Technische Universität Hamburg Harburg

Prof. Dr. rer. nat. habil. Ralf Möller

Vergleich der RIA-Technologien zur Implementierung einer Webanwendung zur Vereinheitlichung logistischer Vorgänge

Studienarbeit

Nikita Khristianov

TUHH
Technische Universität Hamburg-Harburg

Eidesstattliche Erklärung

Ich, Nikita Khristianov, Matrikel-Nr. 22736, versichere hiermit, dass ich meine Studienarbeit mit dem Thema

> *Vergleich der RIA-Technologien zur Implementierung einer Webanwendung zur Vereinheitlichung logistischer Vorgänge*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, wobei ich alle wörtlichen und sinngemäßen Zitate als solche gekennzeichnet habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Hamburg, den 16. Februar 2012

Nikita Khristianov

Danksagung

Ich möchte mich ganz herzlich bei allen bedanken, die mich bei der Entstehung dieser Arbeit unterstützt haben. Besonderer Dank gebührt den Herren Prof. Dr. rer. nat. habil. Ralf Möller, Syn Logistics GmbH, Dipl.-Ing. Stepan Osipenko, Frau Oksana Mustuk und meiner Familie.

Abstract

Im Rahmen dieser Arbeit wurden RIA-Technologien zur Entwicklung von Webanwendungen untersucht. Es wurde über die auf dem Markt vorhandenen Frameworke recherchiert. Die Technologien: **Adobe Flex**, **Silverlight** und **Google Web Toolkit** wurden auf ihre Eignung zur Entwicklung einer Webplattform geprüft. Die Plattform sollte dem Austausch von wirtschaftlichen Daten, wie Transportraten zwischen mehreren Teilnehmern, dienen. Die Untersuchung der Technologien wurde durchgeführt bezogen auf die verschiedenen Aspekte der Entwicklung einer Webanwendung:

- **Gestaltung der Benutzeroberfläche**
  Hierbei wurden Instrumente und Klassen-Pakete sowie deren Einsatzszenarien, wie Bindung der UI-Objekte an die Datenquellen oder Datendarstellung in Tabellenform ermittelt.
- **Kommunikation mit externen Diensten**
  Es wurden die Möglichkeiten zum Zugriff auf Daten untersucht, die mit Hilfe von Diensten bereitgestellt werden. Bei den Diensten könnte man generell zwischen den im Rahmen der Anwendungsentwicklung erstellten und den bereits vorhandenen third-party Webdiensten unterscheiden.

Bei der Untersuchung von Google Web Toolkit wurde der Entwurf der Benutzeroberfläche nach dem Model-View-Presenter Muster vorgestellt sowie der Einsatz der für die Datendarstellung optimierten UI-Elemente und Definition des Layouts mit den XML-basierten UI-Binder-Vorlagen. Zur Bindung der Benutzeroberfläche an die externen Datenquellen wurde über den Einsatz der verschiedenen Ansätze, wie Request Factory und GWT-RPC, nachgeforscht.
Für die Entwicklung der Webanwendungen unter Flex wurde der Einsatz von MXML und ActionScript für die Erstellung einer zustandsorientierten Benutzeroberfläche analysiert. Für die Verbindung mit Backenddaten basierend auf den diversen Technologien wurden die Merkmale des RemoteObjekt in Zusammenhang mit dem AMF-Übertagungsprotokoll sowie ein Satz von Instrumenten namens Data Centric Development untersucht. Die erwähnten Ansätze könnten in dem Flash-Builder Entwicklungswerkzeug mit Hilfe der Data/Services-Editor und Designer Ansicht umgesetzt werden.
Für die Implementierung einer reichen browserbasierten Anwendung mit Silverlight wurde XAML zur deklarativen Beschreibung der Benutzeroberfläche untersucht, sowie die Möglichkeiten zur Verbindung mit externen Daten über Dienste unter dem Einsatz von WCF RIA Services, die speziell für die Entwicklung der Webanwendungen konzipiert sind. Für das Ausführen dieser Aufgaben bietet die Entwicklungsum-

gebung Visual Studio eine reiche Funktionalität mit Instrumenten wie DataSources-Editor und Designer Ansicht, sowie Vorlagen zur Definition des Layouts und Erstellung der Dienste basierend auf vorhandenen Backenddaten-Modellen.
Anschließend wurden die Möglichkeiten und Merkmale der untersuchten Technologien basierend auf den vorher definierten Kriterien verglichen. Entsprechend dem Resultat des Vergleichs wurden hervorgehoben die wichtigsten Details der Entwicklung mit Google Web Toolkit betrachtet. Diese wurden anhand einer Beispielanwendung praktisch umgesetzt. Die meisten möglichen Optimierungen zur Darstellung und Manipulation von Daten wurden dabei während der Implementierung integriert, was einen Überblick über die Kooperation verschiedener Komponenten des Frameworks verschaffen konnte.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1 Einleitung

1.1 Motivation

Logistische Vorgänge, wie Güterbeförderung, beinhalten viele unterschiedliche Schritte und erfordern eine Kommunikation und Datenaustausch mehrerer Kontrahenten, wobei die Entscheidung, auf welchem Weg ein Frachtvorgang durchgeführt wird, von den Raten abhängt. Raten für die Durchführung bestimmter Transportschritte werden unter unterschiedlichen Anbietern und Konsumenten dieser Dienste ausgetauscht. Da die Vorgänge mehrere Parteien miteinbeziehen und diese meistens entfernt interagieren, entstehen immer mehr Plattformen zur Vereinheitlichung und Vereinfachung dieser Interaktionen.
Mit seiner raschen Entwicklung bietet World Wide Web die meisten Möglichkeiten zur sinnvollen Umsetzung dieser Ideen. Plattformen wie Ebay[1] können als Beispiel für erfolgreiche Strukturierung eines wirtschaftlichen Vorganges angesehen werden, in diesem Fall ein Besitzerwechsel verschiedener Wertgegenstände auf Auktionsbasis. Mit dem Internetzugang als einem festen Bestandteil der heutiger Wirtschaft ist der größte Vorteil solcher Ansätze die Zugänglichkeit dieser auf Milliarden von Geräten. Um einen konsistenten Austausch von Raten zwischen allen Teilnehmern logistischer Vorgänge zu ermöglichen, wurde die Entwicklung einer reichhaltigen Webanwendung namens Connection Point von Syn Logistics GmbH geplant.
Datenstruktur der workflowbezogenen Kommunikation, Herkunft der Daten und die Manipulation dieser sind teilweise kompliziert, deswegen werden für die Verwaltung dieser Webanwendungen mit entsprechend gewachsener Funktionalität und Interaktionsmöglichkeiten benötigt. Webanwendungen mit komplizierter Anwendungslogik und den Desktopanwendungen ähnlichen Performance und Benutzerfreundlichkeit werden als **Rich Internet Applications** (auch **RIA** vom Eng. reichhaltige Webanwendungen) bezeichnet.
Für die zu entwickelnde Plattform ist es mit dem laufenden Zugriff mehrerer Teilnehmern auf die gleichen Datensätze sowie auch Veränderung dieser zu rechnen. Des Weiteren ist eine ständige Weiterentwicklung der Plattform und ihre Anpassung an sich ändernde Einsatzbedingungen für verschiedene Teilnehmer zu erwarten. Aus diesen Gründen ist für die Implementierung eine RIA-Technologie auszuwählen, die

[1] www.ebay.com

nicht nur die Reichhaltigkeit bei der Datendarstellung und Manipulation dieser, sondern auch Ansätze zur Optimierung für den angesprochenen Einsatzszenario bietet.

1.2 RIA-Technologien

Der Begriff Rich Internet Application ist zwar nirgendwo fest definiert, was angesichts verschwommener Grenzen auch anderer Bestandteile der Internetkommunikation schwierig wäre, aber solche Anwendungen sind leicht zu identifizieren und von herkömmlichen statischen Webseiten zu unterscheiden. RIA's werden derzeit immer öfter eingesetzt, entsprechend der Nachfrage werden verschiedene Lösungen zur Erstellung dieser angeboten. Bei einer großen Zahl von angebotenen Technologien und noch größerer Zahl von verfügbaren Vergleichen dieser, ist eine Auswahl einer dem Einsatzszenario entsprechender Technologie dennoch schwierig. Grundsätzlich können RIA-Technologien in zwei Kategorien unterteilt werden

- **Plugin-basiert**
  Diese brauchen für die Ausführung einen im Browser vorinstallierten Plugin, der eine Umgebung für die eigentliche Anwendung bietet. Dazu zählen **Flex** 3, **Silverlight** 4,**JavaFX**[2] und **OpenLaszlo**[3] mit Flash als Outputvariante
- **HTML- und JavaScript-basiert**
  Diese Anwendungen brauchen für die Ausführung einen Browser mit aktiviertem JavaScript. Dazu zählen Frameworke, wie **Google Web Toolkit**2 und **OpenLaszlo** mit **DHTML**[4] als Outputvariante

Die oben erwähnten Technologien unterscheiden sich in Konzeption sowie auch Eignung für bestimmte Einsatzszenarien. Frameworke, wie z. B. Flex und Silverlight, stellen reiche Möglichkeiten für die Medienintegration, andere, z. B. Google Web Toolkit, dagegen bieten mehrere Wege den Datenaustausch zwischen Server und Klient zu minimieren. Aus diesem Grund müssen einzelne Bereiche der Entwicklung unter dem Einsatz der verschiedenen Frameworke und das daraus resultierende Produkt genau untersucht werden.

[2] http://javafx.com/
[3] http://www.openlaszlo.org/
[4] http://en.wikipedia.org/wiki/Dynamic_HTML

1.3 Ziel der Arbeit

Ziel dieser Arbeit ist ein Vergleich aktuell der auf dem Markt vorhandenen Frameworke und ihrer Tauglichkeit zur Entwicklung einer wirtschaftlichen reichhaltigen Webanwendung (RIA). Der Vergleich wurde auf der Basis der von Syn Logistics GmbH geplanter Entwicklung einer Plattform für den Austausch von Frachtanfragen und Raten zwischen allen Teilnehmern eines Frachtvorganges durchgeführt. Die Anwendung ist für den gleichzeitigen Einsatz an diversen Klienten in unterschiedlichen Sprachumgebungen vorgesehen, wobei Anfragen viele unterschiedliche Eigenschaften besitzen, die von den allen teilnehmenden Klienten laufend geändert werden.

1.4 Untersuchungskriterien

Aus dem Einsatzszenario der Zielanwendung wurden für die Untersuchung folgende Kriterien als relevant abgeleitet:

- Möglichkeiten zur Gestaltung der Benutzeroberfläche
- Darstellung und Aktualisierung großer Datensätze
- Optimierung für den gleichzeitigen Zugriff
- Plattform- und Browserunabhängigkeit
- Entwicklungs- und Wartungsaufwand
- Testbarkeit und Wiederverwendbarkeit der entwickelten Komponenten

Anhand dieser Kriterien wurden JavaFX und OpenLazzlo nicht in den eigentlichen Vergleich miteinbezogen wegen geringer offizieller Unterstützung verschiedener Betriebssysteme und Browser [5][6].

1.5 Verwandte Arbeiten

Da Rich Internet Applications ein derzeit aktuelles Thema ist, gibt es eine Reihe von Arbeiten, die sich mit diesem befassen.
Diplomarbeit von Timo Ernst[7] befasst sich mit Analyse der Performance verschiedener RIA-Technologien und Möglichkeiten diese zu beschleunigen. Es wurde eine Anzahl von Use-Case Tests, z. B. Generierung 20000 Primzahlen, unterteilt in zwei

[5] http://www.openlaszlo.org/requirements
[6] http://docs.oracle.com/javafx/2.0/system_requirements/jfxpub-system_requirements.htm
[7] [Ernst 2010]

Kategorien ausgeführt: API- und Non-API Tests, sowie Focus-Tests zur detaillierten Untersuchung ausgewählter Ergebnisse. Die Basis für den vorgenommenen Vergleich in der Arbeit von Timo Ernst im Unterschied zu dieser Arbeit bildet nicht das Einsatzszenario der Zielanwendung und die daraus resultierenden Kriterien, sondern die Performanceunterschiede der verschiedenen Frameworke beim Ausführen von grundlegenden Operation.
Eine andere Diplomarbeit von Florian Moritz[8] widmete sich derzeitiger allgemeinen Tendenz zur Annäherung von Desktop- und Webanwendungen aus der Sicht des User-Interface. Im Rahmen der Arbeit wurden verschiedene RIA-Technologien gegen verschiedene auf User-Interface bezogene Aspekte verglichen. Darüber hinaus wurde das JavaFX-Framework im Detail untersucht und vorgestellt. Ziel dieser Arbeit ist der Vergleich der Möglichkeiten für die Gestaltung der Benutzeroberfläche zwischen Desktop- und Webanwendungen. Die positive Antwort auf die daraus resultierende Frage des sinnvollen Einsatzes von Webanwendungen anstatt Desktopvarianten dieser kann als einer der Ausgangspunkte für die vorliegende Arbeit angesehen werden.
Gerhard Janisch hat in seiner Diplomarbeit[9] eine Analyse verschiedener RIA-Frameworks durchgeführt. Die Analyse wurde am Beispiel einer Anwendung zur Thesaurusverwaltung durchgeführt und umfasste die damals (2008) vorhandenen Technologien. Diese Arbeit bietet eine allgemeine Übersicht auf die damaligen Möglichkeiten der RIA-Technologien und ermöglicht die Entwicklung der in vorliegender Arbeit vorgestellten in letzten Jahren zu verfolgen.
Die Diplomarbeit von Mirko Kunath[10] stellt die Übersicht über die verschiedenen Werkzeuge zur Entwicklung der Flex-Anwendungen vor. Die Werkzeuge ermöglichen einen Teil der in vorliegender Arbeit vorgestellten Nachteile des Flex-Frameworks umzugehen.

1.6 Aufbau der Arbeit

Kapitel 2 widmete sich der Untersuchung des Google Web Toolkit. Dabei wurden vor allem die vorhanden Möglichkeiten zur Optimierung der Bandbreitenausnutzung sowie die Entwicklung nach dem MVP-Entwicklungsmuster, der eine bessere Testbarkeit der Anwendung und Wiederverwendbarkeit von Komponenten erlaubt.
Kapitel 3 befasst sich mit dem Adobe Flex-Framework und dessen Besonderheiten, wie eine moderne Entwicklungsumgebung zur Erstellung der Benutzeroberfläche sowie Werkzeuge und Komponenten zur schnellen Entwicklung von zustandbasierten Anwendungen.

[8] [Moritz 2010]
[9] [Janisch 2008]
[10] [Kunath 2010]

In Kapitel 4 wurde Microsoft Silverlight untersucht, der ebenfalls, wie Flex, Werkzeuge zur schnellen Entwicklung einer Daten-darstellenden Benutzeroberfläche besitzt, bietet aber eine bessere Modularität der Anwendung.
In Kapitel 5 wird ein Vergleich der untersuchten Technologien durchgeführt. Der Vergleich fand auf der Basis von Kriterien, die als maßgeblich für das Endprodukt identifiziert wurden statt.
Kapitel 6 fasst die Resultate dieser Arbeit zusammen.

2 Google Web Toolkit

2.1 Plattform

In diesem Kapitel wird der **Google Web Toolkit** (GWT) vorgestellt. GWT wurde erstmals in 2006 von Google der Öffentlichkeit zu Verfügung gestellt. GWT ist ein Framework zur Entwicklung der reichhaltigen AJAX-Webanwendungen in Java-Programmiersprache. Die wichtigsten Komponenten von GWT sind: Java-zu-Javascript Compiler, Bibliothek, die einen Teil der Java-Laufzeitumgebung emuliert, und UI-Bibliothek.

2.1.1 Compiler

Java-zu-Javascript Compiler ermöglicht es dem Entwickler die Klient-Seite in Java zu entwickeln, wobei Java-Code in einen optimierten Javascript compiliert wird. Zur Optimierung des Javascript Codes werden verschiedene Techniken angewandt[11] wie z. B. :

- **Dead Code Elimination**
  Zur Kompilierzeit wird der Anwendungscode analysiert und nicht verwendete Anweisungen ausgelassen.
- **Constant Folding**
  Ausdrücke werden mit deren bereits ermittelten Werten ersetzt, falls die Werte zur Kompilierzeit kalkuliert werden können.

Der Ansatz bietet zwar einen optimierten Code, hat aber einen Nachteil, dass der Code nur als Ganzes kompiliert werden kann, da zur Kompilierzeit wird der gesamte Code zur Ermittlung möglicher Optimierungen analysiert. Eine weitere Optimierung bietet **Deffered Binding** 2.1.3

[11] [Kereki 2010, S. 12]

2.1.2 Emulation-Bibliothek für Java-Laufzeitumgebung

Die Bibliothek emuliert eine Untermenge der Java-Laufzeitumgebung-Bibliothek, da GWT-Compiler einen direkten Zugriff zu dem Quellcode jeder in der Entwicklung benutzter Klasse benötigt[12]. Die Pakete der Emulation-Bibliothek beinhalten nur einen Teil der ursprünglichen Klassen, z. B. java.sql-Paket umfasst nur drei Klassen von ursprünglichen sieben zur Verarbeitung von Zeit und Datum.

