Praxisbericht: Prototypische Implementierung eines JavaFX/Web-Channels zur Integration ins Multichannel-Framework der deg

Niels Gundermann

31. März 2014

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Ausgehend von dem Fazit des Praxisberichtes "Prototypische Implementierung eines JavaFX-Channels zur Integration ins Multichannel-Framework der deg" ist Swing ein veraltetes Framework, dass durch ein neues ersetzt werden sollte. Dies bezieht sich nur auf die Userinterfaces für den Desktop.
PROFIL-C/S wird nicht nur als Desktopanwendung verwendet, sondern auch in Browsern mithilfe eines Web-Clients. Das bedeutet, dass bei einer Neuausrichtung der Technologien für die unterschiedlichen Channels auch der Einsatz in Browsern bedacht werden muss.
Derzeit setzt die deg das wingS-Framework für die Darstellung in Browsern ein. Dieses Framework ist - genauso wie Swing - veraltet und wird nicht mehr gewartet. Support für dieses Framework wird vergebens gesucht.
Aufgrund dessen wird in dieser Arbeit nach einer Alternative für den Web-Channel der deg und PROFIL C/S gesucht.
Da sich der vorherige Praxisbericht mit JavaFX beschäftigte, wird zu Beginn auf Möglichkeiten eingegangen, den implementierten Prototypen für den JavaFX/Desktop-Channel direkt im Browser zu darzustellen.
Im Anschluss wird das Vaadin-Framework, als Beispiel für ein mögliches Web-Framework, genauer beleuchtet. Dazu gehört auch ein Vergleich zum bestehenden wingS-Framework.
Zum Ende wird ein Prototyp einer PROFIL-C/S GUI für den Einsatz im Web vorgestellt und auf Probleme in Bezug auf das Multichannel-Framework der deg eingegangen.

2. JavaFX im Web?

Ausgehend davon, dass es Ziel sein soll, einen Web-Framework für die deg zu evaluieren und prototypisch zu implementieren, biete es sich an zu prüfen, ob der schon bestehende Prototyp mit dem JavaFX-Framework ohne größere Anpassungen im Web gerendert werden kann. Von einer ähnlichen Darstellung auf den unterschiedlichen Oberflächen (Desktop und Browser) durch das gleiche Framework wird dabei ebenfalls ausgegangen.
JavaFX bietet die Möglichkeit sowohl auf dem Desktop, als auch in einem Browser gerendert zu werden [Ora13a] Hierbei handelt es sich um ein Java-Applet. Das Deployment solcher Applets ist jedoch vergleichsweise sehr zeitaufwändig. Der Grund dafür ist vor allem, dass die JVM zuvor initialisiert werden muss und das Applet vor Ausführung komplett heruntergeladen werden muss. [Ame11] Dem entgegenzuhalten ist, dass man

somit gezwungen ist, die Clients möglichst klein[1] zu halten. Allerdings sind Größe und Funktionalität von Client und Server in einem bestehenden System schwer beeinflussbar. Weiterhin ist es auch nicht das Ziel die Clients umzustricken.
Der große Vorteil dieser Technologie ist, dass die GUI für beide Oberflächen nur einmal entwickelt werden muss. Die Zusammenarbeit mit Applikationsservern, was für Profil C/S unbedingt von Nöten ist, ist nicht ohne Weiteres möglich. Dafür müssen die Applets signiert werden. Das bedeutet, dass das Applet mit einem digitalen Zertifikat unterzeichnet werden muss. So wird gewährleistet, dass es sich dabei um ein von der deg enntwickeltes Programm handelt. Folglich kann der Nutzer eventuelle Fehler und aufkommende Sicherheitslücken genau zuordnen. [Ora13b]
Notwendig für die Nutzung von Java-Applets ist eine entsprechende html-Datei, in der eine jnlp-Datei mittels Java-Script eingebunden wird. Weiterhin muss der Anwender mit einem Browser arbeiten, in dem das Java-Plugin integriert oder installiert werden kann. Zusätzlich ist die Aktivierung von Java-Script und damit das Umgehen von Sicherheitsrichtlinien von Nöten. [Ora14]
Abbildung 1 zeigt eine Tabelle in der Vor- und Nachteile dieser Technologie aufgelistet sind.

3. Web-Frameworks

Web-Frameworks bieten die Möglichkeit bestimmte Inhalte mit Hilfe von Technologien zu visualisieren, die vor allem in Internet-Browsern benutzt werden. In der Regel ist das Ausführen von Programmen, die mit Web-Frameworks implementiert wurden, auf dem Desktop nicht möglich. Wir ein solches Framework zum Beispiel neben JavaFX eingesetzt, ist es Möglich die Logik zentral zu implementieren und die Visualisierung zu separieren.
Der Nachteil, in Bezug auf die Implemetierung, ist, dass für unterschiedliche Frameworks, die unterschiedliche Technologien nutzen, auch doppelter Aufwand aufkommt. Um diese doppelten Aufwad möglichstklein zu halten, wurde dafür bei der deg ein Multichannel implementiert.

