



Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

Spring Rich Client Project (Spring RCP)

Michaela Kieneke

Betreuender Dozent: Prof. Dr. Holger Vogelsang

Wintersemester 2008/09

Inhaltsverzeichnis

1	Einführung	1
1.1	Das „Spring Framework“	1
1.2	Das „Spring Rich Client Project“	1
2	Ein „Spring RCP“-Beispiel	3
2.1	Die Entwicklungsumgebung	3
2.2	Mit „Maven“ ein neues „Spring Rich“-Projekt erstellen	5
2.3	Händisch ein neues „Spring Rich“-Projekt erstellen	8
2.3.1	Benötigte Bibliotheken für das „Spring RCP“	8
2.3.2	Eine einfache Anwendung	9
2.3.3	Die „Menubar“	11
2.3.4	Die „Toolbar“	13
2.3.5	Mehrsprachigkeit	14
2.3.6	Eine einfache „View“	16
2.3.7	Eine „View“ mit Datensätzen	18
2.3.8	Neuer Datensatz bzw. Datensatz editieren	20
2.3.9	Validierung der Dateneingaben	22
2.3.10	Mehrere „Views“ - „Docking“	25
2.3.11	Die Oberfläche: „Swing“	26
2.3.12	Die „Icons“	29
2.3.13	Die „Hilfe“	30
2.3.14	Die „ProgressBar“	31
2.4	Fazit	33
3	Ausblick in die Zukunft	34

Abbildungsverzeichnis

1	Eine „Spring Rich“-Anwendung mit „Maven“ erstellt	6
2	Startvorgang mit einem „ProgressSplashScreen“	10
3	„Menubar“ der „Spring Rich“-Anwendung	13
4	„Toolbar“ in der „Spring Rich“-Anwendung	14
5	<i>messages.properties</i> in der „Spring RCP“ Bibliothek	15
6	„Info“ der „Spring Rich“-Anwendung	16
7	Erste „View“ der „Spring Rich“-Anwendung	18
8	Tabellenansicht einer „Spring Rich“-Anwendung mit angepasster Spaltenbreite	20
9	Neuer Datensatz	21
10	Zu editierender Datensatz	21
11	Das „Contextmenu“	22
12	Validierung - Pflichtfelder	23
13	Erweiterung der Validierung um ein „Contextmenu“	24
14	Autovervollständigung in der „Combobox“	24
15	Validierung - Name muss länger als zwei Zeichen sein	24
16	Validierung - Vorname nicht gleich Nachname	25
17	Zwei „Views“ nebeneinander	26
18	Eine „View“ verschieben	27
19	„Views“ hintereinander mit „Tabs“	27
20	Tabellenansicht als „Floating Window“ auf dem Desktop	28
21	Look&Feel „ExperienceRoyale“	28
22	Veränderte „Icons“	30
23	Hilfe für die Anwendung	32
24	ProgressBar-Demo	32