2.1.3 Deffered Binding

Deffered Binding ist der Hauptmechanismus des GWT-Compilers. Unter Einsatz von Deffered Binding wird spezifischer Code zur Kompilierzeit generiert basierend auf einer Anzahl von Klienteigenschaften. Die wichtigsten Eigenschaften sind Browser und Sprachumgebung. Für jede Permutation der Parameter wird eine separate JavaScript-Instanz erstellt.
Zum einen verringert ein derartiger Ansatz die Größe der Datenpakete, die von dem Benutzer zu laden sind, zum anderen wird der Overhead vermieden, der auf die Auswahl der passenden Implementierung zur Ausfuhrzeit zurückzuführen ist. Außerdem ist Deffered Binding in verschiedene Pakete des GWT-Frameworks integriert, z. B. bei dem Einsatz von GWT-RPC 2.4.3 wird der Code zur Implementierung von Schnittstellen und Proxys automatisch generiert.

2.2 Entwicklungsumgebung

Entwicklung in GWT erfolgt fast ausschließlich in Java. Dies ermöglicht die Benutzung von verschieden Java-IDE's unter Benutzung der entsprechenden Plugins. **Eclipse** wird zu einem meistbenutzten Werkzeug dank **Google Plugin for Eclipse**. Sie bietet eine Möglichkeit die meisten GWT relativen Aufgaben zu erledigen. Außerdem wird eine Installation des Java-Development-Kits benötigt. Im Rahmen dieser Arbeit wurden Eclipse SDK 3.7, Google Plugin for Eclipse 2.3.3 und JDK SE 7 benutzt.
Mit der Version 2.0 des GWT wurde der sogenannte **Development Mode** (Entwicklungsmodus) eingeführt. Im Entwicklungsmodus wird der Code der Compiler- und Serverseite von **Java Virtual Machine**(JVM)[13] als kompilierter Java Bytecode ausgeführt, ohne den in Javascript zu compilieren. GWT ermöglicht somit die

[12] [Kereki 2010, S. 14]
[13] http://en.wikipedia.org/wiki/Jvm

Benutzung von allen Debug-Möglichkeiten ihrer Entwicklungsumgebung und Kommunikation mit der Anwendung in einem Browser während des ganzen Entwicklungsprozesses. In Abbildung 2.1 ist ein typischer Ablauf im Entwicklungsmodus gezeigt.

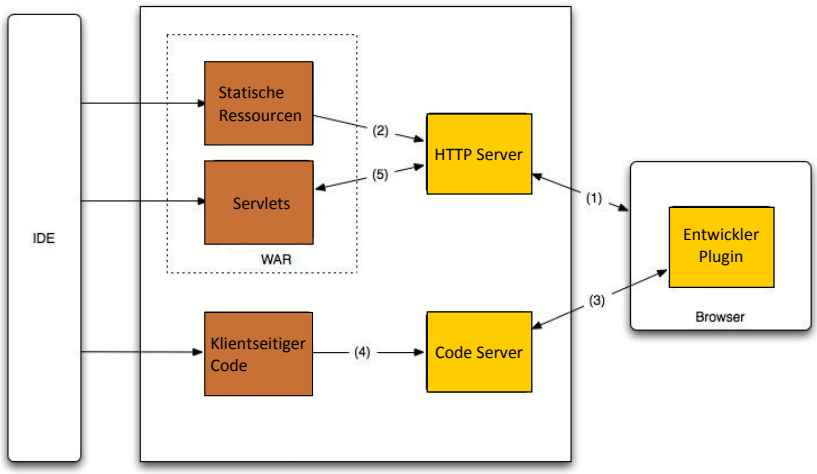

Abbildung 2.1: Typische Sitzung im Entwicklungsmodus
gwtdev

2.3 Benutzeroberfläche

Für die Erstellung dynamischer Benutzeroberflächen bietet GWT den **DOM-Package** an, der die Klassen für den direkten Zugriff auf den Browser-DOM beinhaltet und somit auch eine Möglichkeit die HTML-Seite aus dem Java-Code direkt zu manipulieren bietet. GWT DOM-Klassen erweitern die JavaScriptObjekt-Klasse und können somit ohne einen Größen- oder Geschwindigkeitsoverhead erzeugt und als Javatypen benutzt werden[14].

Elemente der Benutzeroberfläche sind in der GWT UI-Bibliothek beschrieben. Die UI-Bibliothek Klassen sind sehr ähnlich den Klassen anderer grafischen Frameworke für Java, wie **Swing**[15] und **SWT**[16]. Der wichtige Unterschied zu denen ist, dass UI-Objekte (weiter Widgets) nicht in grafische Objekte sondern dynamisch in HTML gerendert werden[17].

Da Widgets größtenteils native HTML-Elemente sind entfällt die Notwendigkeit der browserspezifischen Optimierung. Darüber hinaus bietet GWT selbst fast keine stylingbezogene Lösungen stattdessen wird CSS[18] angewandt. Dies ermöglicht das Aus-

[14] [gwtdom]
[15] http://docs.oracle.com/javase/6/docs/api/javax/swing/package-summary.html
[16] http://www.eclipse.org/swt/
[17] [gwtui]
[18] http://www.w3.org/standards/webdesign/htmlcss

sehen der Anwendung und die Anwendungslogik strikt zu trennen. Die Anzahl der Testvorgänge die auf das Aussehen der Anwendung zurückzuführen sind wird dadurch auch verringert.

2.3.1 Layout

Layout einer GWT-Anwendung wird mit Hilfe von sogenannten Panelen gestaltet. Elemente werden im Unterschied zu anderen UI-Bibliotheken mit Hilfe von HTML ausgelegt. Das Layoutsystem des GWT hat zwei, nicht strikt getrennte, logische Teile: Layoutpanele, die mehr zur Gestaltung der Struktur dienen, und Basispanele wie z. B. FlexTable, die ein HTML <table>-Element erzeugt, oder FormPanel, die enthaltene Widgets in ein <form>-Element einhüllt um das Verhalten des entsprechenden HTML-Objektes zu reproduzieren. Alle Panele können Widgets und auch andere Panele umschließen, da alle Panel-Klassen auch Unterklassen der Widget-Klasse sind.
Das Layoutsystem des GWT ist auf dem nativem CSS-Bedingungssystem aufgebaut und benutzt Eigenschaften wie left, right, width usw. Die unterliegende Logik für die Layoutpanele ist in der Layout-Klasse beschrieben, die die Konsistenz des Layouts in verschiedenen Browsern gewährleistet. Ein wichtige Bedingung für die Benutzung der GWT-Layoutsystems ist der standardkonforme Modus des Browsers, was durch die Dokumenttypdefinition <!DOCTYPE html> erreicht wird[19].
LayoutPanel ist die allgemeine Panel-Klasse die unter Verwendung von Layers (eine Unterklasse der Layout-Klasse für die Festlegung der Position und Große der Widgets) die Kindelemente positioniert. Alle anderen Layoutpanele erweitern die LayoutPanel-Klasse. Eine der erweiternden Klassen ist RootLayoutPanel, die als Container für alle Widgets und Panele der Anwendung dient.
Mit der GWT 2.0 wurden auch zwei Interfaces RequiresResize und ProvidesResize zur Behandlung der mit der Änderung der Fenstergröße verbundenen Ereignisse eingeführt. In meisten Fällen wird die Anpassung der Widgets bei der Größenveränderung des beinhaltenden Fensters vom Browser übernommen. Dennoch bietet GWT mit den obengennanten Interfaces ein Instrument für die Behandlung von größenabhängigen Elementen wie z. B. dynamische Listen und Tabellen.

2.3.2 UI-Elemente

GWT bietet wie auch andere Frameworke eine Vielzahl von vorgefertigten Widgets sowie die Möglichkeit aus bereits bestehenden Bauteilen komplexere Widgets zu erstellen, die sogenannten **Composite**-Widgets. Composite-Widgets umschließen

[19] http://en.wikipedia.org/wiki/Quirks_mode

einen Widget, in der Regel eine Panele, und verhalten sich wie der umgeschlossene Widget. Composite-Widgets sind besonders für wiederverwendbare Formen und Ansichte eine optimale Lösung. Die Klasse des zu erstellenden Widgets muss die Composite-Klasse erweitern.
Komplexe Widgets können von Grund auf neu mit Java oder JavaScript direkt beschrieben werden, was aber nur als keine bevorzugte Alternative wahrgenommen werden soll, da beim Entwurf alle Browserunterschiede in Betracht gezogen werden müssen.

2.3.2.1 Cell-Widgets

Eine Besonderheit des GWT sind sogenannte Cell-Widgets, die speziell für die Darstellung großer Mengen von Daten optimiert sind. Da die Benutzung von innerHTML deutlich schneller als die DOM-Manipulationen ist[20], wird die Oberfläche dieser Widgets mithilfe von innerHTML in einen HTML-String gerendert.
Cell-Widgets sind aus verschieden Cells aufgebaut, die jeweils für das Rendern bestimmter Datentypen zuständig sind. Jeder Celltyp implementiert eine Methode zum Rendern von Daten des entsprechenden Typs in HTML. Es können, genauso wie es mit standardmäßigen Widgets der Fall ist, elementare Celltypen benutzt werden sowie auch Custom-Cells für das Rendern benutzerdefinierter Daten, die eine Kombination der diversen Datentypen sind.
CellTable ist ein cellbasiertes Widget zur Darstellung von Daten im Form einer Tabelle. Beim Einsatz von CellTable müssen die je nach Typ benötigten Spalten (Column) definiert werden, die Cells eines bestimmten Typs für das Rendern der Spaltenelemente umschließen. Ein möglicher Einsatz eines CellTable-Widgets wird in 2.1 dargestellt, dabei wurden zwei verschiedene Spaltentypen eingesetzt:

- eine vom vordefinierten Typ `TextColumn()` für die Darstellung verschiedener Objekteigenschaften
- zweite wurde als Spalte die `CheckboxCell`-Zellentyp umschließt für das Rendern der Checkbox-Elemente und Abfangen der entsprechenden Ereignissen definiert

Listing 2.1: Datendarstellung mit CellTable

```
1
2     ratetable = new CellTable<RateProxy>();
3     ratetable.setPageSize(10);
4     ratetable.setRowCount(10);
5
```

[20] [Koch]

```
6       final SingleSelectionModel<RateProxy> selectionmodel = new SingleSelectionModel<
            RateProxy>();
7       ratetable.setSelectionModel(selectionmodel,DefaultSelectionEventManager.<RateProxy>
            createCheckboxManager());
8       selectionmodel.addSelectionChangeHandler(new SelectionChangeEvent.Handler(){
9
10        @Override
11        public void onSelectionChange(SelectionChangeEvent event) {
12
13          RateProxy selected = selectionmodel.getSelectedObject();
14              if (selected != null) {
15                rateBut.setEnabled(true);
16              }
17        }
18      });
19      initColumns(selectionmodel);
20  .
21  .
22  .
23  private void initColumns(final SelectionModel<RateProxy> selectionmodel) {
24      Column<RateProxy, Boolean> checkColumn = new Column<RateProxy, Boolean>(
25              new CheckboxCell(true, false)) {
26            @Override
27            public Boolean getValue(RateProxy object) {
28              return selectionmodel.isSelected(object);
29            }
30          };
31
32      TextColumn<RateProxy> idCol = new TextColumn<RateProxy>(){
33        @Override
34        public String getValue(RateProxy object) {
35          return object.getId().toString();
36        };
37      };
38  .
39  .
```

Außer den schon oben genannten Konstrukten bietet CellTable Instrumente zur Spaltenverwaltung, Einteilen von geladenen Daten in Seiten sowie zur Definition von Selektierungverhalten innerhalb der Tabelle (siehe Zeile 6 in 2.1).

Verwaltung dynamischer Datenquellen Da die Daten meistens nicht statisch sind und beim Verändern des Anzeigebereichs nachgeladen werden müssen, wird im Falle dynamischer Daten RangeChangeEvent.Handler-Schnittstelle hinzugefügt. Nach der Auslösung des RangeChangeEvent werden neue Daten nachgeladen. GWT bietet zwei

Klassen für die Verwaltung von Datenquellen an, die sich auf der Klient- bzw. Serverseite befinden - ListDataProvider bzw. AsyncDataProvider.

Synchronisation bei Datenveränderung Bei Veränderung der Daten in einem Cell, z. B. über DataPicker-Widget, erfolgt die Synchronisierung mit den Datenquellen über Updater-Schnittstellen. Es wird zwischen ValueUpdater und FieldUpdater unterschieden die für die Synchronisierung für einen CellList bzw. CellTable gesetzt werden. Bei der Veränderung der Daten empfängt der beinhaltende Cell diesen Ereignis über die `onBrowserEvent`-Methode. Darauffolgend wird die `update()`-Methode des entsprechenden Updaters ausgeführt.

2.3.3 Editor Framework

Ein wichtiger Bestandteil der Webanwendungen ist Veränderung der Daten. Hinter den Daten stehen oft Domain-Objekte (auch Bean-Objekte genannt)[21] und deren Eigenschaften.
Editor Framework dient der einfacheren Verbindung zwischen den Bean-Objekten und diese darstellenden UI-Komponenten. Ein Editor ist eine Komposition der Subeditoren. Generell ist jede Klasse, die Editor oder IsEditor Schnittstelle implementiert, ein Editor. Bean-Objekte werden über Driver mit den Editoren verbunden, wobei es zwischen `SimpleBeanEditorDriver` und `RequestFactoryEditorDriver` unterschieden wird.
`RequestFactoryEditorDriver` wird für die Integration mit RequestFactory 2.4.1 eingesetzt, der mit den entsprechenden RequestFactory-Schnittstelle und Editor initialisiert wird (siehe Zeile 43 in 2.2). Synchronisation von Daten erfolgt nach dem Flow Synchronzitation-Muster: Editoren werden mit Objektdaten über `edit(Objekt objekt)` eingepflegt 2.2 und `flush()` überschreibt die Daten von Editor in den Objekt. Während der Ausführung von `flush()` können auch Fehler berichtet werden, die mit `hasErrors()` und `getErrors()` Methoden gehandhabt werden.

Listing 2.2: Einsatz eines Editors

```
1
2
3   public void start(AcceptsOneWidget panel, EventBus eventBus) {
4     //Instanz des RateView
5     impl = new RateViewImpl();
6     impl.setPresenter(this);
7
8     if(RatePlace.REQ.equals(place.getToken())){
```

[21] Objekte mit streng typisierten Getter und optional Setter Methoden zum Abruf von Eigenschaften

```
9         driver. initialize ( clientfactory .getRequestFactory(), impl.getEditor());
10        RateRequest req = clientfactory.getRequestFactory().raterequest();
11        RateProxy rate = req.create(RateProxy.class);
12        // edit gibt eine veränderliche Version des Proxy zurück. Veränderungen werden im
              Rahmen dieses Context akkumuliert.
13        driver.edit(rate, req);
14        req. persist ().using(rate);
15      }
16      ...
17    }
18    .
19    .
20    .
21    public void save(){
22      // flush-Methode des RequestFactoryEditorDriver gibt den während der Aufruf driver.edit(
            rate, req) an den Driver übergebenen RequestContext
23      RequestContext cont = driver.flush();
24      if(driver.hasErrors()){
25        Window.alert("Some error!");
26      }
27      // Anfrage mit allen akkumulierten Aufgaben wird mit save() gestartet
28      cont. fire (new Receiver<Void>() {
29      ...
30      });
31    }
```

Editor Framework stellt mehrere eingebauten Editor-Subschnittstellen für spezialisierte Fälle wie CompositeEditor bereit, der z. B. bei dem Einsatz von CellTable benutzt werden kann. Mit dem Framework wurden einige datentypspezifische Widgets wie IntegerBox, DateLabel und eine Liste von Adapter Klassen der UI-Bibliothek hinzugefügt. Diese stellen eine wiederverwendbare Logik bereit, z. B. ListEditor für das Verwalten einer Liste von Objekten und des zugehörigen Editors.
Die obengenannten Merkmale des Editor-Frameworks verringern die Wahrscheinlichkeit von Fehlern und die Größe der Standardcodeabschnitten bei der Implementierung.

2.3.4 Deklarativer Layout mit UI-Binder

Eine andere Methode das Layout zu beschreiben ist UI-Binder. UI-Binder baut auf der Tatsache auf, dass eine GWT-Anwendung eine Webseite darstellt. Das Layout wird in einer XML-Datei beschrieben, wobei HTML-Elemente sowie auch Widgets als Bausteine benutzt werden. UI-Binder sorgt für eine Trennung des Layouts von der Verhaltensbeschreibung der Benutzeroberfläche. Elementen innerhalb der XML-Vorlage werden Namen vergeben, die im Java Code der dazugehörigen Besitzer-

Klasse z. B. einer View-Klasse aufgerufen werden. Weiterhin für den Zugriff auf die innerhalb der Vorlage beschriebenen Objekte wird die Besitzer-Klasse instantiiert 2.4. Im Unterschied zu deklarativen Ansätzen von Silverlight 4.3.1 und Flex 3.3 wird die UI-Binder Vorlage ausschließlich für das Layout benutzt. Die Besitzer-Klasse stellt einen Composite-Widget dar, der innerhalb anderer Klassen eingesetzt werden kann.

Listing 2.3: UI-Binder Vorlage

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
  xmlns:g='urn:import:com.google.gwt.user.client.ui' xmlns:e='urn:import:com.google.gwt.editor.
      ui.client'>
  <ui:style src="../common.css">
  </ui:style>

  <g:HTMLPanel>
    <div style="{style.rightAlign}">
      <table>
        <tr>
          <td>Pickup:</td>
          <td>
            <e:ValueBoxEditorDecorator ui:field="pickup"
              stylePrimaryName="{style.editField}">
              <e:valuebox>
                <g:TextBox ui:field="pickupBox" stylePrimaryName="{style.editField}" />
              </e:valuebox>
            </e:ValueBoxEditorDecorator>
          </td>
        </tr>
      .
      .
      .