[1] bedeutet in diesem Kontext, dass sich die Funktionen, die im Client implementiert sind, weites gehend nur auf die GUI und entsprechenden Listener, die zum Steuern der Anwendung benötigt werden, beschränkt. Die Logik und Funktionen, die über die Bedienung der GUI hinaus gehen, sollten sich nach Möglichkeit auf dem Server befinden - unter der Voraussetzung, dass dort genügend Speicher vorhanden ist, um die Anfragen der Clients in angemessener Zeit beantworten zu können

3.1. Multichannel

Bezogen auf das Problem des doppelten Aufwands, führt der Multichannel unterschiedlichen Frameworks für die Visualisierung auf unterschiedlichen Oberflächen zusammen. Die Userinterfaces werden dann in der Syntax und Semantik des Multichannel-Framework geschrieben. Die Entwickler müssen somit nicht die unterschiedlichen GUI-Frameworks kennen, sondern nur das Multichannel-Framework. Je nachdem, auf welcher Plattform das Programm ausgeführt wird, wird die GUI vom Multichannel über das entsprechende Framework erstellt.
Um einen Multichannel implmentieren zu können, bedarf es einer gemeinsamen Basis, der unterschiedlichen Frameworks. Die von den Frameworks genutzten Technologien sind hierbei in erster Linie uninteressant. Das Ziel ist ein allgemeines Framework zu schreiben, so dass nur noch unterschieden werden muss, auf welcher Visualisierungsplattform man sich befindet.
Ausgegangen wird davon, dass als GUI-Framework für den Desktop JavaFX oder - wie aktuell - Swing genutzt wird. Somit liegt es nahe ein Framework zu betrachten das auf Java basiert, um die JVM als gemeinsame Basis nutzten zu können. Aus diesem Grund wird im Folgenden das Vaadin-Framework vorgestellt.

3.2. Vaadin

Das Vaadin-Framework ist ein Open-Source Framework, das es erlaubt Userinterfaces für Weboberflächen zu implementieren. Dies kann wie bei JavaFX auf zwei Wegen geschehen. Zum einen durch Java-Code und zum Anderen durch einen Editor [Mar13][S. 237ff.]. Im weiteren Verlauf wird auf die erste Möglichkeit Bezug genommen.
Entwickelt wurde Vaadin vorerst als Ajax-Framework. Damals trug es noch den Namen *IT Mill Toolkit*. Mit dem Einsatz des GWT für das clientseitige Komponenten-Rendering, wurde auch der Name im Jahr 2007 geändert. Seit dem trägt es den Namen *Vaadin* [Mül11].
Es enthält einen großen Pool an UI-Komponenten, die dem Nutzer für den Aufbau einer Web-GUI zur Verfügung stehen. Genauso wie bei anderen GUI-Frameworks, die auf Java basieren, ist es möglich neue und eigene GUI-Komponenten zu entwickeln. Aufgrund dieser Möglichkeitungund durch die große Community gibt viele AddOns für das Vaadin-Framework, die noch nicht in das Standardframework integriert wurden.
Was den Service bzw. die Unterstützung betrifft, gibt verständliche Tutorials, eine gute

Dokumentation sowie entsprechendee Plugins für unterschiedliche Entwicklungsumgebungen. Dadurch findet man leicht einen Zugang zu dem Framework und kann es ohne große Konfiguration einsetzen. Auch wenn in der offiziellen Dokumentation [Mar13] lediglich die Standardkomponenten erklärt sind, gibt es zu jedem AddOn eine kurze Einweisung, sowie Beispiele und Erklärungen. Auch die Community macht einen sehr aktiven Eindruck [2]. Des Weiteren stellen die Vaadin-Entwickler ihr Know-How als externe Berater zur Verfügung.
Implementieren lassen sich die GUI-Komponenten nicht nur mittels Java-Code. Da Vaadin auf der Client-Seite auf das Google-Web-Toolkit (GWT) aufbaut, können die UIs auch mittels anderer Sprachen, die zu der JVM kompatibel sind[3], erstellt werden. Weiterhin kann Vaadin auf allen Servern bereitgestellt werden, die einen Servlet- oder Portlet-Container enthalten. Beispiele dafür sind Tomcat, Glasfish oder der in der deg eingesetzte JBoss.
Die Entwicklung findet auf dem Server statt. Für die Kommunikation zwischen Server und Client wird AJAX verwendet [Mar13][S. 23ff.]. Von daher sind zusätzliche Plugins für Java-Anwendungen nicht nötig und somit besteht auch das durch Plugins enstehende Sicherheitsrisiko nicht mehr.