Listings

1	Download der Bibliotheken mit „Maven“	5
2	Erstellung eines neuen „Spring RCPs“ mithilfe von „Maven“	5
3	Projektdateien für Eclipse mithilfe von „Maven“ erstellen	5
4	Bean <i>splashScreen</i>	9
5	Bean <i>application</i>	10
6	Bean <i>applicationDescriptor</i>	11
7	Bean <i>lifecycleAdvisor</i>	11
8	„Properties“ in der Bean <i>lifecycleAdvisor</i>	11
9	Beans für die „Menubar“ in der <i>window-command-bars.xml</i>	12
10	Neues „Property“ in der Bean <i>lifecycleAdvisor</i>	13
11	Bean <i>toolbar</i>	13
12	Bean <i>messageSource</i>	14
13	Informationen über die Anwendung in der <i>messages.properties</i>	15
14	Verwendung der Einträge in der <i>messages.properties</i>	15
15	Neues „Property“ in der Bean <i>lifecycleAdvisor</i>	16
16	Bean <i>initialView</i>	16
17	Beans <i>serviceLocator</i> und <i>applicationServices</i>	17
18	Einträge für die „View“ in der <i>messages.properties</i>	17
19	Bean <i>applicationObjectConfigurer</i>	17
20	Ausgabe in der Konsole	17
21	Bean <i>personView</i>	19
22	Einträge für die „View“ in der <i>messages.properties</i>	19
23	Funktion <i>configureTable(JTable table)</i> in der Klasse <i>PersonTable</i>	19
24	<i>ActionCommandExecutor</i> in der Klasse <i>PersonView</i>	20
25	Neues „Property“ in der Bean <i>lifecycleAdvisor</i>	20
26	<i>ActionCommandExecutor deleteExecutor</i> in der Klasse <i>PersonView</i>	22
27	Beans <i>rulesSource</i> und <i>formComponentInterceptorFactory</i>	22
28	Neue „Properties“ in der Bean <i>personView</i>	25
29	Bean <i>applicationPageFactory</i>	26
30	Aufruf des <i>Look&Feel</i> über die <i>VM arguments</i>	28
31	Bean <i>lookAndFeelConfigurer</i>	28
32	Bean <i>imageResourcesFactory</i>	29
33	Bean <i>imageSource</i>	29
34	Bilder-Pfade in der <i>images.properties</i>	30
35	<i>simple.hs</i>	30
36	HTML-Seiten in der <i>simple.jhm</i>	31
37	Beans <i>helpMenu</i> und <i>helpContentsCommand</i>	31
38	Bean <i>demoCommand</i> in der <i>window-command-bars.xml</i>	32

1 Einführung

Das „Spring Rich Client Project“ („Spring RCP“) basiert auf dem bekannten „Spring Framework“. Mithilfe des „Spring RCPs“ können plattformübergreifende Java-Anwendungen entwickelt werden, die direkt auf dem Desktop laufen.

1.1 Das „Spring Framework“

Das Anfang 2004 erschienene Open Source Framework bietet die Basis zur Entwicklung einer übersichtlichen Programm-Architektur und hat sich bei vielen Java-Entwicklern etabliert. Das „Spring Framework“ arbeitet mit „Plain Old Java Objects“, so genannten „POJOs“, einfachen Java-Objekten ohne Konvention. Ein „POJO“ darf die Konventionen einer „Bean“ erfüllen, muss es aber nicht. So ermöglicht das „Spring Framework“ die Erstellung von Projekten mit dem vollen Funktionsumfang wie er unter der Verwendung von „Enterprise Java Beans“ („EJBs“) in einer „J2EE“-Umgebung bekannt ist. Es muss jedoch nur das implementiert werden, was wirklich benötigt wird.

Eine der Stärken von Java besteht darin, dass bereits eine große Anzahl verschiedener Bibliotheken existieren, die die Entwicklung enorm vereinfachen. Das „Spring Framework“ liefert dazu eine vereinfachte und einheitliche API-Schicht zur Verwendung dieser Java-SE-APIs, Java-EE-APIs und anderen Open Source Frameworks.

Mit „Dependency Injection“ bietet das „Spring Framework“ außerdem einen einheitlichen Weg, Abhängigkeiten von Objekten aufzubauen. Dies geschieht in einer XML-Datei, wodurch kein selbständiges Holen der Ressourcen durch den Quelltext nötig ist. So entsteht eine Unabhängigkeit von der Umgebung und eine daraus resultierende höhere Wiederverwendbarkeit. Mehr dazu in der Beschreibung zur Erstellung eines „Spring Rich“-Projekts in Kapitel 2.3.2 auf Seite 9.

Die „Spring Aspektorientierte Programmierung“ („Spring AOP“) ermöglicht, dass der Java Code nichts von Sicherheit wissen muss. Viel mehr wird diese über Aspekte realisiert, die in einer XML-Konfigurationsdatei deklariert wurden. So muss nicht eine einzige Zeile Java Code dafür geändert werden.