      </table>
    </div>
  </g:HTMLPanel>
</ui:UiBinder>
```

Listing 2.4: Besitzer-Klasse der UI-Binder Vorlage

```
public class RateEditor extends Composite implements Editor<RateProxy>{

  interface Binder extends UiBinder<Widget,RateEditor>{}

  @UiField
  ValueBoxEditorDecorator<String> pickup;
  @UiField
  ValueBoxEditorDecorator<String> loading;
```

```
9    @UiField
10   ValueBoxEditorDecorator<String> turnin;
11   @UiField
12   ValueBoxEditorDecorator<String> client;
13   @UiField
14   ValueBoxEditorDecorator<String> service;
15   @UiField
16   ValueBoxEditorDecorator<String> price;
17
18   public RateEditor() {
19     //Wickelt die aus dem XML−Markup generierte Oberfläche in einen Composite.
20     initWidget(GWT.<Binder> create(Binder.class).createAndBindUi(this));
21   }
22 }
```

Um Widgets im UI-Binder Vorlagen benutzen zu können muss der UI-Package des GWT mit einem Prefix des XML-Namensraums verbunden werden. Auf alle Widget-Methoden, die für das Setzen bestimmter Attribute zuständig sind, z. B. `setVisibleRowCount()` für einen Listbox, kann aus der Vorlage über ein Attribut direkt zugegriffen werden.
UI-Binder vereinfacht auch die Ereignisverwaltung. @UiHandler Annotation ersetzt den standardmäßigen Code für die EventHandlers, wie es oben beschrieben wurde für den Fall des klassischen Layouts. Dies angesichts einer meistens großen Anzahl verschiedener Ereignisse und EventHandler verringert die Größe der Codes.
CSS-Stile können mit <ui:style> eingefügt werden. Es können Stile innerhalb der Vorlage definiert oder, wie es meistens der Fall ist, auf bzw. externe Stilvorlagen zugegriffen werden, was über den Attribute `src` bzw. `with` bestimmt wird.
UI-Binder ermöglicht nicht nur eine intuitive, sondern auch eine effiziente Auslegung der UI-Elementen. Wie schon oben erwähnt wurde ist eine Aufstellung der DOM-Struktur im Browser über innerHTML-Attribute schneller als über Schnittstellenzugriffe, wovon UI-Binder profitiert[22]. Weiterhin bietet UI-Binder auch Unterstützung für die Internationalisierung 2.6.4.

2.3.5 Aussehen und Design

GWT setzt für das Bestimmen des Aussehens der UI-Elementen **CSS** ein. Jede Widget-Klasse der UI-Bibliothek besitzt einen vorgegebenen CSS Stil der nach der Konvention gwt-<classname> benannt wird. Stile werden in der CSS Datei der Anwendung beschrieben und können für die ganze Klasse, z. B. Aussehen aller Labels über gwt-Label, geändert werden. Einzelnen Elementen können auch unikale IDs vergeben werden und zu der ID zugehöriger Stil beschrieben werden. Stil wird einem

[22] [gwtuibin]

Widget über `setStyleName` zugewiesen, ein weiterer Stil mit `addStyleName()` bzw. `removeStyleName()` hinzugefügt bzw. entfernt werden.

2.4 Kommunikation mit externen Datenquellen

Für die Kommunikation mit externen Datenquellen bietet GWT drei Werkzeuge: **GWT-RPC** Framework, ein Instrument für trasparente RPCs zu Java Servlets, **HTTP-Klientklassen** für beliebige XMLHttp-Anfragen und **RequestFactory**, eine Alternative zum GWT-RPC für datenorientierte Anwendungen.

2.4.1 RequestFactory

Businessanwendungen sind datenorientiert und auf **CRUD**-Funktionalität auf Domainobjekten angewiesen, die für eine GWT-Anwendung mit RequestFactory realisiert werden kann. RequestFactory wird von Domainobjekten und Diensten serverseitig und von Proxys und Anfragenschnittstellen klientseitig repräsentiert. Klient- und serverseitige Komponenten sind mittels Annotationen aufeinander eingezeichnet. Bei der Implementierung wird zwischen Domainobjekten und Diensten unterschieden.

2.4.1.1 Domainobjekte

Es wird zwischen zwei Arten von Domainobjekten unterschieden: Entities und Wertobjekten, wobei Entities stets eine eindeutige ID und Version besitzen und Wertobjekte nicht. Wertobjekte stellen oft eingebettete Objekttypen dar. Auf der Klientseite wird dementsprechend zwischen EntityProxy und ValueProxy Schnittstellen zum Zugriff auf Domainobjekte unterschieden. Proxys enthalten nur Getter und Setter-Methoden.
Ein besonderer Vorteil der RequestFactory ist, dass bei einem klientseitigen Aufruf einer Setter-Methode, nur die eigentlichen Änderungen übertragen werden, was bei Objekten mit vielen Eigenschaften besonders zur Geltung kommt. Beim Erstellen, Updaten oder Löschen eines Domainobjektes auf dem Server wird wiederum die Klientseite über einen Ereignis benachrichtigt. Ein weiterer Vorteil gegenüber GWT-RPC ist, dass Domainobjekte sich komplett auf der Serverseite befinden und nicht in Javascript übersetzt werden müssen, somit müssen die auf der Klientseite existierende Limitierung nicht eingehalten werden.

2.4.1.2 Dienste

Dienste enthalten Methoden die Domainobjekte als Argumente oder Rückgabetyp benutzen. Auf der Klientseite wird eine Erweiterung der RequestFactory-Schnittstelle definiert. Die Schnittstelle beinhaltet Methoden, die verschieden Anfragenschnittstellen zurückgeben. Wobei die RequestContext-Schnittstellen Deklarationen zum eigentlichen Zugriff auf Domainobjekte definieren. Die Rückgabetypen der Methoden innerhalb der RequestContext-Schnittstelle sind Request-Objekte und nicht Domainobjekte selbst, z. B. eine `Request<Long> countRates()` auf der Klientseite entspricht der `Long countRates()`-Methode auf der Serverseite. Ähnlich wie bei einem RPC-Aufruf werden so asynchrone Methodenaufrufe über `Request.fire()` möglich, wobei beim Aufruf ein Receiver-Objekt im Unterschied zu dem den CallBack-Objekt bei einem RPC-Aufruf übergeben wird.

2.4.1.3 Einsatz der RequestFactory

RequestFactory Schnittstelle wird über `GWT.create()` erstellt und muss mit dem EventBus für die Abfertigung von `EntityProxyChange`-Ereignissen initialisiert werden. Für die eigentliche Anfrage wird die RequestContext-Schnittstelle über den Aufruf einer Methode der RequestFactory-Schnittstelle erstellt. Die erstellte RequestContext-Schnittstelle stellt eine Art Anfrageerzeuger dar. Die für die Abfrage relevante EntityProxy wird über `create()`-Methode im Rahmen der Schnittstelle erstellt, genauso werden alle relevanten Setter- und Dienstaufrufe eingereiht. Die Anfrage wird dann über die `fire()`-Methode des RequestContext ausgeführt, die nur einmal pro RequestContext aufgerufen werden kann. Zusätzlich kann die Anfrage über `with()`-Methode auf bestimmte Objekteigenschaften verfeinert werden.

2.4.2 Validierung

Für die Validierung von externen Daten unterstützt GWT mit RequestFactory sowie auch mit Editor Framework die **JSR 303**[23], ein Framework zur Validierung von **JavaBeans**[24]. Dies ermöglicht die Validierungsregeln auf der Serverseite zu Verwalten, dabei werden auf der Klientseite nur Annotationen benutzt.

2.4.3 GWT-RPC

Da GWT-Anwendungen AJAX-Anwendungen sind, werden keine neuen HTML-Seiten während des Anwendungslaufs vom Server geladen sondern nur die server-

[23] http://jcp.org/en/jsr/detail?id=303
[24] http://de.wikipedia.org/wiki/JavaBeans

seitige Dienste und Daten über RPC aufgerufen. Für den Zugriff auf diese baut GWT-RPC auf den Java Servlets auf, wobei klient- und serverseitiger Code für das Serialisieren der Objekte basierend auf dem Deferred Binding eingebaut ist.
Um auf einen serverseitigen Dienst zuzugreifen müssen auf der Klienseite zwei Schnittstellen definiert werden: eine, die die RemoteService-Schnittstelle erweitert und alle Methoden auflistet. Auf der Serverseite muss eine RemoteServiceServlet erweiternde Klasse definiert werden, die die klientseitige Schnittstelle implementiert.
Um asynchrone Aufrufe zu tätigen wird noch ein auf der synchronen Schnittstelle basierende asynchrone Schnittstelle gebraucht. Methoden des asynchronen Interface haben keinen Rückgabewert, sondern ein Callback-Objekt als Argument, der bei einem asynchronen Aufruf übergeben wird. Über das Objekt wird der Aufrufende sobald der Aufruf erfolgt oder misslungen ist benachrichtigt. Für den eigentlichen Aufruf werden zunächst ein Klientproxy zu dem entsprechenden Servlet instantiiert und ein Callback-Objekt erstellt. Danach wird die benötigte Methode aufgerufen. Für das Callback-Objekt werden `onSuccess` und `onFailure` Methoden definiert für den Fall eines erfolgreichen Aufrufs bzw. einer Ausnahme.

2.4.4 HTTP-Anfragen

Webanwendungen werden oft in bereits laufenden Umgebungen integriert, die möglicherweise die Benutzung von GWT-RPC ausschließen. Für den Fall, dass keine Java-Servlets benutzt werden können, erlaubt GWT direkte AJAX-Aufrufe und Bearbeitung von Antworten in verschiedenen Formaten, wie XML oder JSON.
Eine wichtige Beschränkung dabei ist die **Same Origin Policy** (SOP)[25], die es Javascript nicht erlaubt auf Daten von URLs, die nicht mit der Quelle übereinstimmen(Protokoll, Host und Port), zuzugreifen. Eine GWT-Anwendung greift im Entwicklungsmodus standardmäßig nicht auf den Port 80 zu, deswegen wird ein Zugriff auf einen Webservice der auf dem Port 80 läuft, nicht erfolgen. Der compilierte Code der Anwendung wird dabei nicht von SOP beschränkt, da der auch auf dem Port 80 läuft. Das wird mit dem setzen des Parameters `-noserver` für die Entwicklungskonfiguration gelöst. Da oft auch ein Zugriff auf andere Hosts oder ein anderer Protokoll benötigt werden, muss auf der Serverseite ein Proxy eingerichtet werden. Es gibt zwar andere Lösungen, aber diese lösen das Problem nur teilweise.
RequestBilder-Klasse bietet die Kernfunktionalität für einen XmlHttpRequest. Dem Klassenkonstruktor werden die Art der HTTP-Anfrage und das URL übergeben. Benutzername, Password usw. können für das RequestBuilder-Objekt über entsprechende Methoden gesetzt werden. Die `sendRequest()`-Methode wird aufgerufen um die eigentliche Anfrage zu senden. Da GWT nur asynchrone Serverabrufe erlaubt,

[25] http://www.w3.org/Security/wiki/Same_Origin_Policy

wird dabei ähnlich wie bei einem RPC-Aufruf ein RequestCallback-Objekt übergeben, für den die `onError(Request, Throwable)` und `onResponseReceived(Request, Response)` Methoden definiert werden. Für die Weiterverarbeitung der Server Antworten im XML bzw. JSON Format stehen entsprechende Bibliotheken zu Verfügung.

2.5 Programmier-Model

Entwicklung der Webanwendungen mit GWT erfolgt nach dem **Model-View-Presenter** (MVP) Entwurfsmuster. MVP-Muster ist aus dem **Model-View-Controller** (MVC)[26] Muster entstanden, wobei die Rollen des Views und des Presenters, der den Controller im MVC ersetzt hat, sich geändert haben. Presenter ist zu dem Verbindungsglied zwischen Model und View geworden, wobei View und Model keinerlei Kenntnis von einander haben. Im Gegensatz zu MVC steht beim MVP-Muster die verbesserte Testbarkeit und auch die strengere Trennung der Komponenten im Vordergrund[27]. Der Auswahl dieser Vorgehensweise ermöglicht die überwiegende Benutzung von **JUnit**-Tests[28] statt **GWTTestCase**. GWTTestCase-Klasse stellt eine Art Brücke zwischen JUnit und GWT-Umgebung. Nachteil von GWTTestCase ist, dass diese in einem Browser[29] ausgeführt werden und bis zur 100-mal langsamer als JUnit-Tests sind[30].

2.5.1 GWT-MVP

Mit der Version 2.1 des Toolkits wurde ein Framework zu Verfügung gestellt, das auf dem klassischen MVP-Model aufbaut und Klassen zur einfacheren Entwicklung nach diesem Muster sowie zur Verwaltung der Browser-Chronik bereitstellt. Die Hauptbausteine des Frameworks sind Activity und Place Klassen, sowie mehrere Klassen und Schnittstellen zur Steuerung des Ereignisflusses zwischen verschiedenen Zustanden der Anwendung, wie PlaceHistoryHandler, ActivityManager Klassen.

2.5.1.1 Place

GWT-Anwendungen sind AJAX-Anwendungen und laufen in einem Fenster, der dynamisch generiert und verändert wird. Somit ist Navigation mit klassischer Browsersteuerung ohne weiteres nicht möglich. Dieses Problem wird mit Hilfe von **Place-**

[26] http://de.wikipedia.org/wiki/Model_View_Controller
[27] [Kereki 2010]
[28] http://de.wikipedia.org/wiki/JUnit
[29] [Ramsdale 2010]
[30] [gwtunit]

Klassen gelöst. Jede Place-Klasse stellt einen bestimmten Zustand der Anwendung dar und wird vom PlaceHistoryHandler mit Browser-URLs synchronisiert. Somit ist ständig die Möglichkeit gegeben zwischen den jetzigen und vorherigen Zuständen zu navigieren oder Lesezeichen zu bestimmten Stellen der Anwendung zu setzen.
Für jeden Place muss eine Implementierung der PlaceTokenizer-Schnittstelle definiert werden, die für das Serialisieren eines Zustandes in einen Token zuständig ist. Die Tokens werden je nach Zustand an das Haupt-URL der Anwendung angehängt. Falls kein Zugriff via URL für einen bestimmten Place sinnvoll ist, kann die Place-Klasse den BasicPlace erweitern. Die Klasse stellt einen vorgegeben PlaceTokenizer zu Verfügung, der einen `null`-Token zurückgibt.
Die Verbindung zwischen PlaceHistoryHandler und PlaceTokenizer verschiedener Places ist eine Erweiterung der PlaceHistoryMapper-Schnittstelle. Diese beinhaltet eine Aufzählung aller Places innerhalb der Anwendung.

2.5.1.2 Activity

Activity ist der von GWT benutzter Begriff für Presenter. Activities sind das Verbindungsglied zwischen Model und View und umfassen die Steuerung der Views sowie die Kommunikation der beiden Komponenten. Für jede Activity-Klasse werden `start()` und `goTo()`- Methoden definiert. Die erste davon wird beim Starten der Activity von ActivityManager ausgeführt und die Instanziierung der entsprechen Views, Steuerung bei bestimmten Ereignissen und Bereitstellung von Serverdaten umfasst. Die `goTo()`-Methode ruft die gleichnamige Methode des PlaceControllers auf und löst damit den Übergang zu einem anderen Place der Anwendung. Die typischen Vorgänge innerhalb einer MVP-Anwendung werden im 2.5.1.5 beschrieben. Der unten aufgeführte Codeabschnitt präsentiert eine Activity, die den View zur tabellarischen Darstellung der Domainobjekte 2.3.2.1 steuert.

Listing 2.5: Activity-Klasse

```
1  package rateexchange.client.mvp;
2
3  public class RatesTableActivity extends AbstractActivity implements Presenter{
4    private ClientFactory clientfactory;
5
6    public RatesTableActivity(RatesTablePlace place, ClientFactory clientFactory){
7      this.clientfactory = clientFactory;
8    }
9
10   @Override
11   public void start(AcceptsOneWidget panel, EventBus eventBus) {
12     final RatesTableView ratesView= clientfactory.getRatesTable();
```