3.3. Anforderung an die Profil C/S - GUI

Sollte es zu einer Implementierung eines neuen Multichannel-Frameworks kommen, müssen Techologien und Frameworks eingesetzt werden, die den Anforderung von Profil C/S gerecht werden. Im Folgenden wird untersucht, ob es möglich ist mit dem Vaadin-Framework eine allgemeine GUI, wie sie in Profil C/S zur Anwendung kommt, zu implemetieren. Dabei werde ich mich an dem Userinterface, welches aus dem vorangegangen Praxisbericht - "Prototypische Implementierung eines JavaFX-Channels zur Integration ins Multichannel-Framework der deg" hervorgegangen ist, orientieren. Ziel dabei ist vor allem, die gleichen Funktionalitäten umzusetzen.
Die Logik konnte aufgrund dessen, dass beide Frameworks auf Java aufsetzen komplett übernommen werden.
Da Vaadin alle Komponenten, welche für die Umsetzung des Prototypen notwendig waren, enthält, musste der Aufbau nicht verändert werden, sondern nur einige Typen angepasst werden. Im Folgenden sind die Vaadin-Komponenten aufgelistet die zur Umsetzung not-

[2] https://vaadin.com/forum
[3] Bspw. Scala, Ruby, Groovy, Clojure, Python

wendig waren.

```
MenuBar;
Tree;
VerticalLayout;
HorizontalLayout;
HorizontalSplitPanel;
VerticalSplitPanel;
GridLayout;
BorderLayout;
Table;
TabSheet;
```

Bei der Implementierung der Menüleiste, gab es keine weiteren Probleme. Hier bietet Vaadin einen entsprechend vorgefertigten *MenuBar*-Typ an. Dieser kann mit Elementen gefüllt werden, wie es auch schon von JavaFX bekannt ist. Auch die Verschachtelung von Untermenüs und das Anfügen von Icon an die Menüpunkte ist durch die Methode *addItem(TEXT, ICON)* möglich.

```
MenuBar menu = new MenuBar();
MenuBar.MenuItem schliessen = menu.addItem("Schließen", null);
```

Um Aktionen an diesem Menü ausführen zu können muss auch bei Vaadin ein entsprechender Listener implementiert. Dies geschieht über die Methode *setCommand(COMMAND)*. Diese Listener-Interfaces gleichen von der Implementierung her den Action-Listener, aus dem Swing- oder JavaFX-Framework.

```
public class SchliessenCommand implements Command {
        private MappenView vertragsMappenView;

        @Override
        public void menuSelected(MenuItem selectedItem) {
                vertragsMappenView.getUI().close();
        }
}
});
```

Für die Umsetzung geteilter Layouts bietet Vaadin zwei Komponenten an, die je nach Orientierung der Teilungsrichtung verwendet werden sollten. Dazu gehört zum Einen das *VerticalSplitPanel* - für die vertikale Teilung - und das *HorizontalSplitPanel* - für die horizontale Teilung.
Voreinstellung für die Aufteilung der geteilten Bereiche können durch die Methode *setSplitPosition(VALUE, UNIT)* getroffen werden. Die *UNIT* bestimmt dabei, ob es sich um einen absoluten, oder um einen prozentualen, Wert handelt.

Mit dieser Komponente ist es jedoch nur möglich einen Bereich in zwei weitere Bereiche aufzuteilen. Diese können dann mittels *setFirstComponent(COMPONENT)* und *setSecondComponent(COMPONENT)* befüllt werden.

```
1 VerticalSplitPanel vertikalerSplit = new VerticalSplitPanel();
2
3 vertikalerSplit.setSplitPosition(30, Sizeable.Unit.PERCENTAGE);
4
5 vertikalerSplit.setFirstComponent(Component);
6 vertikalerSplit.setSecondComponent(Component);
```

Tabellen sind weitaus komplexer strukturiert. Die Spalten werden bei Vaadin als Container betrachtet. Das hat den Vorteil, dass je Spalte auch festgelegt werden kann, welcher Datentyp dort angezeigt wird. Das geschieht mit der Methode *addContainerProperty(TEXT, TYPE, DEFAULT)*. Allgemein ist somit gewährleistet, dass nur Typen angezeigt werden, die auch angezeigt werden sollen. Bezogen auf objektorientierte Programmierung, ist diese Art der Strukurierung vorteilhaft, da mit Objekten und nicht mit nativen Datentypen gearbeitet wird. Im Prototypen wird dieser Vorteil jedoch nicht genutzt. Der Grund dafür ist, dass für Zeichenketten und Zahlen keine spezifischen Typen implementiert wurden.

```
1 Table table = new Table();
2 table.addContainerProperty("Vorgang", String.class, "");
3 table.addContainerProperty("Status", String.class, "");
4 table.addContainerProperty("Zuwendungs-\nsumme[EUR]", String.class, "");
5 table.addContainerProperty("Zahlungs-\nbetrag[EUR]", String.class, "");
6 table.addContainerProperty("Zahlungs-\ndatum", String.class, "");
7
8 initLines(table);
9 ...
10
11 private void initLines(Table table) {
12         int i = 0;
13         for(Document teilvorgang : teilvorgaenge){
14                 TeilvorgaeneTableData data = new TeilvorgaeneTableData(teilvorgang);
15                 table.addItem(new Object[]{ data.getVorgang(), data.getStatus(),
16                 data.getZuwendungssumme(), data.getZahlungsbetrag(), data.getZahlungsdatum()}, i++);
17         }
18 }
```