1.2 Das „Spring Rich Client Project“

Das „Spring Rich Client Projekt“ zeigt die Flexibilität und Wiederverwendbarkeit von Desktop-Anwendungen, die auf dem „Spring Framework“ basieren. Eine einmal erstellte Applikation kann für weitere Projekte als Basis verwendet werden. Zur Erstellung der Oberfläche wird die „Swing“-Bibliothek verwendet. Hierbei bietet das „Spring RCP“ Komponenten, einzelne Elemente wie z.B. „Menu-“ und „Toolbar“, „Wizards“, „Views“ oder Prioritäten bzw. Abhängigkeiten mit vergleichsweise wenigen Codezeilen zu realisieren. Die Oberflächenprogrammierung

1 EINFÜHRUNG

und das Applikationsmanagement werden somit vereinfacht. Darüber hinaus bietet es ergänzende Klassen zur Datenbindung und -validierung. Zur Datenbankanbindung können Komponenten des „Spring Frameworks“ verwendet werden¹.

Zur Ausarbeitung lagen leider sehr wenig Informationen vor. In erster Linie wurde für die Projektgrundlagen mit „Programmieren in Java“ [PiJ] und der User Documentation der „Spring RCP“ [SRCP] gearbeitet. Für weitere Einblicke in die Möglichkeiten des „Spring RCPs“ wurde „Getting Started with Spring RCP“ [GSwSRCP] verwendet.

¹Es konnte keine Information gefunden werden, ob das „Spring RCP“ auch hier neue Möglichkeiten bietet.

2 Ein „Spring RCP“-Beispiel

Zur Erstellung eines „Spring Rich“-Projekts wurde zuerst die Verwendung von „Maven“ getestet. Auf den ersten Blick erscheint dieses auf Java basierende Build-Management-Tool der „Apache Software Foundation“ zu Beginn viel Kleinarbeit abzunehmen, jedoch ist bei näherer Betrachtung die händische Erstellung eines Projekts übersichtlicher und verständlicher.

2.1 Die Entwicklungsumgebung

Für das Projekt wurde „Eclipse SDK“², Version: 3.3.0, Build id: I20070625-1500 als Entwicklungsumgebung verwendet. Darüber hinaus war die „Java Runtime Environment“³ („JRE“) in der Version 6 installiert.

Um ein Projekt basierend auf das „Spring Framework“ zu erstellen, bietet es sich als Erstes an, das entsprechende Eclipse-Plugin „Spring IDE“⁴ zu installieren, welches z.B. die Erstellung der XML-Konfigurationsdateien eines „Spring“-Projekts enorm vereinfacht und aktuell in der Version 2.2 vorliegt.

Des Weiteren werden eine Reihe von Bibliotheken benötigt. An dieser Stelle gibt es die Möglichkeit, „Maven“⁵ zur Erstellung eines einfachen Projekts zu verwenden. Über einfache Eingaben in der Konsole wird ein komplettes Grundgerüst für eine „Spring RCP“-Anwendung aufgebaut, welchem gleich alle Bibliotheken auf dem Rechner zur Verfügung stehen. Der Nachteil eines auf diesem Wege erstellten Projekts besteht darin, dass alle benötigten Jar-Dateien nur lokal auf dem Rechner gespeichert sind und nicht mit dem Projekt weitergegeben werden können, da dieser Zugriff nur über den *Build Path* entsprechend konfiguriert wird.

Bei der Erstellung eines Projekts mithilfe von „Maven“ werden viele Bibliotheken mit eingebunden, die evtl. gar nicht benötigt werden. Für ein übersichtlicheres Projekt empfiehlt es sich daher, das Grundgerüst selbst von Hand aufzubauen. Hierbei besteht zudem die Möglichkeit, das Zusammenspiel der einzelnen Komponenten zu verstehen und zu nutzen. Die benötigten Bibliotheken können auf der „Spring richclient“-Seite⁶ bzw. der dazugehörigen „SourceForge“-Seite⁷ heruntergeladen werden. Da das „Spring RCP“ auf dem „Spring Framework“ basiert, wird die entsprechende Bibliothek *spring.jar*⁸ benötigt.