```
13      //Setzen des dataprovider des Ansichts
14      RateDataProvider dataProvider = new RateDataProvider(clientfactory);
15      ratesView.setDataProvider(dataProvider);
16      //Einrichtung der beidseitiger  Kommunikation zwischen View und Activity.
17      ratesView.setPresenter(this);
18      RateRequest ratesreq = clientfactory.getRequestFactory().raterequest();
19      ratesreq.countRates().fire(new Receiver<Integer>() {
20        @Override
21        public void onSuccess(Integer response) {
22          ratesView.setRowCount(response);
23        }
24      });
25    //Übergabe des View an den Container-Widget
26    panel.setWidget(ratesView.asWidget());
27    }
28    @Override
29    public void goTo(Place place) {
30       clientfactory .getPlaceController().goTo(place);
31
32    }
33
34  }
```

Mit der `setPresenter()` wird der View-Instanz die betreibende Activity gesetzt. Der Aufruf der `setWidget()`-Methode übergibt die vorbereite View-Instanz dem Container-Widget. Dieses schließt die ganze Oberfläche der Anwendung um. Über das `panel`-Attribut wird der Container der aktuell ausgeführten Activity übergeben.

2.5.1.3 View

Da MVP-Entwurfsmuster eine strikte Trennung der logischen Bausteine vorsieht, beschreiben Views ausschließlich das Layout der Benutzeroberfläche und die mit Benutzeroberfläche zusammenhängenden Ereignisse, z. B. Knopfdruck. Somit wird die Voraussetzung für einen transparenten Austausch der View-Implementierungen ohne irgendeinen Einfluss auf den restlichen Code gegeben. Die Erweiterung der IsWidget-Schnittstelle (wird von allen Widgets auch Composite-Widgets 2.3.2 erweitert) bietet über die Methode `asWidget` den Zugriff auf den von dem View bereitgestellten Widget und darüber hinaus die Möglichkeit während der JUnittests eine Dummy-View Instanz zu erstellen [31].
Um eine beidseitige Kommunikation zwischen View und Activity zu erlauben, wird

[31] http://google-web-toolkit.googlecode.com/svn/javadoc/2.1/com/google/gwt/user/client/ui/IsWidget.html

innerhalb der View-Schnittstelle die Presenter-Schnittstelle für die zugehörige Activity definiert. Die Presenter-Schnittstelle wird von der Activity implementiert. Die Gestaltung der Benutzeroberfläche wird im Kapitel 2.3 näher beschrieben.

2.5.1.4 ClientFactory

Der Einsatz relativ rechenintensiver DOM-Manipulationen[32] für die Erzeugung von Widgets innerhalb eines Views macht eine Benutzung von ClientFactory sinnvoll[33]. ClientFactory verwaltet Referenzen auf die in der Anwendung benutzen wiederverwendbaren Objekte, wie Views, Eventbus u. a. Ein weiterer Vorteil ist die Möglichkeit verschiedene ClientFactory-Implementierungen abhängig von der Klienteigenschaften zu benutzen, was im Zusammenhang mit Deffered Binding effizient ist. Die entsprechenden Einstellungen sind der Konfigurations-XML-Datei hinzuzufügen 2.6. ClientFactory-Schnittstelle wird dabei über `create()`-Methode via Deffered Binding instantiiert.

Listing 2.6: Definition einer Klassenersetzung

```
1  <replace-with class="rateexchange.client.ClientFactoryImpl">
2    <when-type-is class="rateexchange.client.ClientFactory"/>
3  </replace-with>
```

2.5.1.5 MVP Lebenszyklus der Anwendung

Nach dem Start der Anwendung wird das vorgegebene Place vom PlaceController instantiiert. Dies wird über das PlaceChange-Ereignis angekündigt. Darauf folgend wird die `start()`-Methode der entsprechenden Activity vom ActivityManager aufgerufen. Nach dem Initiieren eines Übergangs zu einem anderen Place wird die `goTo()`-Methode vom PlaceController aufgerufen, der nachfolgend den ActivityManager via PlaceChangeRequest-Ereignis informiert.
Ein Vorteil der GWT Activities sind die automatischen Warnungen, falls eine Activity gestoppt wird, z. B. bei der Navigation zu einem anderen Place oder beim Schließen des Fensters. Nach der Auslösung des PlaceChangeRequest-Ereignisses wird die Methode `mayStop()` vom ActivityManager aufgerufen und, falls das Anhalten der Activity vom Benutzer erlaubt wurde oder der Rückgabewert der Methode auf null gesetzt ist(in diesem Fall wird keine Warnung ausgegeben), wird die Navigation von dem PlaceController zum entsprechenden Place instantiiert und das PlaceChange-Ereignis wird erzeugt. Als Reaktion auf das Ereignis aktualisiert PlaceHistoryManager die URL-Chronik und der ActivityManager startet die mit dem Place verknüpfte

[32] [gwtact]
[33] [Louis und Müller 2007, S. 729]

Activity. Mapping von Places auf zugehörige Activities wird in der ActivityMapper-Klasse beschrieben.

2.6 Lokalisierung

Die Lokalisierung von GWT-Anwendungen basiert auf der Klienteigenschaft locale, die diversen Sprachumgebungen entspricht. Die Eigenschaft wird entweder über den eingebetteten Metatag in der HTML-Seite oder über den Anfragen-String des URLs der Hostseite gesetzt. Die zur Kompilierzeit mitzuberücksichtigenden Sprachumgebungen werden im Konfiguration-XML-Datei definiert. Es wird zwischen statischen und dynamischen String-Internationalisierung unterschieden, die entsprechend dem Einsatzszenario der Anwendung benutzt werden.

2.6.1 Lokalisierung von Datum und Zahlendarstellung

Wie schon erwähnt wurde, umfasst die GWT-Emulationsbibliothek nur einen Teil der ursprünglichen Klassen. Davon sind auch Klassen zur Formatierung von Daten und Zahlen betroffen, wie java.text.DateFormat und java.text.NumberFormat. Die Funktionalität dieser Klassen wird teilweise über die DateFormat bzw. NumberFormat Klassen des com.google.gwt.i18n.client-Moduls verfügbar. Der Unterschied liegt daran, dass GWT-Klassen das Umschalten zwischen verschiedenen Sprachumgebungen zur Laufzeit ermöglichen. Das Laden der für die aktuelle locale relevanter Logik geschieht automatisch via Deffered Binding 2.1.3.

2.6.2 Statische Internationalisierung

Statische String-Internationalisierung ist eine auf streng definierten Java-Schnittstellen und Properties-Datei basierende Methode. GWT bietet drei Schnittstellen zur Verwaltung von Konstanten Strings und Strings, die variable Parameter beinhalten: Constants, ConstantsWithLookup und Messages. Um die Internationalisierung zu erlauben werden entsprechende statischen Werte durch Methoden ersetzt. Methoden werden in einer Erweiterung einer der oben beschriebenen Schnittstellen definiert. Für jede Sprachumgebung wird eine der Schnittstelle zugewiesene Properties-Datei angelegt, die eigentliche Übersetzungen beinhaltet.
Im Falle von Strings mit variablen Parametern wird die Messages-Schnittstelle benutzt. Die Schnittstelle erlaubt der Einsatz von Platzhaltern in den zu übersetzenden Strings für die Parameter der Schnittstellenmethoden. Die Parameter können vor dem Einfügen formatiert werden.
Die Benutzung der statischen Internationalisierung ist bevorzugt, da die Deffered

Binding dabei zum Einsatz kommt. Da locale eine Klienteigenschaft ist, wird zur Kompilierzeit ein separater Code pro Lokale/Benutzerklient-Paar erstellt. Dies erlaubt dem Benutzer nur die lokalisierte Version der Anwendung herunterzuladen. Ein weiterer Vorteil ist, dass Fehler bei Parameter-abhängigen Strings während der Entwicklung entdeckt werden.

2.6.3 Dynamische Internationalisierung

Für den Fall, dass das GWT-Modul in eine mit bereits definiertem Lokalisierungsprozess Webanwendung eingebaut wird, bietet GWT die Dictionary-Klasse. Die entsprechenden lokalisierten Strings werden als Javascript zu Laufzeit in die HTML-Seite eingefügt.

2.6.4 Internationalisierung mit UI-Binder

Bei dem Einsatz des UI-Binders für die Gestaltung der Benutzeroberfläche werden die zu lokalisierenden Strings der XML-Vorlage mit dem <ui:msg>-Tag markiert. HTML-Elemente innerhalb dieser Strings werden weiterhin erkannt, wobei auch einzelne Attribute der HTML-Elemente lokalisiert werden können, z. B. description einer Bilddatei. Übersetzungen werden ähnlich wie bei statischen Internationalisierung von Strings in speziell formatierten Properties-Dateien definiert.

3 Adobe Flex

3.1 Plattform

In diesem Kapitel wird **Adobe Flex** vorgestellt, das 2004 erstmal als Präsentationsserver[34] erschien. Ab der Version 2.0 wurde Flex in ein Framework zur Erstellung der Webanwendungen umgewandelt. Flex wurde im Laufe der Zeit zum Herzstück einer Reihe von Produkten, die eine komplette Lösung zur Entwicklung der reichhaltigen Webanwendungen und Seiten bietet. Darin sind **Flex**, **Flash Builder**[35], **Flash Catalyst**[36] und **LiveCycle**[37] enthalten.
Die wichtigsten Bestandteile von Flex sind die Layout-Sprache **MXML** 3.3.1, die auf dem XML basiert, und die **ActionScript**-Programmiersprache 3.1.1. Flex-Anwendungen bestehen aus den MXML- und ActionScript-Quelldateien, die nach dem Kompilieren in eine **SWF** bzw. **AIR**-Anwendung[38] verpackt werden. Die Anwendung wird in einer **Flash-Player**-Umgebung[39] im Browser bzw. AIR-Laufzeitumgebung auf dem Desktop ausgeführt. AIR-Anwendungen werden im Rahmen dieser Arbeit nicht weiter betrachtet.

3.1.1 ActionScript

ActionScript ist eine auf dem **ECMAScript 4**[40] basierte Programmiersprache für Flex und Flash-Anwendungen. Seit der Version 3.0, die zum Zeitpunkt der Anfertigung dieser Arbeit aktuell war, ist ActionScript zu einer objektorientierten Sprache geworden, was deren Einsatz in komplexen Anwendungen ermöglichte[41]. Dem ActionScript fehlen manche Konzepte im Vergleich zu anderen objektorientierten Sprachen, wie z. B. abstrakte Klassen oder private Konstruktoren, was als Nachteil angesehen werden kann[42]. Diese Einschränkungen stammen aus dem ECMAScript

[34] http://en.wikipedia.org/wiki/Adobe_Flex#Macromedia_Flex_Server_1.0_and_1.5
[35] http://www.adobe.com/products/flash-builder.html
[36] http://www.adobe.com/de/products/flashcatalyst.html
[37] [livecycle]
[38] http://www.adobe.com/de/products/air.html
[39] http://www.adobe.com/de/products/flashplayer.html
[40] http://www.ecmascript.org/
[41] [Widjaja 2010, S. 24]
[42] [Moock 2007]

4 Standard[43].
ActionScript kann unterschiedlich eingesetzt werden:

- Im `<fx:Script>`-Tag der MXML-Datei in einem **[CDATA[ ]]**-Block. Dieser Block wird beim Parsen des MXML vom Parser ausgelassen[44].
- Innerhalb von MXML-Tag-Attributen. Auf diese Weise können kleinere Anweisungen bestimmten Ereignissen der UI-Objekte zugewiesen werden. Da aber diese Anweisungen wie MXML geparst werden, schließt es den Einsatz mancher Zeichen, z. B. <, aus. Aus diesem Grund werden so meistens den Ereignissen die im **[CDATA[ ]]**-Block vordefinierte Methoden zugewiesen. Beschriebene Vorgehensweise ist dem 3.1 zu entnehmen.
- In einer separaten .as-Datei. Dieser Ansatz ermöglicht eine saubere Trennung der Benutzeroberfläche von deren Logik und dient somit einer besseren Austauschbarkeit der Komponenten.
- ActionScript kann auch verschachtelt in MXML-Tags benutzt werden. Dazu wird wiederum der **[CDATA[ ]]**-Block benutzt. Dies fuhrt aber zu einem weniger übersichtlichen Code und einer unerwünschten Vermischung des Codes mit dem Layout.

Listing 3.1: Einsatz von ActionScript innerhalb einer MXML-Datei

```
<fx:Script>
  <![CDATA[
    protected function toggleBtn_clickHandler(event:MouseEvent):void
    {
      if(toggleBtn.selected){
        deptDg.setStyle("fontSize",14);
        toggleBtn.label="Smaller Text";
      }
      else{
        deptDg.setStyle("fontSize",12);
        toggleBtn.label="Bigger Text";
      }
    }
  ]]>
</fx:Script>

<s:ToggleButton id="toggleBtn" x="591" y="113" label="Bigger Text"
            click="toggleBtn_clickHandler(event)"/>
```

[43] http://www.mikechambers.com/blog/2008/08/14/actionscript-3-and-ecmascript-4/
[44] http://www.w3schools.com/xml/xml_cdata.asp

3.2 Entwicklungsumgebung

Für die Entwicklung einer Flex-Anwendung wird das Flex-SDK benötigt, das kostenlos zur Verfügung steht und die Klassenbibliothek (Flex-Framework) sowie den Flex-Kompilator beinhaltet. Eine effiziente Entwicklung benötigt die Flash-Builder-Entwicklungsumgebung, die ein kommerzielles Produkt darstellt. Flash-Builder basiert auf dem Eclipse und bietet alle Vorteile einer modernen Entwicklungsumgebung. Im Rahmen dieser Arbeit wurden Flex-SDK 4.6 und Flash-Builder 4.6 eingesetzt.

3.3 Benutzeroberfläche

Da Flex aus der grafikorientierten Flash-Plattform entstanden ist, ist die Gestaltung der Benutzeroberfläche eine der Stärken deren Frameworks. Flex besitzt ein breites Spektrum von Instrumenten zur Gestaltung der Benutzeroberfläche. Dazu zählen über 70 visuelle UI-Komponenten 3.3.1, ein entwickeltes System zur Behandlung von Ereignissen 3.3.5, Darstellung von Daten auf listenbasierte Weise 3.3.6 und Techniken zur visuellen Anpassung der Komponenten.
Die Tags innerhalb der MXML-Dateien entsprechen den Klassen der Klassenbibliothek oder deren Eigenschaften. Die Flex-Klassenbibliothek an sich ist im ActionScript geschrieben[45]. MXML-Dateien werden von Flex geparst und in SWF-Dateien kompiliert, wobei diese den MXML-Tags entsprechenden ActionScript-Objekte enthalten[46].

3.3.1 UI-Elemente

Flex bietet zahlreiche visuelle Elemente zur Erstellung der Benutzeroberfläche. Die mit der SDK gelieferten UI-Elemente können direkt eingesetzt werden oder auch mit zusätzlichen Eigenschaften erweitert werden. UI-Komponenten werden durch folgende Eigenschaften charakterisiert:

- Größe der Komponente
- Ereignisse, auf die eine Reaktion des Elements definiert werden kann, z. B. onClick für ein Button-Element
- Stil zu Beschreibung von z. B. Schrifttyp und Große

[45] [Widjaja 2010, S. 20]
[46] [flexsdk]

- Effekte(visuelle oder audio), die bei den bestimmten Ereignissen ausgelöst werden
- Skins - Klassen, die das Aussehen der Komponente definieren

Des Weiteren sind visuelle Komponenten in Flex in 4 Gruppen unterteilt und decken viele benötigten Elemente zur Erstellung von reichhaltigen Anwendungen ab:

- **Controls**
  Interaktive Steuerelemente, wie Button, CheckBox und DataPicker
- **Layout-Container**
  Elemente zur Anordnung von anderen Elementen
- **Navigators**
  Visuelle Trennung der Daten nach bestimmten Kriterien, wie TabNavigator
- **Charts**
  Visuelle Darstellung der Daten in Form von Diagrammen und Graphiken

3.3.2 UI-Elementenpakete

Visuelle Objekte in Flex werden in zwei verschiedenen Paketen zu Verfügung gestellt: **Spark** und **mx**-Package. Spark-Paket wurde erst in Version 4 des Flex-SDK eingeführt. Die Beiden Pakete bieten teilweise gleiche Komponenten die sich im Aufbau unterscheiden [47]. Spark-Komponente wird von 2 Klassen anstatt von einer beschrieben. Skin-Klasse definiert das Aussehen und die verbleibende die Logik der Komponente. Der Vorteil daran ist, dass außer den standardmäßigen Einsatz von CSS, kann die Gestaltung der Komponenten über den Austausch der Skin-Klasse erfolgen ohne die dahinterstehende Logik dabei in Betracht zu ziehen. Speziell für Layout-Elemente gibt es noch weitere Unterschiede 3.3.3. Adobe empfiehlt[48] den Einsatz von Spark-Komponenten anstatt diesen des mx-Packages. Da manche Elemente nur jeweils in einem der Pakete vorhanden sind, werden derzeit beide in Flex Anwendungen eingesetzt. Denen werden innerhalb der MXML-Datei Namensräume zugewiesen 3.2.

Listing 3.2: Definition der Namensräume in MXML

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
3               xmlns:s="library://ns.adobe.com/flex/spark"
4               xmlns:mx="library://ns.adobe.com/flex/mx">
```

[47] [Widjaja 2010, S. 184]
[48] [flexsdk]

```
5 ...
6 </s:Application>
```

Werkzeuge wie Flash Catalyst ermöglichen eine Erstellung von Komponenten-Skins aus Designeransicht und stellen einen Verbindungsglied zwischen Programmierern und Designern dar.

3.3.3 Layout

Elemente der Benutzeroberfläche werden in sogenannten Containern ausgelegt, diese können wiederum andere Container sein oder andere UI-Elemente, die als **Kindelemente** bezeichnet werden. Es wird zwischen Layout-Containern, die die Position der Kindelemente definieren, und Navigator-Containern, die Navigationalgorithmen zwischen mehrere Kind-Containern beinhalten.

Paketunterschiede Es gibt weitere Unterschiede zwischen Layout-Elementen der zwei früher angesprochenen Pakete 3.3.2:

- **Modularität**
  Layout Algorithmen für Elemente aus dem mx-Package können nicht verändert werden im Unterschied zu diesen des Spark-Pakets. Aufgrund dieser Tatsache bietet mx-Paket mehrere strikt spezifizierten Layout-Elemente. Benutzerdefinierte Layoutregeln für Spark-Komponenten können in einer separaten Klasse definiert werden. Im Falle des mx-Pakets muss eine Unterklasse der Elementenklasse erstellt werden, wobei die Regeln überschrieben werden.
- **Kindelemente**
  Layout Komponenten des mx-Pakets können nur Objekte der Unterklassen von UIComponent-Klasse umschließen, diese sind nur standardmäßige Objekte der Benutzeroberfläche. Spark-Komponenten erlauben auch graphische Objekte (Unterklassen von GraphicElement-Klasse) als Kindelemente.