Will man Daten in die Tabelle einfügen, so hat man auf objektorientierter Ebene viele Freiheiten, da die Methode *addItem(OBJECT[])* ein Array mit Werten vom Typ *Object* erwartet. Ist der Typ durch *addContainerProperty(TEXT, TYPE, DEFAULT)* festgelegt, wird für den Fall, dass ein falscher Typ in die Tabelle geschrieben werden soll, der Standardwert eingetragen. In der Klasse *TableTest* im Anhang ist ein solcher Fall implementiert

[4]. Bezüglich der Typsicherheit müssen hier noch Erweiterungen vorgenommen werden. Bei Angabe eines falschen Types sollte nicht einfach der Standardwert zurückgegeben werden. Vielmehr sollte hier darauf hingewiesen werden, dass der Falsche Typ verwendet wurde. Dies könnte durch eine generische Erweiterung der Vaadin-*Table* und einem Container, der die zu visualisierenden Daten des Businessobjekts enthält, realisiert werden. Durch den Container wird dann abgesichert, dass die Tabelle keine inkompatible Typen enthält. Im Prototypen geschieht dies in der Klasse *CommonTable*. Hier fungiert die Methode *getTableValues()* als ein solcher Container, der die Werte und die dazugehörigen Typen mitliefert. Da die Klassen, die von *CommonTable* erben, generisch auf das entsprechende Datenobjekt abgestimmt sind, kann es nicht mehr zu inkompatiblen Typen kommen.
Beim Befüllen des Verweise- und Inhaltsbaum traten keine weiteren Probleme auf. Hierbei handelt es sich um eine einfache Eltern-Kind-Beziehung. Jedoch muss beim Hinzufügen zum Baum, auch das Elternelement explizit gesetzt werden. Dieses Problem kann allerdings durch Auslagerung in eine allgemeine Methode vereinfacht werden.

```
1 private void appendDocuments(Document doc, Tree tree) {
2         for (Document children : doc.getUnterDokumente()) {
3                 tree.addItem(children.getTitel());
4                 tree.setParent(children.getTitel(), getRootItem(tree));
5                 tree.setChildrenAllowed(children.getTitel(), false);
6         }
7 }
```

Bei den Klick-Events für die Bäume ist folgende Codezeile besonders wichtig.

```
1 setImmediate(true);
```

Das hat zur Folge, dass der Listener sofort auf den Klick reagiert. Ansonsten wird das entsprechende Event später abgesetzt. Das größte Problem bei der Umsetzung des Prototypen war die Toolbar. Hierfür gibt es keine Standardkomponente, wie man es von anderen UI-Frameworks gewohnt ist. Es gibt zwar zwei Toolbar-Add-Ons, die in das Projekt eingebunden werden können [Jon, Jou], aber diese werden ab Vaadin 7 nicht mehr unterstützt.
Aufgrund dessen musste ich für den Prototypen eine eigene Toolbar implementieren. Den Aufbau habe ich dabei so einfach wie möglich gehalten. Die Toolbar ist ein Layout-Container in den mehrere Buttons eingefügt werden.

```
1 HorizontalLayout toolBar = new HorizontalLayout();
```

[4] siehe Abbildung 3

```
String basepath = VaadinService.getCurrent().getBaseDirectory().getPath();
FileResource imageDrop = new FileResource(new File(basepath+"/img/TbCopy.gif"));
FileResource imagePrint = new FileResource(new File(basepath+"/img/TbPrint.gif"));

Button btDrop = new Button();
btDrop.setIcon(imageDrop);

Button btPrint = new Button();
btPrint.setIcon(imagePrint);
btPrint.setEnabled(false);

toolBar.addComponents(btDrop, btPrint, btLossOrg, btGetOrg, btHelp);
```

Der Web-Client der deg hält die unterschiedlichen Fenster in Tabs bereit. Dadurch musste im Prototypen eine Tab-Ansicht als übergreifendes GUI-Element implementiert werden. Dadurch veränderen sich auch die Methoden zum öffnen neuer Mappen. Das einzige Problem welches dann noch bleibt ist, die Tab-Ansicht für jeden Tree-Controller zugänglich zu machen, da im Controller die Events zum öffnen einer neuen Mappe implementiert sind. Für den Prototypen wurde die Tab-Ansicht einfach statisch implementiert.

```
static TabSheet tabsheet = new TabSheet();
VertragsMappenView vertragsMappe = new VertragsMappenView(null);
// Parameter ist die Bezeichnung der zu oeffnenden Mappe
// null = Standard-Mappe
tabsheet.addTab(vertragsMappe).setCaption(vertragsMappe.getTitle());
tabsheet.getTab(vertragsMappe).setClosable(true);
```