²The Eclipse Foundation: „eclipse“, <http://www.eclipse.org/>

³Sun Microsystems, Inc.: „Java Download“, <http://www.java.com/de/>

⁴Christian Dupuis: „Spring IDE“, <http://springide.org/blog/> [Stand 15.10.2008]

⁵The Apache Software Foundation: „Apache Maven Project“, <http://maven.apache.org/index.html>

⁶The Spring Framework: „Spring richclient“,
<http://spring-rich-c.sourceforge.net/1.0.0/index.html>

⁷SourceForge.net: „Spring rich client“,
http://sourceforge.net/project/showfiles.php?group_id=113660

⁸SourceForge.net: „Spring Framework“,
<http://sourceforge.net/projects/springframework/>

2 EIN „SPRING RCP“-BEISPIEL

Darüber hinaus wurde für Log-Ausgaben die Bibliothek *commons-logging.jar*⁹ eingebunden.

⁹The Apache Software Foundation: „Apache Commons“, <http://commons.apache.org/logging/>

2.2 Mit „Maven“ ein neues „Spring Rich“-Projekt erstellen

Nachdem „Maven“ installiert wurde, werden die gepackten „Spring RCP“-Dateien entpackt. Mit der Konsole wird in den Ordner „spring-richclient-full“ gewechselt und folgende Eingabe gemacht:

```
maven - mvn install -DdownloadSources=true
```

Listing 1: Download der Bibliotheken mit „Maven“

Die enthaltene *pom.xml* enthält die Angaben zum Herunterladen aller benötigten Bibliotheken inklusive Sourcecode. Darüber hinaus wird der Sourcecode für die „Spring RCP“ kompiliert. Danach wird in einen Ordner gewechselt, in dem das neue „Spring Rich“-Projekt erstellt werden soll. Dort wird folgende Eingabe (in einer Zeile) gemacht:

```
mvn archetype:create -DarchetypeGroupId=org.springframework.  
    richclient  
    -DarchetypeArtifactId=spring-richclient-archetype  
    -DarchetypeVersion=1.0.2-SNAPSHOT  
    -DgroupId=de.hska.spring  
    -DartifactId=SpringRCP
```

Listing 2: Erstellung eines neuen „Spring RCPs“ mithilfe von „Maven“

Hierbei beschreibt die ersten beiden Eingaben, dass es sich um eine Anwendung der „Spring RCP“ handelt. Die *DarchetypeVersion* muss entsprechend der installierten „Spring RCP“-Version angepasst werden. Die *DgroupId* bestimmt die Package-Hierarchie des Projekts, die *DartifactId* den Projektnamen.

Wurde das Projekt fehlerfrei erstellt, muss nun in den entsprechenden Ordner gewechselt werden. Im Beispiel heißt er „SpringRCP“. Hier müssen noch zwei weitere Eingaben getätigt werden:

```
mvn install  
  
mvn eclipse:eclipse -DdownloadSources=true
```

Listing 3: Projektdateien für Eclipse mithilfe von „Maven“ erstellen

Auf diese Weise wurde ein Projekt erstellt, was über die „Import“-Funktion von Eclipse problemlos in die Entwicklungsumgebung eingefügt werden kann. Lediglich der Pfad zu den verwendeten Bibliotheken, die sich unter „C:/Dokumente und Einstellungen/Username/.m2/repository“ befinden, muss noch angepasst werden. Hierzu kann bei Eclipse eine Variable mit dem Namen „M2_REPO“ angelegt werden. Alle nachfolgenden mit „Maven“ erstellten Projekte finden so eigenständig ihre benötigten Bibliotheken auf dem Rechner.

Über die Klasse `SimpleApp` kann die Anwendung gestartet werden (siehe Abbildung 1 auf Seite 6). Sie gibt einen ersten Einblick, was das „Spring RCP“ zu bieten hat: eine Anwendung, welche