Optimierung des Ladevorgangs Da Benutzeroberfläche von Flex-Anwendungen aus graphischen Objekten besteht und das gleichzeitige Laden alle Layout-Elemente mit darin ausgelegten Kindelementen den Performance verschlechtern kann, werden die Kindelemente standardmäßig nur dann erzeugt, wenn diese benötigt werden. Die Erzeugungsregeln können mittels der ContainerCreationPolicy-Eigenschaft des Layout-Elementes definiert werden.

3.3.4 Zustandbasierte Oberfläche via States

Flex-Anwendungen sind zustands- und nicht seitenbasiert. Verschiedene Zustände der Anwendung werden als Objekte der State-Klasse definiert. Ein Übergang von einem Zustand (**State**) zu einem anderen findet meistens ereignisbasiert statt. Verschiedene States werden innerhalb des <s:states>-Tags definiert 4.5. Jedes Element der Benutzeroberfläche kann innerhalb eines oder mehreren States existieren, was über das Attribute includeIn bestimmt werden kann (s. Zeile 16 in 4.5 für das DataGrid). Diese Vorgehensweise erlaubt die Wiederverwendung der bestimmten Elementen innerhalb der unterschiedlichen States. Die Möglichkeit zur Wiederverwendung der kompletten Ansichten wird jedoch verringert.

3.3.5 Ereignisbehandlung

Ereignis-Modell des Flex-Frameworks ist von besonderer Bedeutung, da die Flex-Anwendung zustands- und nicht seitenbasiert ist. Es wird zwischen den System- und Benutzer-Ereignissen unterschieden. Die letzteren tauchen bei den Benutzer-Interaktionen auf, wie Knopfdruck. System-Ereignisse werden beim Erreichen eines bestimmten Zustands ausgelöst, wie z. B. das Eintreffen einer Antwort auf eine Dienstanfrage.
Nach dem Auslösen eines Ereignisses, was über `dispatchEvent()`-Methode initiiert wird, überprüft Flash-Player alle Komponenten der Benutzeroberfläche auf Ereignisbehandlungsroutinen für dieses Ereignis. Dies geschieht in drei Phasen:

- **Capturing-Phase**
  Die Oberflächenhierarchie wird von oben (Application-Element) bis zur Ebene, die den Auslöser enthält, durchgelaufen. Ereignisbehandlungsroutinen sind für diese Phase standardmäßig deaktiviert, können aber über den `useCapture`-Attribut der `addListener()`-Methode aktiviert werden.

- **Targeting-Phase**
  In dieser Phase wird nur nach für den auslösenden Objekt definierten Ereignisbehandlungsroutinen gesucht. Falls definiert, wird der Objekte entsprechend manipuliert und der Behandlungsmethode übergeben.

- **Bubbling-Phase**
  Diese Phase ist der Capturing-Phase ähnlich. Die Suche hier verläuft in der umgekehrten Richtung.

Die Ereignisse können im MXML über die entsprechenden Attribute sowie auch im ActionScript über `addListener()`-Methode abgefangen werden. Die Anpassung der

Behandlung von Ereignissen kann jedoch nur via ActionScript erfolgen und ermöglicht das Setzten der Prioritäten, Aktivierung der Routinen für Capturing-Phase usw. Das Durchlaufen der Phasen beim Auslösen eines Ereignisses kann auch über `stopPropagation()`-Methode angehalten werden. Dieser Ansatz macht die Behandlung von Ereignissen auf alle Ebenen mit einem geringen Aufwand möglich.

3.3.6 Listenbasierte Datendarstellung

Zur Darstellung von Daten in Listen- oder Tabellenform bietet Flex verschiedene Komponenten an, die vom Verhalten her ähnlich aufgebaut sind.
Listenbasierte Elemente der Benutzeroberfläche werden an eine Datenquelle gebunden und bei deren Veränderung automatisch aktualisiert. Als Datenquelle können Objekte von Klassen benutzt werden, die `mx.collections.IList`-Schnittstelle implementieren. Die Elemente werden an die Datenquellen über die `dataProvider`-Eigenschaft gebunden.
In 4.5 wurde die `dataProvider`-Eigenschaft des DataGrids auf eine Instanz der AsyncListView-Klasse gesetzt, die an das Antwort-Objekt einer Dienst-Anfrage gebunden ist. Dabei wird DataGrid bei Veränderung des Antwort-Objektes automatisch aktualisiert.

Listing 3.3: Beispiel DataGrid

```
1  <s:states>
2      <s:State name="Employees"/>
3      <s:State name="EmployeeDetails"/>
4      <s:State name="EmployeeAdd"/>
5      <s:State name="EmployeeUpdate"/>
6  </s:states>
7  <fx:Declarations>
8    <s:CallResponder id="getEmployeesResult"
9                result="employee = getEmployeesResult.lastResult[0] as Employee;employee =
                         getEmployeesResult.lastResult[0] as Employee"/>
10   <employeeservice:EmployeeService id="employeeService"
11               fault="Alert.show(event.fault.faultString + '\n' + event.fault.faultDetail)"
12               showBusyCursor="true"/>
13   <valueObjects:Employee id="employee"/>
14 </fx:Declarations>
15 <fx:Binding source="empDg.selectedItem as Employee" destination="employee"/>
16 <s:DataGrid id="empDg" includeIn="EmployeeAdd,EmployeeDetails,EmployeeUpdate,
       Employees" x="60" y="136" width="650"
17         creationComplete="empDg_creationCompleteHandler(event)" requestedRowCount="4"
18         selectionChange="empDg_selectionChangeHandler(event)"
19         editable="true" gridItemEditorSessionSave="empDg_gridItemEditorSessionSaveHandler(
               event)">
```

```
20    <s:columns>
21      <s:ArrayList>
22        <s:GridColumn dataField="lastname" headerText="Last Name" width="110"></
              s:GridColumn>
23        <s:GridColumn dataField="firstname" headerText="First Name" width="110"></
              s:GridColumn>
24        <s:GridColumn dataField="title" headerText="Title" width="170"></s:GridColumn>
25        <s:GridColumn dataField="cellphone" headerText="Cellphone" width="110"></
              s:GridColumn>
26        <s:GridColumn dataField="email" headerText="Email"></s:GridColumn>
27      </s:ArrayList>
28    </s:columns>
29    <s:typicalItem>
30      <fx:Object id="id1" cellphone="cellphone1" city="city1" departmentid="departmentid1"
31            email="email1" firstname="firstname1" lastname="lastname1" office="office1"
32            officephone="officephone1" photofile="photofile1" state="state1"
33            street="street1" title="title1" zipcode="zipcode1"></fx:Object>
34    </s:typicalItem>
35    <s:AsyncListView list="{getEmployeesResult.lastResult}"/>
36  </s:DataGrid>
```

Das Verhalten des DataGrid sowie dessen Spalten ist über Attribute, wie `editable`, `resizable` usw. anpassbar. Wenn eine besondere oder zusammengefasste Darstellung oder das Editieren verschiedener Daten in einer Spalte notwendig ist, stellt Flex ItemRenderer bzw. ItemEditor-Klassen zur Verfügung.
ItemRenderer bzw. ItemEditor kann z. B. innerhalb eines <fx:Component>-Tags definiert werden, das wiederum Inhalt eines DataGridColumn-Tags ist. Über `rendererIsEditor`-Eigenschaft kann der gleiche ItemRenderer auch für das Editieren der Spaltendaten benutzt werden.

3.4 Behandlung der klientseitigen Daten

Flex verfügt über verschiedene Instrumente um lokale Daten zu validieren, im vordefinierten Format darzustellen sowie zwischen den Objekten zu synchronisieren.

3.4.1 Data Binding

Data Binding ermöglicht, ähnlich wie im Silverlight 4.3.1.5, die Übetragung der Eigenschaften eines Objektes auf das andere. Falls das Quellobjekt verändert wird, wird der Zielobjekt benachrichtigt und entsprechenden synchronisiert. Data Binding kann auf drei verschiedenen Weisen erfolgen:

- Inline - direkte Bindung an eine Eigenschaft eines Objekts innerhalb dessen Definition in MXML
- In einem separaten <fx:Binding>-Tag
- In ActionScript innerhalb des script-Abschnitts der MXML-Datei oder in einer separater .as-Datei

3.4.2 Data Model

Um die Daten strukturiert zuzugreifen und nach dem MVC-Entwurfsmuster zu implementieren verfügt das Flex-Framework über die Modell-Klasse (Data Model). Ein Data Model ermöglicht die logisch zusammenhängenden Daten in Form von Objekten zu behandeln und diese für verschiedene Teile der Anwendung verfügbar zu machen. Das Modell kann in MXML oder Script-basiert definiert werden.

3.4.3 Klientseitige Datenvalidierung und Formatierung

Für die Validierung der Benutzereingaben oder Ausgabe von Daten in formatierter Weise bietet das Flex-Framework eine Anzahl von vordefinierten Validator- und Formatter-Klassen, z. B. für Telefonnummern und Zeitstempeln, sowie Basisklassen zur Implementierung der benutzerdefinierten Ansätze.
Der Zusammenhang und typischer Einsatz erwähnter Werkzeuge ist anhand eines Beispiels in 3.1 vorgestellt.

3.5 Kommunikation mit externen Datenquellen

Das Flex-Framework stellt verschiedene Klassen zur Erstellung von Verbindungen mit unterschiedlichen Datenquellen zur Verfügung, wie HTTPService 3.5.1, WebService 3.5.3 und RemoteObject. Dabei können sämtliche Backendtechnologien zum Einsatz kommen[49], die z. B. Plain Text, XML-Daten oder SOAP-Datenströme liefern[50]. Der gleichzeitige Einsatz des Data Centric Development von Flex 3.5.4 ermöglicht eine schnelle Implementierung der Kommunikation und Präsentation der Backenddaten.

[49] Java wird bevorzugt bei der Entwicklung mit Flex
[50] [Widjaja 2010, S. 359]

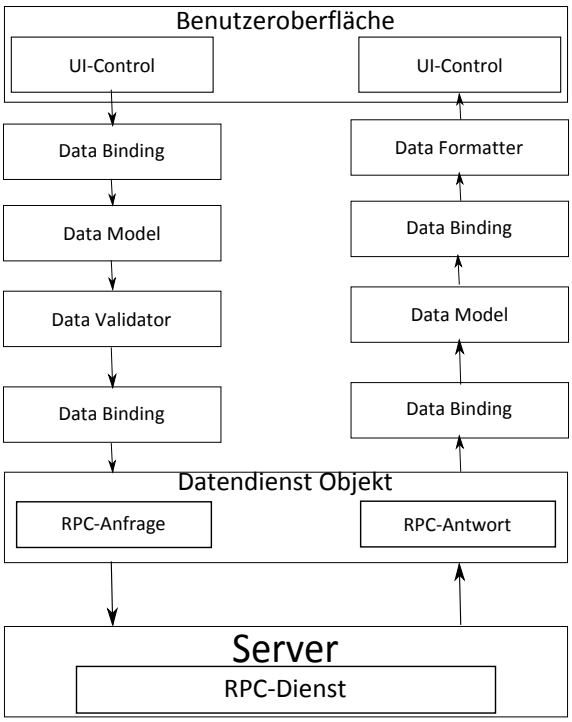

Abbildung 3.1: Datenfluss der Flex-Anwendung
Widjaja [2010]

3.5.1 HTTPService

HTTPService kann zur Übertragung von Text, XML-Dateien und URL-kodierten Variablenpärchen benutzt werden[51]. Bei dem Einsatz von HTTPService können GET und POST-Anfragen gesendet werden. Kommunikation erfolgt asynchron und wird über Ereignisse gesteuert.
Über das Attribut contentType="application/xml" wird dem Dienst mitgeteilt, dass Daten im XML-Format zurückgegeben werden. Dabei wird die Struktur der Datei automatisch analysiert und ein direkter Zugriff auf die einzelne Knoten möglich gemacht 3.4.

Listing 3.4: Einlesen einer externer XML-Datei über einen HTTPService

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
3                xmlns:s="library://ns.adobe.com/flex/spark"
4                xmlns:mx="library://ns.adobe.com/flex/mx">
```

[51] [Widjaja 2010, S. 361]

```
5   <fx:Script>
6     <![CDATA[
7       import mx.controls.Alert;
8       import mx.events.FlexEvent;
9       import mx.rpc.events.FaultEvent;
10
11      protected function dataGrid_creationComplete(event:FlexEvent):void {
12        httpService.send();
13      }
14      protected function httpService_fault(event:FaultEvent):void {
15        trace( "httpService_fault" );
16        var msg:String = event.fault.faultString
17          +"\n\n"+event.fault.faultDetail;
18        Alert.show(msg, "Fehler im HTTPService");
19      }
20    ]]>
21  </fx:Script>
22
23  <fx:Declarations>
24    <s:HTTPService id="httpService"
25                   url="daten/employee.xml"
26                   contentType="application/xml"
27                   fault="httpService_fault(event)"/>
28  </fx:Declarations>
29
30  <mx:DataGrid dataProvider="{httpService.lastResult.root.employee}"
31               creationComplete="dataGrid_creationComplete(event)"/>
32
33 </s:Application>
```

Listing 3.5: XML mit Angestelltendaten

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <root>
3    <employee>
4      <name>Jack</name>
5      <age>27</price>
6    </employee>
7    <employee>
8      <name>Sarah</name>
9      <age>45</price>
10   </employee>
11   <employee>
12     <name>John</name>
13     <age>34</price>
14   </employee>
15 </root>
```

3.5.2 WebService

WebService wird beim Einsatz von Diensten benutzt, die Daten in SOAP-Format übertragen. Generell ist dieser Einsatz dem des HTTPService ähnlich, verfügt aber über mehrere Einstellungen, wie makeObjectsBindable um Objekte für Data Binding 3.4.1 verfügbar zu machen. Genauso wie bei dem HTTPService sind Daten bei einer erfolgreichen Antwort auf eine Serveranfrage über den result-Attribut des ResultEvent zugänglich.

3.5.3 RemoteObject

RemoteObject wird für den direkten Zugriff auf die Methoden des entsprechenden Dienstes benutzt, die über verschiedene Frameworke, wie **BlazeDS**[52], **LiveCycle Data Services ES**[53] usw. , verfügbar gemacht werden. RemoteObject erlaubt das Ausführen der Methoden auf den Objekten der Java-Klassen[54] oder **ColdFusion**[55]. Third-party Komponenten erlauben den Zugriff auf PHP- und .Net-Komponenten[56]. 3.6 stellt den Einsatz eines RemoteObject auf der Klientseite zur Verbindung mit einem Dienst dar. Der Dienst wird von einem BlazeDS-Backend zur Verfügung gestellt. Dabei sind zwei Methoden auf den Wertobjekten des Typs Employee für der RemoteObject deklariert.

Listing 3.6: Einsatz von RemoteObject

```
1  internal class EmployeeService extends com.adobe.fiber.services.wrapper.
       RemoteObjectServiceWrapper
2  {
3      // Constructor
4      public function EmployeeService()
5      {
6
7    // Inizialisierung des Dienstcontrols
8  _serviceControl = new mx.rpc.remoting.RemoteObject();
9
10 //RemoteClass alias für alle von den Methoden des Dienstes zurückgegebenen Entitäten
11 valueObjects.Employee._initRemoteClassAlias();
12 valueObjects.Department._initRemoteClassAlias();
13
14 var operations:Object = new Object();
15 var operation:mx.rpc.remoting.Operation;
16
17 operation = new mx.rpc.remoting.Operation(null, "getEmployeesByName");
```

[52] [blazeds]
[53] [livecycle]
[54] [flexsdk]
[55] http://www.adobe.com/products/coldfusion-family.html
[56] [flexdata]

```
operation.resultElementType = valueObjects.Employee;
operations["getEmployeesByName"] = operation;
operation = new mx.rpc.remoting.Operation(null, "updateEmployee");
operations["updateEmployee"] = operation;

_serviceControl.operations = operations;
.
.
.
public function getEmployeesByName(arg0:String) : mx.rpc.AsyncToken
 {
    var _internal_operation:mx.rpc.AbstractOperation = _serviceControl.getOperation("
        getEmployeesByName");
  var _internal_token:mx.rpc.AsyncToken = _internal_operation.send(arg0) ;
    return _internal_token;
}
public function updateEmployee(arg0:valueObjects.Employee) : mx.rpc.AsyncToken
{
    var _internal_operation:mx.rpc.AbstractOperation = _serviceControl.getOperation("
        updateEmployee");
  var _internal_token:mx.rpc.AsyncToken = _internal_operation.send(arg0) ;
    return _internal_token;
}
```

Listing 3.7: Behandlung der Antwort auf eine Dienstanfrage in MXML

```
<fx:Script>
  <![CDATA[
    protected function empDg_creationCompleteHandler(event:FlexEvent):void
      {
        getEmployeesResult.token = employeeService.getEmployees();
      }
  ]]>
</fx:Script>

<fx:Declarations>
    <s:CallResponder id="getEmployeesResult"
            result="employee = getEmployeesResult.lastResult[0] as Employee;employee =
                getEmployeesResult.lastResult[0] as Employee"/>
    <employeeservice:EmployeeService id="employeeService"
                    fault="Alert.show(event.fault.faultString + '\n' + event.fault.faultDetail
                        )"
                    showBusyCursor="true"/>
    <s:CallResponder id="getDepartmentsResult"/>
    <valueObjects:Employee id="employee"/>
    <s:CallResponder id="updateEmployeeResult" result="updateEmployeeResult_resultHandler
        (event)"/>
</fx:Declarations>
```

Die Methoden schließen die Dienstanfragen um und geben ein AsyncToken-Objekt zurück. Die result-Eigenschaft des Objekts enthält die eigentliche Antwort auf die Dienstanfrage. Um auf das Resultat aus dem MXML zuzugreifen wird ein CallResponder definiert 3.7. Seiner token-Eigenschaft wird der zurückgegebene AsyncToken zugewiesen. Weitere Details über den praktischen Einsatz mit verschiedenen Backend-Technologie können in [flexsdk] gefunden werden.