Das Styling lässt sich bei Vaadin auch mittels Cascading-Style-Sheets umsetzen. Somit ist es möglich dieselbe Datei zu verwenden, die auch bei dem JavaFX-Prototyen zum Einsatz kommt. Die Einbindung der CSS-Datei erfolgt bei Vaadin jedoch auf einem anderen Weg. Hier wird sie in folgendes Verzeichnis gelegt: *WebContent/VAADIN/themes/sampletheme/sample.css*. Im Code erfolgt die Einbindung in der Hauptklasse durch eine Annotation.

```
@Theme("sampletheme")
public class MyApplication extends UI {
```

Innerhalb der CSS-Datei ist es wichtig, dass ein Standard-Theme importiert wird. Es sei denn das eigene Layout umfasst das komplette Styling und bedarf somit keiner Standardeinstellung. Bei den Standards handelt es sich um zwei unterschiedliche Layouts, die wie folgt importiert werden. Für den Prototypen wurde aufgrund des großen Aufwands für ein komplettes Theme ein Standard-Theme importiert.

```
@import url("<../reindeer/styles.css"<)
//oder
```

```
3 @import url("<../runo/styles.css"<)
```

3.4. Probleme der deg im Web

Große Probleme in der deg bezüglich der GUI des Web-Clients bereitet die Strukturierung. Im Desktop-Bereich wird dazu das GridBag-Layout genutzt. Die Web-Frameworks stellen ein solches Layout nicht bereit.
Aus diesem Grund wird in den Web-Client derzeit auf eine Strukturierung mittels Tabellen-Layouts zurückgegriffen [winb, winc]. Nach den Kommentaren im Code orientiert sich das Gridbag-Layout von wingS an das Gridbag-Layout von Swing. Allerdings wird dabei nur erreicht ein Swing-Gridbag-Layout so ähnlich wie möglich nachzubauen. Das zeigt, dass das hier verwendetet Framework keine optimale Lösung ist, da das Gridbag-Layout ein zentraler Bestandteil der Profil C/S-GUI ist.
Eine Lösung für die Nutzung von Gridbag-Layouts mit Vaadin oder anderen Web-Frameworks kann leider nicht gegeben werden. Eine kleiner Versuch zeigt aber, dass mit Vaadin ein solches Layout schon mit weitaus weniger Komplexität umsetzbar ist[5]. Dabei erweitert die Klasse *GridBagLayout* das Vaadin GridLayout. Die Strukturierung des GridLayouts von Vaadin ist in die allgemeine Layoutstruktur von Vaadin integriert [IT]. Aus diesem Grund ist es für eine strukturierte und einheitliche GUI besser geeignet, als eine Lösung, die aus der herkömmlichen Layout-Struktur herausfällt, wie es bei wingS der Fall ist.
Probleme bereiten weiterhin die Zwischenabstände der Komponenten (Insets). Diese müssten im Web über einen *margin* realisiert werden, der bei Vaadin nicht direkt gesetzt werden kann, sondern nur über CSS änderbar ist.

3.5. Vergleich zu wingS

Bei wingS handelt es sich um das Web-Framework, welches momentan von der deg eingesetzt wird. Das korrespondierende Framework für den Desktop-Client ist derzeit Swing [wina]. Das größte Problem das sich bei dem Einsatz von wingS abzeichnet ist der rückläufige Support. Die letzten Updates für wingS stammen aus dem Jahr 2008. Seit dem wurde das Framework nicht weiterentwickelt. Das lässt ein baldiges aussterben vermuten.
Die größte Gemeinsamkeit von wingS und Vaadin besteht darin, dass beide die Kommunikation zwischen Serven und Client über AJAX abwickeln. [Mar13][S 23.] [wina]
Die größten Unterschieden bestehen darin, das Vaadin eine weitaus größere Community

[5] siehe im Code das Package com.example.vaadinprofilsample.guicomponents.gridbag

hat. Das liegt unter anderem auch daran, dass Vaadin auf dem GWT aufsetzt. Weiterhin ist die Struktur von Vaadin nicht so stark an Swing orientiert.
Weitere Gemeinsamkeiten und Unterschiede sind Abbildung 2 zu entnehmen. Die Unterschiede zwischen den Frameworks in Bezug auf die Schwerpunkte bezüglich Profil C/S-GUIs sind nicht sehr groß. Das zeigt, dass Vaadin ebenfalls die nötigen Features mit sich bringt, die zur Implementation eines Web-Channels für Profil C/S notwendig sind.
Bezüglich die Architekturen der Frameworks ist ebenfalls ein großer Unterschied zu finden. Diese unterscheiden sich dahingehend, dass bei wingS[6] die Kommunikation zwischen Client und Server ausschließlich über eine Java Servlet API abgewickelt wird. Vaadin[7] hingegen kann diese Kommunikation auch über einen Portlet-Container abwickeln [Mar13][53-56] [Sch08].

