
67              marker = !marker;
68          }
69      }
70      else if (folder.getId() == "continentMarkers")
71      {
72          // In diesem Paket sind Kontinent-Daten uebergeben worden.
73
74          var placemarks = folder.getFeatures().getChildNodes();
75
76          for (i = 0; i < placemarks.getLength(); i++)
77          {
78              var placemark = placemarks.item(i);
79              this.m_model.continentPlacemarks.push(new Array(placemark.getId(),
80                      placemark));
81          }
82      }
83
84      // Hier werden die in der KML-Zeichenkette definierten Objekte tatsaechlich
85      // ins Model des Plugins eingefuegt.
86      this.m_ge.getFeatures().appendChild(folder);
87  }
```