3.5.3.1 AMF

Während HTTPService und WebService das HTTP-Protokoll benutzen, werden Daten beim Einsatz von RemoteObject als binärer Datenstrom im **Action-Message-Format** (AMF)[57] übertragen. AMF wurde von Adobe als offenes Protokoll für das Serialisieren bzw. Deserialisieren von komplexen Objekten zwischen Klient und Serverseite entwickelt. AMF bietet einige Vorteile gegenüber den textbasierten Übetragungsmethoden[58]:

- **Performantes Serialisieren und Deserialisieren**
  C-Sprache ist nativ für die Plattform, wo Flash Player ausgeführt wird. AMF-Implementierungen für z. B. BlazeDS werden in C-Sprache definiert. Beim Erstellen der Objekte wird somit kein Parsing benötigt.
- **Effiziente Komprimierung**
  Datentypen werden erkannt und Werte vom gleichen Typ werden zusammengefügt übergeben. Dadurch entfällt der Overhead für die Datentypenbezeichnung jedes Wertes.
- **Effiziente und robuste Verbindung zwischen Klient und Server**
  In Flex unterstützt die AMF-Implementierung eine automatische Stapelung von Anfragen und Failover-Regeln[59].
- **Effiziente Ausnutzung der Bandbreite**
  Laut Statistiken[60] benutzt AMF bis zur Hälfte der Bandbreite, die beim Einsatz von anderen Protokollen aufgebraucht wird, und ist 3 bis 10 mal performanter abhängig vom Umfang der zu übertragenden Daten.

[57] [flexsdk]
[58] [Fain u. a. 2010, S. 266]
[59] http://de.wikipedia.org/wiki/Failover
[60] [census]

3.5.4 Data Centric Development

Data Centric Development ist ein Paket von Werkzeugen zum vereinfachten Umgang mit externen Daten und Diensten. Dabei steht eine große Auswahl an unterstützten Backends wie WebServices, Java mit BlazeDS[61], PHP mit ZendFramework[62] usw. zur Verfügung. Die wichtigsten Bestandteile des Data Centric Development sind[63]:

- Wizards zum Import der Dienstimplementierungen
- Testen der einzelnen Dienstmethoden
- Überblick über Methoden und Objekte der importierten Dienste direkt in der Entwicklungsumgebung
- Feststellung von Argumenten und Rückgabetypen beim Import und falls nötig deren Manipulation
- Generierung des Codes der Daten Transfer Objekte und Service-Wrapper
- Bindung der Dienstaufrufe an UI-Elemente per Drag and Drop
- Automatische Daten Verwaltung mit persistenten Objekten auf der Klientseite

3.6 Validierung

Neben den Möglichkeiten die Daten klientseitig zu validieren, können je nach eingesetzter Backendtechnologie entsprechende serverseitige Validierung-Frameworke eingesetzt werden, z. B. **Hibernate Validator**[64] für einen Java-Backend.

3.7 Lokalisierung

Flex unterstützt die Lokalisierung von Anwendungen über entsprechende Klassen des `spark`-Packages, wie CurrencyFormatter, NumberFormatter usw. Die eingesetzten Formatter-Klassen werden innerhalb der MXML-Datei in einem <fx:declarations>-Tag definiert oder in einem **Resource Bundle**, der alle lokalisierten Teile der Anwendung enthält, wie Texte, Bilder, Formate usw. Resource Bundles werden in locale-Ordner in entsprechenden Unterordnern gespeichert. Eine Referenz auf die eingesetzten Resource Bundles wird in einem <fx:Metadata>-Tag definiert.

[61] [blazeds]
[62] [zend]
[63] [Widjaja 2010, S. 380]
[64] http://www.hibernate.org/subprojects/validator.html

Bindung an eine lokalisierte Ressource erfolgt über `resourceManager`-Eigenschaft des jeweiligen Elementes der Benutzeroberfläche.

4 Silverlight

4.1 Plattform

Microsoft Silverlight ist eine Implementierung des **.NET-Frameworks**[65] zur Erstellung von reichhaltigen Webanwendungen. Damit eine Silverlight-Anwendung auf dem Klient ausgeführt werden kann, benötigt man eine entsprechendes Plugin, das kostenlos zur Verfügung gestellt wird. Die Hauptbestandteile der Silverlight-Plattform sind[66]:

- **Kernpräsentationsframework**
  Das Framework beinhaltet alle Benutzeroberflächen-relevanten Elemente und Dienste, wie Steuerelemente und Benutzeroberflächenrendering. Außerdem ist **Extensible Application Markup Language (XAML)** zur deklarativer Definition des Benutzeroberflächenlayouts verfügbar.
- **.NET Framework für Silverlight**
  Ein Teil der für Webanwendungen relevanten Bibliotheken zur Kommunikation über Netzwerk, Integration von Daten usw. Eine komplette Implementierung des **Common Language Runtime** (CLR)[67] Laufzeitumgebung ist auch enthalten, was eine Entwicklung in unterschiedlichen Sprachen ermöglicht.

Die Zusammenhänge der zur Verfügung stehenden Komponenten sind in 4.1 abgebildet: Silverlight-Anwendungen werden als XAP-Dateien bereitgestellt, die in Assemblies compilierten Code und Ressourcen beinhalten. XAP-Dateien werden von Server heruntergeladen und auf dem Klient vom Silverlight-Plugin ausgeführt.

4.2 Entwicklungsumgebung

Für die Entwicklung von Silverlight-Anwendungen ist Microsoft Visual Studio die optimale Entwicklungsumgebung. Visual Studio enthält alle modernen Werkzeuge, die zur Entwicklung der auf den verschiedenen Microsoft-Produkten basierenden

[65] http://msdn.microsoft.com/de-de/netframework/aa569263
[66] [msdnlib]
[67] http://msdn.microsoft.com/de-de/library/8bs2ecf4.aspx

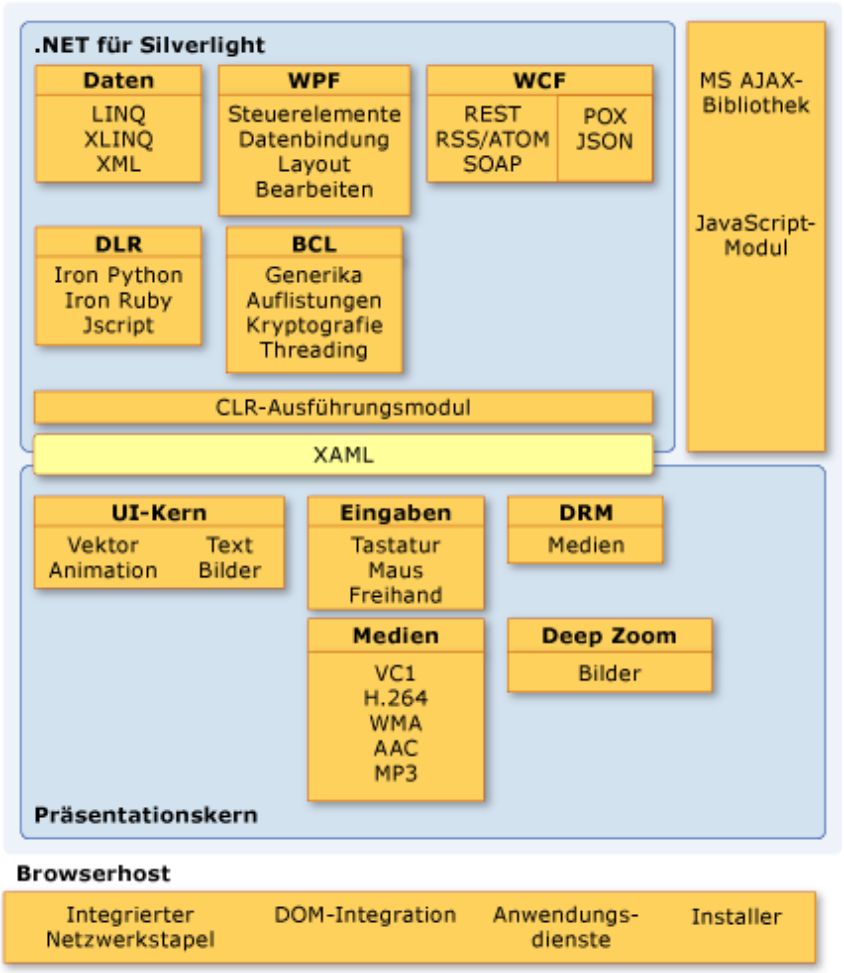

Abbildung 4.1: Architektur der Silverlight-Plattform
msdnlib

Anwendungen benötigt werden. Speziell für die Entwicklung mit Silverlight kann die Web Developer Variante der Entwicklungsumgebung benutzt werden. Darüber hinaus wird eine Installation von **Silverlight Tools for Visual Studio** und **WCF RIA Services Toolkit** 4.4 gebraucht. Im Rahmen dieser Arbeit wurden **Visual Studio Professional 2010**[68], **Silverlight 4 Tools for Visual Studio**[69] und **WCF RIA Services Toolkit (September 2011)**[70] eingesetzt.

[68] http://www.microsoft.com/germany/visualstudio/products/team/visual-studio-professional.aspx
[69] http://www.microsoft.com/downloads/de-de/details.aspx?FamilyID=b3deb194-ca86-4fb6-a716-b67c2604a139
[70] http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=26939

4.3 Benutzeroberfläche

Benutzeroberfläche bei der Entwicklung mit Silverlight wird in **XAML** und Code-Behind-Dateien definiert. Die Objekt-Hierarchie wird dabei in einer XAML-Datei definiert und die Code-Behind-Datei beinhaltet je nach Implementierungscenario Erreignishandler und einen weiteren Code zur Steuerung der Ansicht. Die in XAML definierten sichtbaren Elemente der Benutzeroberfläche werden in Objekte der Vektorgrafik gerendert[71].

4.3.1 XAML

XAML ist eine deklarative auf dem XML basierende Sprache die für .NET 3.5 und .NET 4 in **Windows Presentation Foundation**[72] entwickelt wurde. Die Struktur der Benutzeroberfläche wird mit Hilfe von Tags beschrieben, die UI-Objekte sowie deren Eigenschaften definieren. Jeder Tag des XAML-Dokumentes ist einer .NET-Klasse zugeordnet. Im Unterschied zur Flex werden XAML-Dateien nicht beim Kompilieren des Projekts geparst, sondern zur Laufzeit. XAML-Dateien werden dem XAP-File zusammen mit anderen Ressourcen hinzugefugt.
In weiteren Abschnitten wird XAML-Syntax zur Definition von Objekten sowie deren Eigenschaften vorgestellt.

4.3.1.1 Eigenschaften von UI-Objekten

Eigenschaften der Objekte können je nach Art auf verschiedenen Wegen mit XAML gesetzt werden. Eigenschaften mit primitiven Wertetypen werden über Attribute gesetzt, z. B. Text eines Textblockes <TextBlock Text=ÖK»

4.3.1.2 Inhaltselementsyntax

Manche Elemente definieren eine Eigenschaft, der der Inhalt des Elementes zugewiesen wird. Die Content-Eigenschaft erlaubt somit einen komplexen, aus beliebigen XAML-Elementen bestehenden Inhalt einem UI-Element zuzuweisen. Zum Beispiel kann der Inhalt eines Label-Elements in einem StackPanel-Element definiert werden.

[71] [Czernicki 2009, S. 139]
[72] http://msdn.microsoft.com/de-de/netframework/aa663326

Listing 4.1: Einsatz von Inhaltselementsyntax

```
<Label Width="80" Height="35">
  <StackPanel Orientation="Horizontal">
    <Image Source="smile.png" Width="16" Height="16" />
    <TextBlock Margin="7,0,0,0" Text="Hello World!" />
  </StackPanel>
</Label>
```

4.3.1.3 Eigenschaftenelementsyntax

Um Objekteigenschaften komplexe Werte zuzuweisen erlaubt XAML die Eigenschaften im Form eines Elementes zu erstellen und zu definieren. Die Eigenschaft-Elemente werden mit der folgender Namenskonvention beschrieben <Objekt.Eigenschaft>. Diese Definitionsweise wird oft benutzt um die Ressource-Eigenschaft eines Objekts zu setzen, z. B. für Spalten eines DataGrid-Elementes.

Listing 4.2: Einsatz von Eigenschaftenelementsyntax

```
<sdk:DataGrid AutoGenerateColumns="False" HorizontalAlignment="Left" ItemsSource="{
    Binding ElementName=truckersratesDomainDataSource, Path=Data}" Margin="0,149,0,0"
    Name="truckersratesDataGrid" RowDetailsVisibilityMode="VisibleWhenSelected"
    VerticalAlignment="Top" Width="892" Height="261">
    <sdk:DataGrid.Columns>
    <sdk:DataGridTextColumn x:Name="client_CompaniesIdColumn" Binding="{Binding Path
        =Client_CompaniesId}" Header="Client Companies Id" Width="SizeToHeader" />
        <sdk:DataGridTextColumn x:Name="idRatesColumn" Binding="{Binding Path=idRates
            , Mode=OneWay}" Header="Id Rates" IsReadOnly="True" Width="SizeToHeader"
            />
  </sdk:DataGrid.Columns>
</sdk:DataGrid>
```

4.3.1.4 Angefügte Eigenschaften

Das Konzept ermöglicht es den Objekten bestimmte Eigenschaften als angefügte zu registrieren, damit diese von anderen Objekten gesetzt werden können. Dies erlaubt zum Beispiel die Positionierung eines TextBlock-Elementes in einem Canvas-Layout-Element, wobei TextBlock, wie auch die meisten Steuerelemente, selbst verfügt über keine Eigenschaften zur Bestimmung der Position.

Listing 4.3: Bestimmung der Positionierung über angefügte Eigenschaften

```
<Canvas>
  <TextBox Canvas.Left="30" Canvas.Top="15" />
</Canvas>
```

4.3.1.5 Markuperweiterung

Im Falle, dass Werte der Objekteigenschaften erst zur Laufzeit gesetzt werden können, kommt eine Markuperweiterung zum Einsatz. Was zum Beispiel eine einfache Möglichkeit zur Darstellung von Serverdaten in UI-Objekten über die Binding-Erweiterung bietet. In 4.2 wird ein DataGrid-Element mit der auf der Datenbank basierende Domain Ressource Quelle verbunden, sowie den Spalten des DataGrid bestimmte Attribute des Domain-Objektes zugewiesen. Daten können über den **Data Sources Editor** in Visual Studio schnell im Form von UI-Objekten dargestellt werden. Alle dafür benötigten Bindungen werden automatisch generiert.
Markuperweiterungen können auch als Werte für Attribute anderer Markuperweiterungen sein. Wie im folgendem Codeabschnitt dem Source-Attribut der Binding-Erweiterung die StaticResource-Erweiterung zugewiesen ist.

Listing 4.4: Verschachteln von Markuperweiterungen

```
1 <TextBlock x:Name="ApplicationNameTextBlock" Style="{StaticResource
      ApplicationNameStyle}"
2                                   Text="{Binding ApplicationStrings.ApplicationName, Source={
                                        StaticResource ResourceWrapper}}"/>
```

4.3.2 Layout

Steuerelemenente des Silverlight-Toolkits besitzen, wie früher erwähnt, keine Attribute zur Bestimmung deren Position. Steuerelemente werden innerhalb Layout-Steuerelemente ausgelegt. Es stehen drei Layout-Komponenten zur Verfügung:

- Canvas zur absoluten Positionierung der Elemente.
- StackPanel zur relativen Positionierung der Elemente zu einander. Elemente können horizontal oder vertikal ausgelegt werden.
- Grid-Element ermöglicht die Steuerelemente innerhalb einer Tabelle zu positionieren, wobei Spalten und Reihen des Grid-Elementes sehr flexibel eingestellt werden können auf fixierte oder sich dem Browserfenster anpassende Maße.

Designer View des Visual Studio ist ein moderner **WYSIWYG**-Editor[73]. In Designer View kann das Layout der Oberfläche sowie Eigenschaften der UI-Oberfläche definiert werden. In Verbindung mit dem **Data Sources**-Editor, der für die Anwendung verfügbare Datenquelle anzeigt, können diese Daten-darstellende Elemente direkt in Designer View erstellt werden. Dabei kann auch die eigentliche Darstellung näher definiert werden, z. B. ob ein Datum als Text oder als ein Kalender-Element

[73] http://de.wikipedia.org/wiki/WYSIWYG

mit Auswahlmöglichkeit angezeigt werden soll. Die Erstellung einer komplexen Benutzeroberfläche der Anwendung mit diesen Werkzeugen kann in einer kurzen Zeit erfolgen.