2 EIN „SPRING RCP“-BEISPIEL

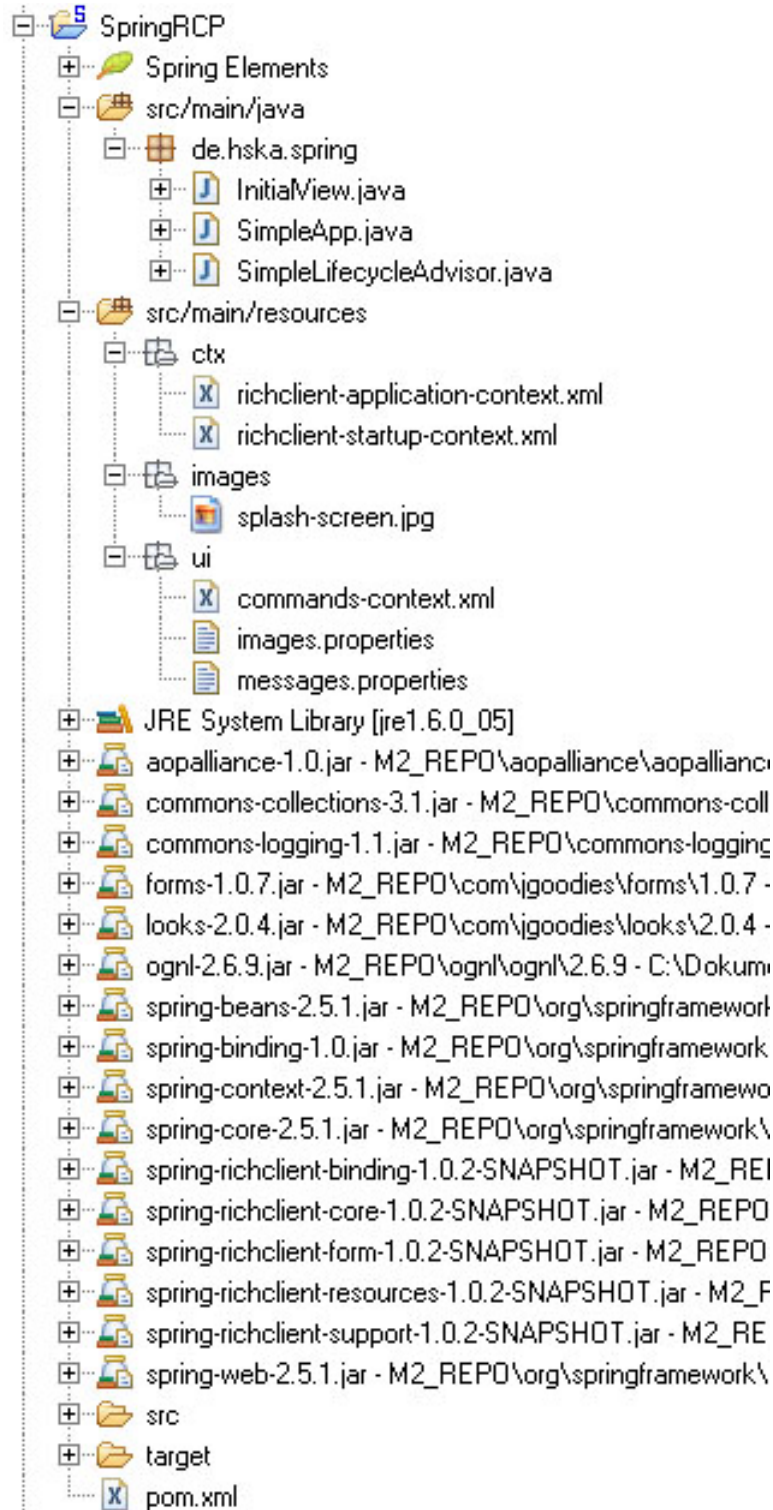


Abbildung 1: Eine „Spring Rich“-Anwendung mit „Maven“ erstellt

2 EIN „SPRING RCP“-BEISPIEL

mit einem netten Bildchen startet und eine erste View zeigt. Automatisch sind bereits „Menu-“ und „Toolbar“ mit ersten Möglichkeiten zur Ausführung eingefügt. Bei näherer Betrachtung findet man eine Datei für Mehrsprachigkeit und eine für die Zuweisung von Icons. Netterweise sind die zur Konfiguration verwendeten XML-Dateien recht ausführlich kommentiert, womit jedoch das Weiterarbeiten am Projekt nicht unbedingt vereinfacht wird.