4. Fazit

Dass wingS abgelöst werden muss, steht aufgrund der Stagnation und des mangelnden Supports fest. Das Vaadin-Framework eignet sich als Ersatz, aufgrund der entsprechenden Standardkomponenten, sowie dem Support der an vielen Stellen in Anspruch genommen werden kann. Zur Not kann man auch Unterstützung von den Entwicklern von Vaadin in Anspruch nehmen. Auch wenn nur das Vaadin-Framework vorgestellt wurde, gibt es zahlreiche andere Web-Frameworks, die dafür in Frage kommen. Vor allem das GWT, auf dem Vaadin aufsetzt ist ebenfalls ein Kandidat dafür.
Da mit Abschluss dieses Praxisberichtes jeweils ein neues Framework für den Desktop- und den Web-Client für Profil C/S evaluiert wurde, muss festgelegt werden welche Frameworks für die neuen Channels eingesetzt werden sollen. Erst dann kann eine konkrete Umsetzung dieser erfolgen.
Dabei werden sich die Entwickler an den bestehenden Channels für Swing und wingS orientieren müssen. Eine Umstrukturierung des Multichannels ist meiner Meinung nach keine Option, da die Ressourcen in deg derzeit für solche Umsetzungen zu knapp sind.
Jedoch könnte man dabei ein einheitliches *Look and Feel* umsetzen, da mit JavaFX auch der Desktop-Client mittels Css gestaltet werden kann.

[6] siehe Abbildung 4
[7] siehe Abbildung 5

Anhänge

A. Literaturverzeichnis

[Ame11] America, Oracle: *Java Virtual Machine Specification - Chapter 5. Loading, Linking, and Initializing.* URL: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-5.htmljvms-5.5, Juli 2011. [Online, eingesehen 30. März 2014].

[IT] IT Mill Ltd.: *Code: Vaadin GridLayout.* URL: http://grepcode.com/file/repo1.maven.org/maven2/com.vaadin/vaadin/6.6.0/com/vaadin/ui/GridLayout.java. [Online; eingesehen 13. März 2014].

[Jon] Jonathan Nash: *Toolbar.* URL: https://vaadin.com/directoryaddon/toolbar. [Online; eingesehen 12. März 2014].

[Jou] Jouni Koivuviita: *ToolbarWindow.* URL: https://vaadin.com/directoryaddon/toolbarwindow. [Online; eingesehen 12. März 2014].

[Mar13] Marko Grönroos: *Book of Vaadin - Vaadin 7 Edition - 1st Revision.* Vaadin Ltd, 2013. 598 S.

[Mül11] Müller, Florian: *Vaadin - ein Java-Webframework im Visier.* URL: http://jaxenter.de/artikel/Vaadin-ein-Java-Webframework-im-Visier, September 2011. [Online, eingesehen 30. März 2014].

[Ora13a] Oracle: *JavaFx Documentation Home: Deploying JavaFX Applications.* URL: http://docs.oracle.com/javafx/2/deployment/jfxpub-deployment.htm, 2013. [Online; eingesehen 30. August 2013].

[Ora13b] Oracle: *Oracle Deployment Tutorial.* URL: http://docs.oracle.com/javase/tutorial/deployment/jar/intro.html, 2013. [Online; eingesehen 30. August 2013].

[Ora14] Oracle: *Oracle Deployment Tutorial*. URL: http://docs.oracle.com/javase/tutorial/deployment/applet/index.html, 2014. [Online; eingesehen 28. Januar 2013].

[Sch08] Schmid, Benjamin: *Get your wingS back!* Javamagazin, 2–7, Januar 2008.

[wina] *wingS*. URL: http://java-source.net/open-source/web-frameworks/wings. [Online; eingesehen 13. März 2014].

[winb] wings development team: *Code: Wings GridBagLayout*. URL: http://www.java2s.com/Open-Source/Java/Swing-Library/wings3/org/wings/SGridBagLayout.java.htm. [Online; eingesehen 13. März 2014].

[winc] wings development team: *Code: Wings GridBagLayoutCG*. URL: http://www.java2s.com/Open-Source/Java/Swing-Library/wings3/org/wings/plaf/css/GridBagLayoutCG.java.htm. [Online; eingesehen 13. März 2014].

B. Implementierung

- Java-Projekt: VaadinProfilSample (vaadinprofilsample.rar)

C. Abbildungen

Vorteile
J2SE-API ist im vollen Umfang nutzbar
komplexe Anwendungen in Zusammenarbeit mit Servlets und Applikationsservern
Installation lokaler Software ist unnötig
In unterschiedlichen Browsern nutzbar
GUI muss nur einmal mit Hilfe eines Frameworks implementiert werden
Der Client muss (gezwungenermaßen) klein gehalten werden
Nachteile
Java-Plugin wird benötigt (Sicherheitsrisiko)
Java-Script muss aktiviert sein (Sicherheitsrisiko)
Abgleich der Java-Versionen notwendig
lange Initialisierung der JVM
Applet muss komplett heruntergeladen werden