4.3.3 Datenbindung

Silverlight bietet mit XAML und oben erwähnter Binding-Markuperweiterung eine Möglichkeit Steuerelemente an die anzuzeigenden Daten zu binden. Jede Eigenschaft (muss eine Abhängigkeitseigenschaft[74] sein) des Framework-Elementes kann mit einem Objekt verbunden werden. Der Bindungsmodul des Silverlight ruft folgende Eigenschaften des Binding-Objekts auf[75]:

- Zieleigenschaft für die Darstellung und Veränderung der Daten
- Quelle der Daten, die ein beliebiges CLR-Obejkt[76] sein kann, z. B. DomainDataSource-Objekt 4.4.1.4 als Datenquelle verwendet
- Art der Synchronisierung zwischen dem Quell- und Ziel-Objekt. Der Datenfluss kann für eine bzw. beide Richtungen oder gar als einmalig definiert werden. Dies erfolgt über die Mode-Eigenschaft, die folgenden Werte einnehmen kann: OneWay, OneTime oder TwoWay.
- Optional können Daten mittels eines Wertekonverters manipuliert werden

Wenn Darstellung der Daten aus einer Quelle in mehreren Elementen geschieht, z. B. verschieden Spalten des Datagrid-Elementes, kann die DataContext-Eigenschaft (im Falle des DataGrid der ItemSource-Eigenschaft) des übergeordneten Elementes die Quelle zugewiesen werden. Damit benutzen alle untergeordneten Elemente die gleiche Quelle und mit der Eigenschaft Path können Zieleigenschaften mit einer bestimmten Eigenschaft des Quellobjekts verbunden werden (siehe 4.2).

4.3.4 Listenbasierte Datendarstellung

Für die Darstellung der Daten in Form von Listen oder Tabellen können in Silverlight DataGrid oder ListBox Elemente benutzt werden. Die Quelle der Daten kann eine Auflistung sein, die IEnumerable-Schnittstelle implementiert. Ein listenbasiertes Element wird über die Eigenschaft an die Auflistung gebunden, z. B. ItemSource des DataGrid-Elementes. Diese Bindung kann in XAML oder in dazugehöriger Code-Behind-Datei

[74] [msdnlib]
[75] [Beres u. a. 2010]
[76] http://msdn.microsoft.com/de-de/library/8bs2ecf4.aspx

erfolgen. In später aufgeführten XAML-Code ist DataGrid an ein DomainDataSource-Element gebunden, das eine Auflistung vom Domänendienst verfügbar gemachten Entitäten darstellt.

Listing 4.5: Datendarstellung mit DataGrid

```
1  <sdk:DataGrid AutoGenerateColumns="False" HorizontalAlignment="Left" ItemsSource="{
       Binding ElementName=truckersratesDomainDataSource, Path=Data}" Margin="0,149,0,0"
       Name="truckersratesDataGrid" RowDetailsVisibilityMode="VisibleWhenSelected"
       VerticalAlignment="Top" Width="892" Height="261">
2      <sdk:DataGrid.Columns>
3          <sdk:DataGridTextColumn x:Name="client_CompaniesIdColumn" Binding="{Binding
               Path=Client_CompaniesId}" Header="Client Companies Id" Width="SizeToHeader
               " />
4          <sdk:DataGridTextColumn x:Name="idRatesColumn" Binding="{Binding Path=idRates
               , Mode=OneWay}" Header="Id Rates" IsReadOnly="True" Width="SizeToHeader"
               />
5          <sdk:DataGridTemplateColumn x:Name="validFromColumn" Header="Valid From"
               Width="SizeToHeader">
6              <sdk:DataGridTemplateColumn.CellEditingTemplate>
7                  <DataTemplate>
8                      <sdk:DatePicker SelectedDate="{Binding Path=ValidFrom, Mode=TwoWay
                           , NotifyOnValidationError=true, ValidatesOnExceptions=true,
                           TargetNullValue=''}" />
9                  </DataTemplate>
10             </sdk:DataGridTemplateColumn.CellEditingTemplate>
11             <sdk:DataGridTemplateColumn.CellTemplate>
12                 <DataTemplate>
13                     <TextBlock Text="{Binding Path=ValidFrom, StringFormat=\{0:d\}}" />
14                 </DataTemplate>
15             </sdk:DataGridTemplateColumn.CellTemplate>
16         </sdk:DataGridTemplateColumn>
17     </sdk:DataGrid.Columns>
18 </sdk:DataGrid>
```

Im Falle eines DataGrid, das eine tabellarische Darstellung von Daten ermöglicht, werden Regeln für das Rendern der Spalteninhalte auf der Basis von Vorlagen definiert. Dafür stehen drei Vorlagen zur Verfügung:

- DataGridTextColumn rendert die darzustellenden Daten in Text
- DataGridCheckBoxColumn für die Darstellung von `boolean` in Form eines CheckBox-Elementes
- DataGridTemplateColumn, für die können Zellenvorlagen(CellTemplate) definiert werden, die über vordefinierte Dateivorlagen eine Darstellung der Daten in einer beliebigen Form erlauben.

Der Einsatz von verschiedenen Spalten ist 4.5 zu entnehmen. Dabei wurden Spalteninhalte an Entitäteneigenschaften über die Path-Eigenschaft der Spalten gebunden.

4.3.4.1 Auflistungenverwaltung

Silverlight unterstützt verschiedene generischen Auflistungstypen, wie List oder Dictionary. Silverlight und WPF unterstützen auch einen besonderen Typ von Auflistungen ObservableCollection.

ObservableCollection<T> Das ist ein ebenso generischer Typ von Auflistungen, der die INotifyPropertyChanged-Schnittstelle implementiert. Diese Schnittstelle erlaubt das Auslösen eines CollectionChanged, wenn die Auflistung manipuliert wird. UI-Objekte, die an so eine Auflistung gebunden sind, stellen die neuen Daten automatisch dar. Der Vorteil daran ist, dass die Auflistungen im Code manipuliert werden können ohne jeder Kenntnis von UI-Objekten.

Views Ein weiteres Konzept von Silverlight sind sogenannte Views. View einer Auflistung erlaubt das Manipulieren (sortieren, filtern usw.) dieser ohne diese zu verändern, z. B. das Anklicken des Headers einer Spalte einer DataGrid-Tabelle sortiert die dargestellten Daten ohne die darunterliegende Auflistung zu manipulieren. Ein View, der die Auflistung umschließt, wird automatisch beim Binden dieser an ein darstellendes Objekt erstellt. Die Manipulation der Views kann entweder automatisch erfolgen oder vordefiniert werden[77]. Dafür stehen verschiedene Viewtypen zur Verfügung, die Einsatzszenarien dieser sind in [78] näher erläutert.

4.3.5 Dateneingabeformen

Formen für Erstellung neuer Datenobjekten oder Editieren bereits vorhandener werden im Silverlight mit Hilfe eines DataForm-Steuerelementes erstellt. Die endgültige Gestaltung des DataForm-Elementes kann über Attribute, wie AutoEdit und AutoCommit definiert werden. Dabei werden die logisch benötigten Elemente, z. B. Knopf-Elemente zum Abschicken der Veränderungen, automatisch hinzugefügt.
DataForm-Element verfügt, genauso wie DataGrid-Element, über ein eingebautes Verhalten der Felder bei Validierungfehlern, rote Schattierung usw. , sowie einen ValidationSummary-Steuerelement, das die Liste der vorgekommenen Fehlern unter dem

[77] [Czernicki 2010, S. 97]
[78] [Beres u. a. 2010, S. 144]

DataForm-Element anzeigt. Dies erspart die Implementierung dieser, für Business-anwendungen relevanten Verhaltens.

4.3.6 Navigation-Framework

Silverlight stellt ein Framework zur Navigation durch die verschiedene Ansichten der Anwendung, die Übergänge von einer Ansicht zur anderen werden dabei mit der Browsergeschichte synchronisiert. Die Hauptkomponenten des Frameworks sind Frame-Steuerelement und die Page-Klasse. Die Hauptseite der Anwendung beinhaltet das Frame-Element als Container für andere Ansichten. Ansichten werden über URI[79] navigiert, die beim Laden der Ansicht dem URL der Hostseite hinzugefügt werden. Navigation-Framework unterstützt auch die Übergabe von Anfragenattributen, was verbunden mit einstellbaren Caching der Ansichten die Implementierung einer vom Zustand abhängigen Navigation vereinfacht.

4.4 Kommunikation mit externen Daten

4.4.1 WCF RIA Services

Synchronization der Anwendungslogik auf Klient- und Serverseite in Silverlight Anwendungen wird mit Hilfe von WCF RIA Services (weiter RIA Services) gehandhabt. RIA Services ist eine Technologie, die auf Windows Communication Foundation[80] aufbaut und ermöglicht die Entwicklung der Anwendungslogik komplett auf die mittlere Ebene zu verschieben.
RIA Services stellt ein Framework für die Datenverwaltung, Autorisierung, Authentifizierung und den Codegenerator zu Verfügung. Der Code der Klientseite für den Zugriff auf Dienste und Daten wird von RIA Services generiert. Somit entfällt der Aufwand zur Koordination der Anwendungslogik zwischen der Präsentations- und mittleren Ebene. Im folgendem Bild 4.2 wird schematisch eine mehrschichtige Anwendung dargestellt, wobei RIA Services sich mit Bereitstellung der auf der Datenzugriffsebene definierten Anwendungslogik für die Präsentationsebene beschäftigt.
Um auf die Daten zuzugreifen bietet sich der Einsatz von **ADO.NET Entity Framework** 4.4.1.1 an, der mit einem geringem Aufwand mit RIA Services in das Projekt integriert werden kann. Es sei anzumerken, dass RIA Services nicht an eine bestimmte Datenzugriff Technologie gebunden ist[81].

[79] http://de.wikipedia.org/wiki/Uniform_Resource_Identifier
[80] http://msdn.microsoft.com/de-de/netframework/aa663324
[81] [Cleeren und Dockx 2010, S. 375]

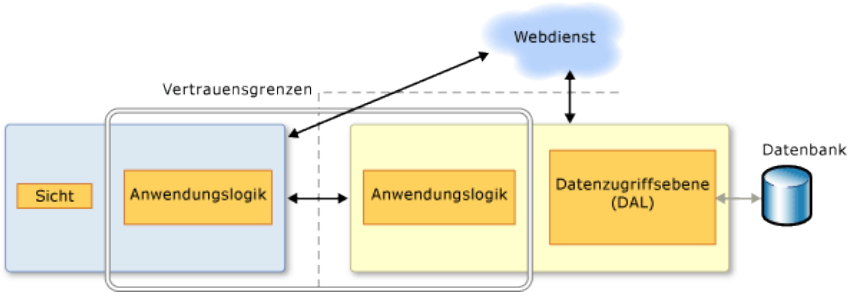

Abbildung 4.2: Struktur einer mehrschichtigen Anwendung
msdnlib

4.4.1.1 Entity Framework

ADO.NET Entity Framework ist ein Framework für Objekt-relationale Abbildung. Es ermöglicht die Erstellung von Objektmodelen (Entität Model), die auf Datenbankschemata (Tabellen, Ansichte usw.) abgebildet sind. Objektmodelle können im Code über **LINQ to Entities**[82] angefragt werden. Entity Framework übersetzt die LINQ-to-Entities Anfragen in SQL-Anfragen und gibt die Daten als Objekte des definierten Models zurück[83].
Das Model, z. B. auf Basis eines MySql-Datenbankes, kann mit Hilfe von ADO.NET Entity Data Model Vorlage in Visual Studio erstellt werden.

4.4.1.2 Domänendienste

Zugriff auf Serverdaten geschieht über Domänedienstklassen, die auf der Basis von existierenden Objektmodelen mit Hilfe entsprechender Vorlage erstellt werden können. Über Metadaten-Klassen, die partielle Klassen der erstellten Domänedienstklassen sind, werden Attribute (z. B. Validierungsregeln) für Entitäten gesetzt.

Konsumieren der Domänedienste Entitäten werden meistens über IQueryable<T>-Schnittstellen Ausdrücke von Domänenoperationen zuruckgegeben[84]. LINQ-Provider (in unserem Fall Entity Framework) führt den Ausdruck aus und gibt eine Auflistung der Entitäten zurück. Dies ermöglicht LINQ-Anfragen auf der Klientseite so zu definieren, dass Resultate der Anfragen bereits sortiert oder gefiltert zurückgegeben werden[85].

[82] [msdnlib]
[83] [Beres u. a. 2010, S. 101]
[84] [Beres u. a. 2010, S. 98]
[85] [Beres u. a. 2010, S. 108]

Alle Manipulationen der Entitäten innerhalb der zurückgegebenen Auflistung werden von dem DomainContext 4.4.1.4 in einem `changeset` verwaltet und beim Ausführen der `SubmitChanges` an den Domänedienst übergeben. Der Domänedienst überprüft darauffolgend den `changeset` auf Zugriffsrechte für bestimmte Operationen und Validierungsfehler und führt die beschriebenen Operationen auf der Datenbank durch.

Domäneoperationen Code für standardmäßige CRUD-Methoden wird bei der Erstellung der Domänedienstklasse automatisch von RIA Services generiert. Weitere Methoden können der Dienstklasse hinzugefügt werden. Man kann der Namenskonvention[86] folgen oder die Methoden mit der Methodenart beschreibenden Annotationen dekorieren, z. B. muss der Name einer Methode zum Aktualisieren von Entitäten aus Update, Change oder Modify und dem Namen der Entität bestehen oder über die Update-Annotation als solche definiert werden. Beide Methoden können gleichzeitig eingesetzt werden. Diese Namenskonvention wird für den Codegenerator des RIA Services benötigt4.4.

4.4.1.3 Verwaltung der Entitäten

Entitäten können mittels Attribute direkt oder über Metadatenklassen konfiguriert werden. Dies wird zur Definition der Validierungsregeln, Bestimmung der Erstellungsregeln von Entitäten auf der Klientseite und Darstellung dieser benutzt. In 4.6 ist eine Metadatenklasse vorgestellt, die Validierungsregeln für Entitäteneigenschaften sowie das Ausschließen bestimmter Eigenschaften von den Erstellung auf der Klientseite bestimmt.

Listing 4.6: Metadatenklasse zur Entitätenverwaltung

```
1  [MetadataTypeAttribute(typeof(truckersrates.truckersratesMetadata))]
2      public partial class truckersrates
3      {
4          internal sealed class truckersratesMetadata
5          {
6              private truckersratesMetadata()
7              {
8              }
9          [Required]
10         [StringLength(6)]
11             public string Client_CompaniesId { get; set; }
12         [Exclude]
13             public long idRates { get; set; }
```

[86] [Beres u. a. 2010, S. 109]

```
14          [RegularExpression("[0-9]*")]
15                public string PickUp_CitiesZipsId { get; set; }
16            }
17      }
18  }
```

4.4.1.4 Klientseitiger Code

Klientseitiger Code für die Kommunikation mit dem Server wird von RIA Services automatisch generiert. Für jedes Domänendienst wird eine DomäneContext-Klasse auf der Klientseite erzeugt, dabei die Methoden des DomäneContext haben Anfragenobjekte[87] als Rückgabetyp, die den Methoden der Dienstklasse entsprechen. Ausgenommen sind Methoden zur Manipulation der Entitäten, da diese nicht von Klient ausgeführt werden, sondern im changeset vom DomäneContext verwaltet werden. Genauso werden Proxyklassen für alle von Domänedienst verfügbar gemachten Entitäten unter Berücksichtigung, der für die Entitäten definierten Regeln, generiert[88].
Außerdem stellen RIA Services einen DomainDataSource-Steuerelement auf der Klientseite zur Verfügung, das direkt im XAML zur Kommunikation mit der DomänenContext benutzt werden kann. Die Verbindung eines auf dem Domänendienst basierenden DataSource-Elementes via Binding Markuperweiterung an die ItemSource-Eigenschaft des DataGrid-Elementes wurde bereits in 4.5 demonstriert.

4.4.1.5 Darstellungsmodelle

Oft sind Veränderungen auf der Datenzugriffsebene notwendig und, da die Präsentationsebene an Entitäten des Entity Framework Models gebunden ist, werden auch Veränderungen auf der Klientseite nötig, was bei einer komplexen Anwendung aufwendig sein kann. Darüber hinaus entsprechen die auf der Datenbank Basis erstellten Modelle nicht immer den auf der Präsentationsebene zu darstellenden Informationen, vor allem wenn Daten aus verschiedenen Entitäten in einem Element dargestellt werden müssen. Um dies umzugehen, werden Darstellungsmodelle benutzt, die eine Art Datentransferobjekte darstellen und eine Trennung zwischen Ebenen erlauben. Darstellungsmodelle werden als Klassen mit den für die Benutzeroberfläche relevanten Eigenschaften definiert. Der Code der Datentransferobjekte wird auf der Klientseite wiederum von RIA Services generiert.

[87] [Cleeren und Dockx 2010, S. 386]
[88] [Beres u. a. 2010, S. 95]

4.4.2 HTTP-Klassen

Zur Verbindung mit bereits existierenden Webdiensten über den HTTP-Protokoll stellt Silverlight HttpWebRequest bzw. HttpWebResponse oder WebClient bereit. Diese bieten die Funktionalität um Anfragen an einen Webdienst über HTTP-Protokoll zu senden und zu empfangen. Silverlight bietet keine Möglichkeit HTTP-basierte Dienste zu hosten. Praktischer Einsatz von HTTP-Klassen in Silverlight-Anwendungen ist in [Cleeren und Dockx 2010, S. 183] näher behandelt.

4.5 Lokalisierung

Silverlight Anwendungen werden mittels Ressourcedateien lokalisiert. Die Ressourcedateien werden auf Basis einer Vorlage erstellt, wobei Ressourcen hierarchisch aufgebaut sind. An der Spitze der Hierarchie befindet sich die Defaultressource gefolgt von regionalsneutralen und kulturspezifischen Ressourcen. Bei dem Versuch lokalisierte Ressourcen zu laden durchläuft der Ressource Manager die Hierarchie von unten nach oben.
Für die Bereitstellung können Silverlight Anwendungen alle oder nur die Defaultressource beinhalten. Nachladen benötigter Lokalisierungen zur Laufzeit macht besonders Sinn bei einer Großzahl an unterstützten Kulturen.

5 Vergleich

In diesem Kapitel wird ein Vergleich vorgestellter Technologien durchgeführt. Der Vergleich bezieht sich auf den Satz von Kriterien, die im Kapitel 1 definiert wurden.