Neben dem Nachteil, dass der Entwickler bei der Erstellung eines Projekts auf diese Weise keinen Einblick in den Grundaufbau eines „Spring Rich“-Projekts bekommt, ist es nicht gerade wünschenswert, alle zu „Spring“ zu findenden Bibliotheken auf den Rechner geladen zu bekommen. Des Weiteren enthalten einige Jar-Dateien, auf die verwiesen wird, folgende Meldung:

„This file is here because if you build a jnlp with the maven-webstart-plugin, this empty jar will be included in the jnlp. That works for javaws 5, but not for javaws 6, which horribly fails. I’ve posted a bug issue at sun, and provided a proof of concept, they are looking into it.“
empty-form-module-placeholder.txt

Dies kann bei der Ausführung der Anwendung zu Fehlern führen, die dadurch behoben werden können, die entsprechenden Bibliotheken nachträglich noch komplett einzubinden.

Eine Projekt-Erstellung mit „Maven“ wurde ab dieser Stelle nicht weiter betrachtet. Interessanter und lehrreicher erscheint die Erstellung der einzelnen Dateien von Hand, da hierbei der Aufbau und die Funktion genau beobachtet werden kann, wie sie im Folgenden beschrieben wird.

2.3 Händisch ein neues „Spring Rich“-Projekt erstellen

Eine „Spring Rich“-Anwendung besteht zum Einen aus Java Quellcode, zum Anderen werden XML-Dateien verwendet. Sie vereinfachen z.B. die Konfiguration, die sonst mühsam in Java realisiert werden musste. Zudem bietet das „Spring RCP“ Komponenten, die die einheitliche Erstellung einer Anwendung enorm vereinfachen. Dies soll an dem folgenden Projekt verdeutlicht werden.

2.3.1 Benötigte Bibliotheken für das „Spring RCP“

spring.jar Die *spring.jar* beinhaltet Basiskomponenten des „Spring Frameworks“ - siehe Kapitel 2.3.2 auf Seite 9.

spring-richclient-full.jar Die *spring-richclient-full.jar* beinhaltet einen großen Teil der Komponenten des „Spring RCPs“ - siehe Kapitel 2.3.2 auf Seite 9.

commons-logging.jar Die *commons-logging.jar* ermöglicht Log-Ausgaben während des Ausführens einer Anwendung - benötigt in Kapitel 2.3.2 auf Seite 9.

spring-binding-1.0.jar Die *spring-binding-1.0.jar* enthält Komponenten, die zum Starten einer „Spring Rich“-Anwendung benötigt werden, die jedoch nicht in der Standard-Bibliothek enthalten sind - siehe Kapitel 2.3.2 auf Seite 11.

forms-1.0.7.jar Die *forms-1.0.7.jar* beinhaltet Komponenten zur Anzeige von Dialogen - siehe Kapitel 2.3.3 auf Seite 12.

spring-richclient-sandbox-1.0.0.jar Die *spring-richclient-sandbox-1.0.0.jar* enthält Komponenten zur Darstellung von Tabellen - siehe Kapitel 2.3.7 auf Seite 18.

glazedlists_java15-1.7.0.jar Die *glazedlists_java15-1.7.0.jar* enthält Komponenten für den Zugriff auf Tabellen - siehe Kapitel 2.3.7 auf Seite 18.

commons-collections-3.2.1.jar Die *commons-collections-3.2.1.jar* wird benötigt, wenn bei einem Datensatz eine „ComboBox“ zur Dateneingabe verwendet wird - siehe Kapitel 2.3.8 auf Seite 21.

vldocking-2.1.4.jar Die *vldocking-2.1.4.jar* enthält „Swing“-Komponenten für das „Docking“ - siehe Kapitel 2.3.10 auf Seite 26.

spring-richclient-docking-1.0.0.jar Die *spring-richclient-docking-1.0.0.jar* enthält „Spring RCP“-Komponenten für das „Docking“ - siehe Kapitel 2.3.10 auf Seite 26.

jhhelp-2.0.jar Die *jhhelp-2.0.jar* wird benötigt, um eine Hilfe in die Anwendung einzubauen - siehe Kapitel 2.3.13 auf Seite 30.