Abbildung 1: Vor- und Nachteile von Java-Applets

	wingS	**Vaadin**
Container	Servlet	Servlet, Portlet
Stylability	Css, Layout-Templates	
		Sass (Erweiterung von CSS3)
Komponenten	unterschiedliche GUI-Elemente, eigene Komponenten	
Render-Technik	HTML, CSS, Java-Script	
Verbreitung	großer Bekanntheitsgrad	
	rückläufig	v.a. durch das GWT

Abbildung 2: Vergleich: wingS - Vaadin

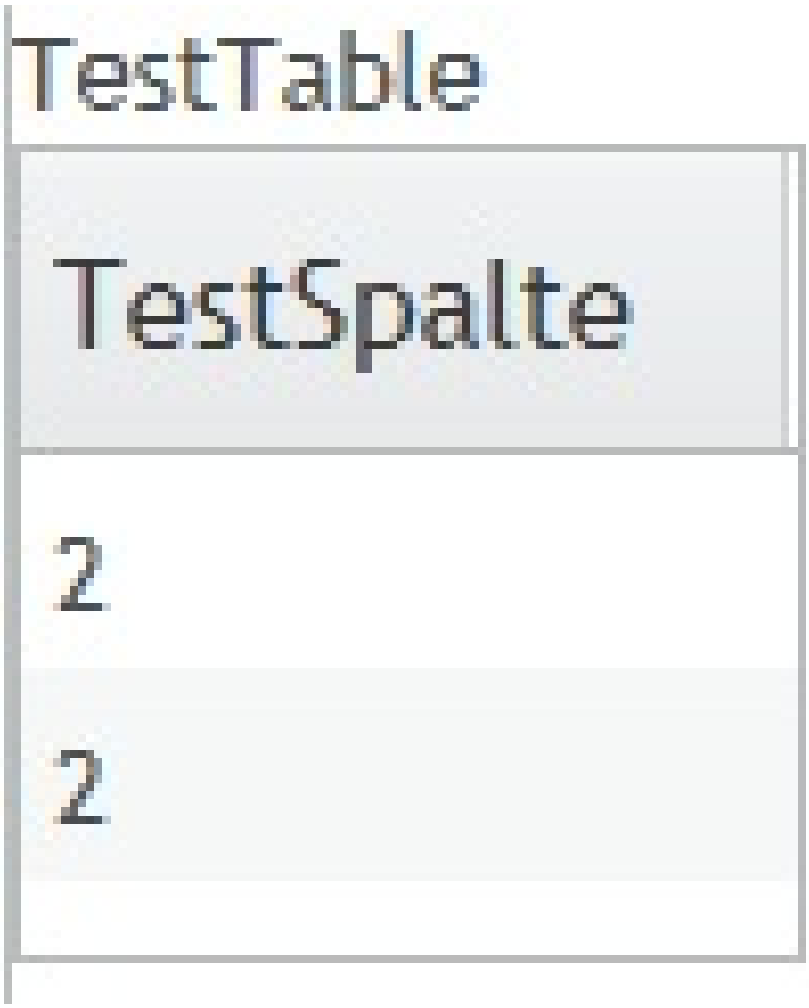

Abbildung 3: TestTable-GUI

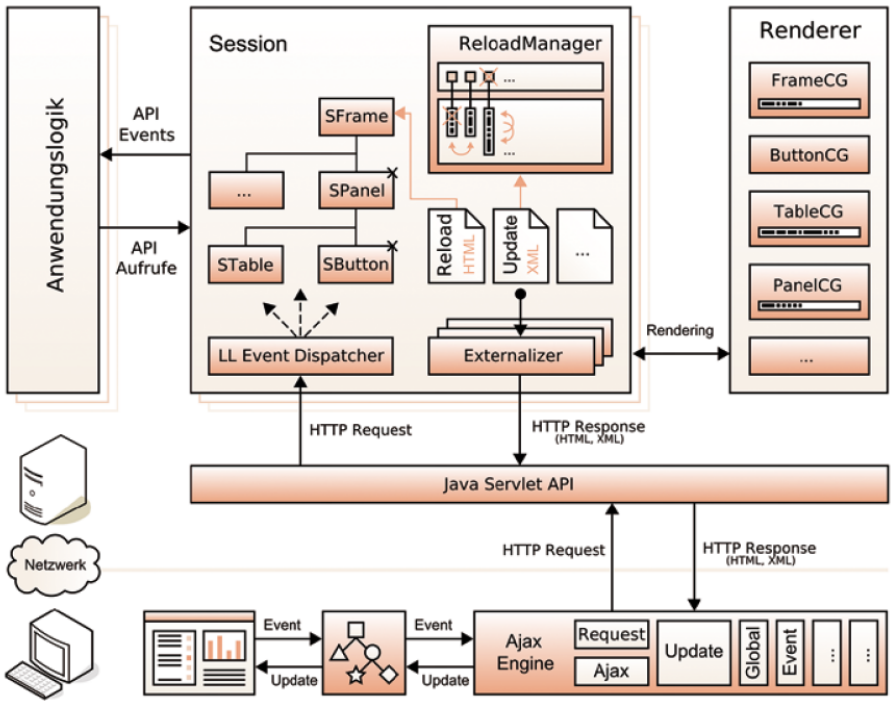

Abbildung 4: wingS-Architektur

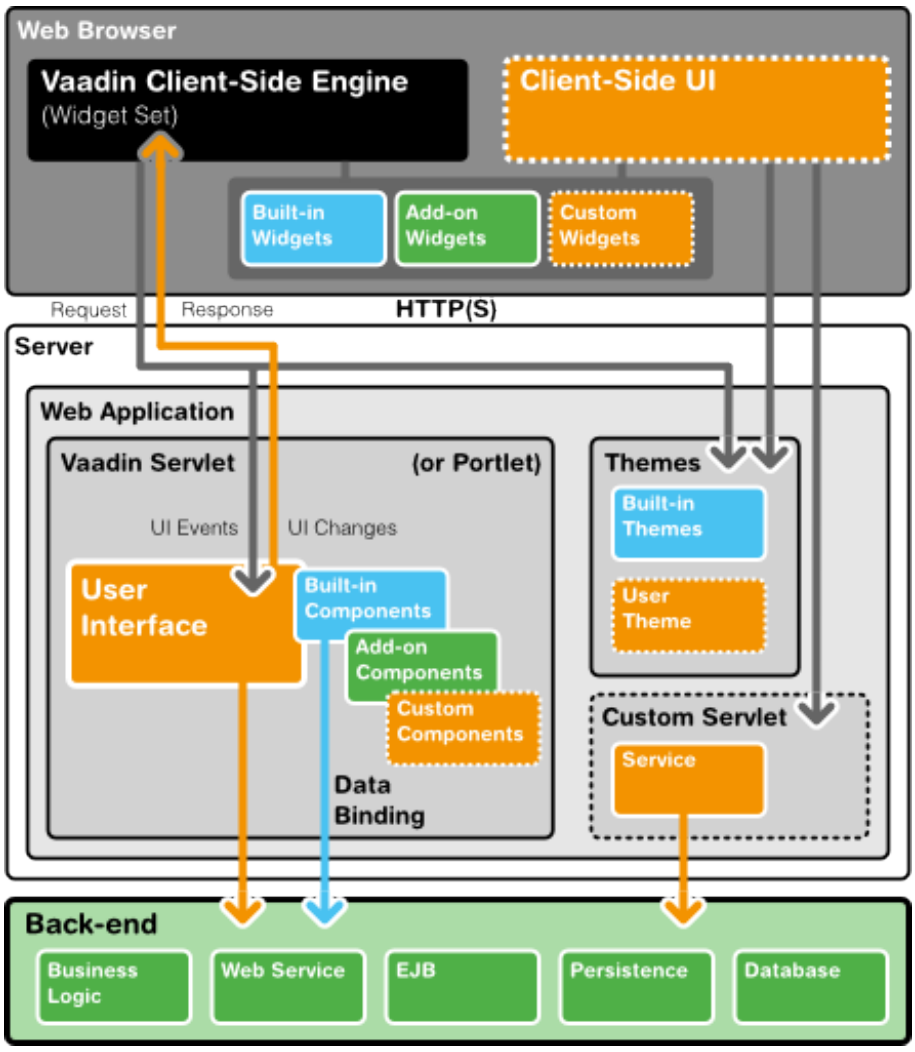

Abbildung 5: Vaadin-Architektur

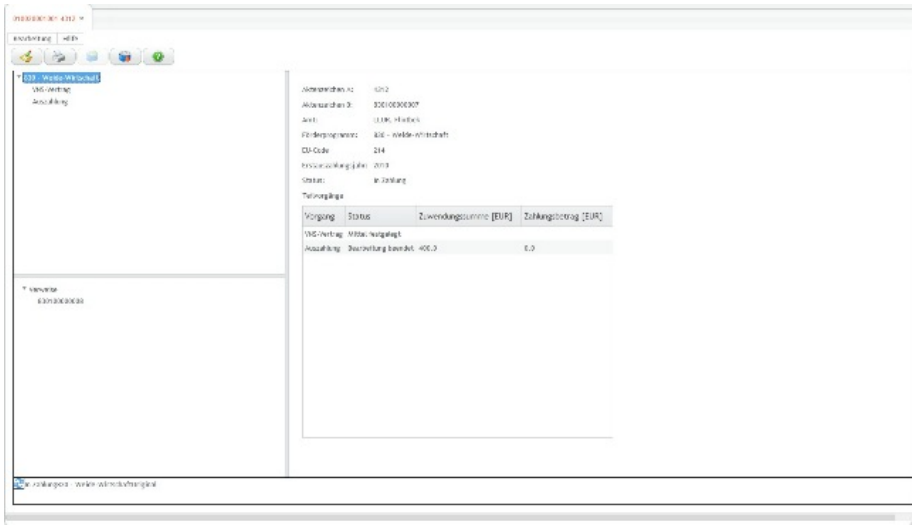

Abbildung 6: Prototyp - Vaadin