5.1 Entwicklungsumgebung

Alle drei Plattformen ermöglichen den Einsatz von modernen Entwicklungsumgebungen über entsprechende Plugins. Silverlight und Flex bieten einige Werkzeuge, die GWT gar nicht oder in weniger handlicher Form besitzt.

5.1.0.1 Zugriff auf serverseitige Daten und Objekte

Silverlight stellt den Data Sources Editor zur Verfügung zur Bindung von serverseitigen Objekten an Elemente der Benutzeroberfläche. Data/Services-Editor von Flex ermöglicht nicht nur das Binden an Domäneobjekte sondern auch an Methoden der serverseitigen Dienste. GWT bietet solche Werkzeuge nicht an.

5.1.0.2 Designer

Alle drei Plattformen bieten eine Designer-Ansicht an. Designer für GWT ist eindeutig den entsprechenden Werkzeugen des Flex und Silverlight unterlegen und ermöglicht nur eine grobe Aufbau der Struktur der Benutzeroberfläche.

5.2 Benutzeroberfläche

Das System zur Beschreibung der Benutzeroberfläche der drei untersuchten Technologien unterscheiden sich nach den Grundkonzepten, die bei der Entwicklung der Anwendung umgesetzt werden. Daraus resultieren verschiedene Möglichkeiten zur Gestaltung der Benutzeroberfläche und somit auch der Aufwand.

5.2.1 Rendern der Benutzeroberfläche

Der Hauptunterschied der Benutzeroberflächen der Anwendungen in vorgestellten Technologien ist, dass diese für Silverlight- und Flex-Anwendungen aus grafischen Objekten und für GWT-Anwendungen aus HTML besteht. Parsing der Oberflächenhierarchie passiert für GWT und Flex zur Kompilierzeit und für Silverlight zur Laufzeit. Diese Unterschiede beeinflussen die Größe der Anwendungen und somit auch das zu übertragende Datenvolumen sowie auch die Ladezeit der Anwendung, was bei einem gleichzeitigen Einsatz auf einer großen Zahl von Klientmaschinen ein wichtiges Entscheidungskriterium ist. Dabei ist anzumerken, dass die Größe auch von weiteren Faktoren abhängt, die in weiteren Abschnitten besprochen werden.
Im Rahmen dieser Arbeit die erstellten Beispielanwendungen umfassten jeweils eine Tabelle zur Darstellung von Daten aus einem Datenbank und einer Form zur Erstellung neuer Domänenobjekte. Die Größe der erstellten Anwendungen wurde mit FireBug[89] im Mozilla Firefox Browser[90] gemessen und beträgt folgende Größen:

Tabelle 5.1: Größe einer Beispielanwendung mit unterschiedlichen Technologien

Plattform	**Größe**
Silverlight	*2,3 MB*
Flex	*2,5 MB*
GWT	*89,2 KB*

5.2.1.1 Deklaratives Layout

Alle drei Frameworke bieten einen deklarativen Ansatz für das Layout der Benutzeroberfläche: Silverlight mit XAML, Flex mit MXML und GWT mit UI-Binder. Die Vorteile und Nachteile dieser werden in Tabelle 5.2 zusammengefasst.

Tabelle 5.2: Vergleich der Instrumente verschiedener Frameworke zur deklarativen Beschreibung der Benutzeroberfläche

Plattform	**Vorteile**	**Nachteile**

[89] http://getfirebug.com/whatisfirebug
[90] http://www.mozilla.org/de/firefox/fx/

Silverlight	• Einbau von komplexen Szenarien dank der Inhalts- und Eigenschaftselement-syntax, den Angefügten Eigenschaften und vor allem den Markuperweiterungen 4.3, wie DataBinding • Einfache und schnell zu entwickelnde Struktur der Oberfläche - 2 oder mehr Klassen pro View	• Die Grenze zwischen Layout und dahinter stehender Logik nicht immer eindeutig • Schlechte Austauschbarkeit und Wiederverwendbarkeit der Komponenten
Flex	• Leichte Aufbau einer zustandsabhängigen Oberfläche über die State-Klasse 3.3.4 • Entwickeltes Verwaltungsmodel für Ereignisse 3.3.5	• Oberfläche wird in 2 Klassen beschrieben, wobei diese dank dem State-Konzept mehrere Views in einer Datei umfasst • Einbuße in Austauschbarkeit und Wiederverwendbarkeit der Komponenten
Google Web Toolkit	• UI-Binder-Vorlage beschreibt nur das Layout und ist strikt von der Logik getrennt • Leichte Austauschbarkeit und Wiederverwendbarkeit der Komponenten der Oberfläche, dank höherer Modularität der Struktur	Aufwändiger Aufbau der Oberfläche, die 4 und mehr Klassen pro View erfordert

5.2.2 Listenbasierte Datendarstellung

Zur Darstellung von Daten in Tabellenform bieten alle Plattformen entsprechende UI-Komponenten. Wie schon oben erwähnt wurde, sind UI-Elemente von Silverlight und Flex grafische Objekte. In diesen werden die Daten vom entsprechenden Plugin eingepflegt. GWT rendert dagegen die CellWidgets 2.3.2.1 für die Datenpräsentation in einen HTML-String unter Benutzung von innerHTML. Die Spaltendaten werden von entsprechenden Cell-Klassen wiederum in HTML gerendert. Damit wird eine leichtgewichtige Darstellung von Daten unter dem Einsatz der nativen Browser Werkzeugen erreicht.

5.3 Kommunikation mit externen Daten

Für die Kommunikation mit externen Daten bieten alle Plattformen eigene, auf die optimierten Lösungen. Diese ermöglichen zwar das Verbinden mit beliebigen externen Diensten, aber der Einsatz einer bestimmten Backend-Technologie bietet immer die meisten Möglichkeiten zur Optimierung des Datenaustauschs.
Dabei ist anzumerken, dass Silverlight sowie Flex das Definieren der Anwendungslogik auf der mittleren Schicht erlauben und der komplette Code der Klientseite wird vom Compiler generiert. Dies ist jedoch nicht der Fall bei der Entwicklung mit GWT. Hier muss ein Teil des Codes für die Klientseite per Hand definiert werden, wie Proxy- und Request-Schnittstellen 2.4.1. Dies ist zwar nicht aufwendig, kann aber, wie es auch im Rahmen dieser Arbeit der Fall war, zu Problemen bei der Suche nach Fehlern führen.

5.4 Programmiermodel

Verschiedene Programmiermodelle werden zur Entwicklung in vorgestellten Frameworken vorgeschlagen, die jeweils Vorteile und Nachteile in Bereichen Testbarkeit, Wiederverwendbarkeit usw. mit sich bringen. Eine Entwicklung mit Silverlight und Flex nach dem vorgeschlagenen Model geschieht unter Einsatz von externen Tools, wobei die genaue Umsetzung, besonders bei Silverlight, vom Werkzeug abhängt. GWT bietet ein Paket von Klassen, so dass eine Entwicklung nach dem entsprechenden Muster sowie Integration verschiedener Komponenten von Anfang an berücksichtigt wird. Dies ermöglicht das Erzielen einer optimierten Anwendung, was aber einen größeren Aufwand mit sich zieht.

5.5 Betriebssystem und Browserunabhängigkeit

Eine Unterstützung der diversen Plattformen ist eine der wichtigen Voraussetzungen für wirtschaftliche Anwendungen, die von zahlreichen Benutzern in verschiedenen Umgebungen zum Einsatz kommen. In diesem Sinne unterscheiden sich die untersuchten Frameworke.
Flex mit dem Flash Player Plugin ist für alle Browser und Plattformen vorhanden, außer mobiler Geräte von Apple[91]. Das Plugin ist auf 96 Prozent[92] der Browser installiert.
Silverlight-Plugin ist nur für Windows und Macintosh Betriebssystemen verfügbar, Opera-Browser wird offiziell nicht unterstützt. Die Verbreitung von Silverlight-Plugin liegt laut Statistik[93] bei 76 Prozent.
GWT-Anwendungen brauchen kein Plugin und laufen in jedem Browser, in dem JavaScript aktiviert ist, vom Betriebssystem unabhängig. Angesichts der Tatsache, dass GWT-Anwendungen auch speziell angepasst für jeden aktuellen Browser separat optimiert und kompiliert werden, ist die Lauffähigkeit in unterschiedlichen Umgebungen auch in Zukunft zu erwarten.

5.6 Optimierungen für Einsatz

Ein der wohl wichtigsten Vorteilen von GWT ist, dass die Anwendung in mehrere auf Browser und Sprachumgebung spezifizierten Versionen kompiliert wird, was die Größe des Endprodukts, das von jedem Benutzer herunterzuladen ist, verringert. GWT-Compiler führt eine Anzahl von Optimierungsmaßnahmen, wie Dead Code Elimination 2.1. Bei der Entwicklung mit z. B. Silverlight müssen für diese Aufgaben externe Tools, wie XAPOptimizer[94], eingesetzt werden. Für Flex und Silverlight-Anwendungen können nur die lokalisierten Ressourcen separat nachgeladen werden.

[91] http://www.apple.com/hotnews/thoughts-on-flash/
[92] [riastats]
[93] [riastats]
[94] http://www.componentone.com/superproducts/xapoptimizer/

6 Zusammenfassung

In der vorliegenden Arbeit wurden unterschiedliche Technologien zur Erstellung von Rich Internet Applications untersucht, die für einen gleichzeitigen Einsatz von mehreren Parteien zum Austausch von Informationen über die wirtschaftliche Vorgänge geeignet sind. Es wurden für die Entwicklung solcher Anwendungen relevante Instrumente, wie Techniken zur Kommunikation mit externen Datenquellen, dieser Frameworke betrachtet und dann in Beispielanwendungen eingesetzt. Anschließend wurde ein Vergleich dieser vorgenommen.
Adobe Flex ergab sich als ein am meisten für die schnelle Entwicklung einer Zustandbasierter Webanwendung geeignet, was vor allem beim Erstellen von sogenannten Click-Dummys von Vorteil ist. Jedoch weisen die Flex-Anwendungen eine schwache Trennung verschiedener Schichten, was einen Einfluss auf die Testbarkeit und Wiederverwendbarkeit von Komponenten hat sowie darauffolgend auch einen höheren Wartungsaufwand nachweist. Microsoft Silverlight ermöglicht über die entsprechenden Werkzeuge auch eine schnelle Erstellung einer Daten-darstellenden Oberfläche, die aber im Flex deutlich schneller erstellt werden könnte. Silverlight bietet dagegen eine bessere Modularität der Anwendung. Für das Ausführen der Beispielanwendungen auf der Basis dieser beiden Frameworke geladenen Daten wiesen eine 25-fache Größe im Vergleich zur Google Web Toolkit Anwendung gleicher Funktionalität. Dieser Unterschied ist teilweise darauf zurückzuführen, dass bestimmte Komponenten der Anwendungen nicht in den entsprechenden Laufzeitumgebungen enthalten sind und zusammen mit der Anwendung geladen werden müssen. Ein weiterer Nachteil von Silverlight ist eine geringere Unterstützung verschiedener Betriebssysteme und Browser im Vergleich zur anderen Technologien.
Google Web Toolkit erschien als am meisten für die Entwicklung einer Webanwendung geeignet, die den im Rahmen dieser Arbeit definierten Kriterien entspricht. Google Web Toolkit bietet in allen Bereichen der Entwicklung etliche relevante Optimierungsmöglichkeiten. Die wichtigsten davon beeinträchtigen Große der entstehender Anwendung, Minimierung des Datenverkehrs, Testbarkeit, Austauschbarkeit der Komponenten sowie eine explizite Berücksichtigung der Browserunterschiede. Um die Optimierungen maximal zu halten, bietet Google Web Toolkit Klassen zur Entwicklung nach dem vorgeschlagenen MVP-Entwurfsmuster. Die Implementierung dieser optimalen Lösungen verringert den zukünftigen Wartungsaufwand und ermöglicht dank der hohen Modularität einen schnellen Austausch der Komponenten mit

geringem oder keinem Einfluss auf andere, ist aber anfangs mit einem höheren Aufwand in Vergleich zu anderen Technologien verbunden. Dieser ist auf die spezifischen Entwicklungsszenarien aber auch Mängel im Bereich der Entwicklungsumgebung zurückzuführen.
Den Vergleichsresultaten folgend wurde in Rahmen dieser Arbeit eine Beispielanwendung mit Hilfe von Google Web Toolkit erstellt. Die Implementierung miteinbezog die meisten zur Verfügung stehende Instrumente und Optimierungen. Die resultierende Anwendung ist ausbaufähig und unter http://connection-point.appspot.com/ online verfügbar.

Literaturverzeichnis

Beres u. a. 2010

Beres, J. ; Evjen, B. ; Rader, D.: *Professional Silverlight 4*. John Wiley & Sons, 2010 (Wrox Programmer to Programmer). http://books.google.de/books?id=PhaclwHda28C. – ISBN 9780470650929

blazeds

Adobe Open Source BlazeDS. http://opensource.adobe.com/wiki/display/blazeds/BlazeDS,

census

Data rendering and loading performance comparison for RIA technolohies. http://www.jamesward.com/census2/,

Cleeren und Dockx 2010

Cleeren, G. ; Dockx, K.: *Microsoft Silverlight 4 Data and Services Cookbook*. Packt Publishing, Limited, 2010 http://books.google.de/books?id=ntP4RQAACAAJ. – ISBN 9781847199843

Czernicki 2009

Czernicki, B.: *Next-Generation Business Intelligence Software with Silverlight 3*. Apress, 2009 (Apress Series). http://books.google.de/books?id=83LMgWGba6oC. – ISBN 9781430224877

Czernicki 2010

Czernicki, B.: *Silverlight 4 Business Intelligence Software*. Apress, 2010 (Apress Series). http://books.google.de/books?id=JFpyG86gKdwC. – ISBN 9781430230601

Ernst 2010

Ernst, Timo: *Performance Analysis and Acceleration for Rich Internet Application Technologies*, University of Ulm, Diplomarbeit, 2010

Fain u. a. 2010

Fain, Y. ; Rasputnis, V. ; Tartakovsky, A.: *Enterprise Development with Flex: Best Practices for RIA Developers*. O'Reilly Media, 2010 (Adobe Developer Library). http://books.google.de/books?id=kjCaK4jXOJEC. – ISBN 9780596154165

flexdata

Zugriff auf Daten mit Flex. http://help.adobe.com/de_DE/Flex/4.0/AccessingData/index.html,

flexsdk

Using Flex 4. http://help.adobe.com/de_DE/Flex/4.0/UsingSDK/index.html,

gwtact

GWT Development with Activities and Places. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideMvpActivitiesAndPlaces.html,

gwtdev

Google Web Toolkit Documentation - Compile and Debug. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideCompilingAndDebugging.html,

gwtdom

Classes for low-level DOM programming. http://google-web-toolkit.googlecode.com/svn/javadoc/latest/com/google/gwt/dom/client/package-summary.html,

gwtui

Google Web Toolkit Documentation - Build User Interfaces. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideUi.html,

gwtuibin

Google Web Toolkit Documentation - Declarative Layout with UiBinder. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/DevGuideUiBinder.html,

gwtunit

Unit Testing GWT Applications with JUnit. http://code.google.com/intl/de-DE/webtoolkit/doc/latest/tutorial/JUnit.html,

Janisch 2008

Janisch, Gerhard: *Analyse von Rich Internet Application Frameworks am Beispiel einer Thesaurusverwaltung*, Fachhochschule Oberösterreich, Diplomarbeit, 2008

Kereki 2010

Kereki, F.: *Essential GWT: Building for the Web with Google Web Toolkit 2*. ADDISON WESLEY (PEAR, 2010 (Developer's Library). http://books.google.com/books?id=rLjf8rSkOM4C. – ISBN 9780321705143

Koch

KOCH, Peter-Paul: *Benchmark - W3C DOM vs. innerHTML*. http://www.quirksmode.org/dom/innerhtml.html,

Kunath 2010

KUNATH, Mirko: *Entwicklung einer Rich Internet Application mit Flex unter Verwendung von Architektur-Frameworks*, Hochschule für Technik und Wirtschaft (FH), Diplomarbeit, 2010

livecycle

Adobe Live Cycle. http://www.adobe.com/products/livecycle/dataservices/,

Louis und Müller 2007

LOUIS, D. ; MÜLLER, P.: *Das Java 6 Codebook*. Addison-Wesley, 2007 http://books.google.de/books?id=jxJKE2QA9L4C. – ISBN 9783827324658

Moock 2007

MOOCK, C.: *Essential ActionScript 3.0*. O'Reilly, 2007 (O'Reilly Series). http://books.google.de/books?id=gUHX2fcLKxYC. – ISBN 9780596526948

Moritz 2010

MORITZ, Florian: *Rich Internet Applications (RIA): A Convergence of User Interface Paradigms of Web and Desktop Exemplified by JavaFX*, Fachhochschule Kaiserslautern, Diplomarbeit, 2010

msdnlib

MSDN Library. http://msdn.microsoft.com/de-de/library/,

Ramsdale 2010

RAMSDALE, Chris: *Large scale application development and MVP*. http://code.google.com/intl/de-DE/webtoolkit/articles/mvp-architecture.html, 2010

riastats

Rich Internet Application Statistics. http://www.riastats.com/,

Widjaja 2010

WIDJAJA, S.: *Adobe Flex 4*. Hanser Fachbuchverlag, 2010 http://books.google.de/books?id=ZgvSSAAACAAJ. – ISBN 9783446422681

zend

Zend Framework. http://framework.zend.com/,