2.3.2 Eine einfache Anwendung

Die Klasse `HskaApplication` ist sozusagen das „Herzstück“ der Anwendung. Sie startet die „Spring Rich“-Anwendung und lädt die `richclient-startup-definition.xml` und die `richclient-application-definition.xml`, welche die „Beans“ beinhalten, die die Anwendung konfigurieren. Für Log-Ausgaben wird die Bibliothek `commons-logging.jar` benötigt. Darüber hinaus müssen die Bibliotheken `spring.jar` und `spring-richclient-full.jar` eingebunden werden, welche die Basis-Komponenten des „Spring Frameworks“ bzw. des „Spring RCPs“ beinhalten.

Wie schon in Kapitel 1.1 auf Seite 1 erwähnt, verwendet das „Spring Framework“ so genannte „POJOs“, „einfache, alte Java Objekte“, die zwar die Konventionen einer „Bean“ nicht erfüllen müssen, es jedoch *dürfen*. Aus historischen Gründen werden sie trotzdem „Beans“ genannt¹⁰, was auch im Folgenden der Fall sein wird.

Die Definition jeder Bean wird im „Spring Framework“ in einer XML-Datei vorgenommen. Zu jeder Bean muss eine entsprechende Klasse existieren, die alle Schnittstellen beinhaltet, welche über die XML-Datei angesprochen werden. „Spring“ erzeugt von jeder dieser Klassen ein Objekt und von diesem eine Instanz. Um auf die Instanzen über eine „Factory“ zugreifen zu können oder um die definierten Beans zu referenzieren, ist für jede Bean eine eindeutige ID nötig.

Die `richclient-startup-definition.xml` beinhaltet die Konfiguration zum Start der Anwendung. Hierbei kann z.B. zwischen einem `SimpleSplashScreen` und einem `ProgressSplashScreen` mit Fortschrittsbalken gewählt werden, wobei je nach Auswahl noch bestimmte Eigenschaften übergeben werden müssen.

```
1 <bean id="splashScreen"  
2   class="org.springframework.richclient.application.splash.  
   ProgressSplashScreen"  
3   scope="prototype">  
4   <property name="imageResourcePath"  
5       value="/images/splash-screen.jpg" />  
6   <property name="showProgressLabel"  
7       value="true" />  
8 </bean>
```

Listing 4: Bean `splashScreen`

Über diese Definition der Bean `splashScreen` wird eine Instanz der Klasse `ProgressSplashScreen` erzeugt. Die „Dependency Injection“ besagt, dass ein Objekt, welches mit anderen arbeitet, dieses injiziert bekommt. Zur Injektion von Eigenschaften gibt es bei Beans zwei Möglichkeiten: die „Konstruktor-Injektion“ und die „Setter-Injektion“. In diesem Fall wird die „Setter-Injektion“ verwendet. In der Bean-Klasse müssen `get`-Methoden vorhanden sein, wenn später auf Werte zugegriffen werden soll. Für die „Setter-Injektion“ werden zusätzlich `set`-Methoden zum Setzen

¹⁰siehe [Spr2] S. 12

2 EIN „SPRING RCP“-BEISPIEL

der übergebenen Werte benötigt. Die Bean wird dazu um „Properties“ ergänzt, welche jeweils die Attribute „name“ und „value“ besitzen. Der Klasse `ProgressSplashScreen` besitzt (unter anderem) die Eigenschaft `imageResourcePath`, welche einen Pfad zu einem Bild als Wert erwartet, und die Eigenschaft `showProgressLabel`, welche auf `true` oder `false` gesetzt werden kann.

Standardmäßig ist jede Bean im „Spring Framework“ ein „Singleton“. Soll jedes Mal, wenn eine injizierte Bean angefordert wird, eine neue Instanz erzeugt werden, muss als Geltungsbereich `prototype` verwendet werden.

Dieser `ProgressSplashScreen` zeigt während des Ladevorgangs das Bild `splash-screen.jpg` und verdeutlicht den Ladestatus mit einem Balken, in dem die gerade bearbeiteten Dateien eingeblendet werden.



Abbildung 2: Startvorgang mit einem „ProgressSplashScreen“

Für das erste Laden einer Anwendung sind in der `richclient-application-definition.xml` lediglich die Beans `application`, `applicationDescriptor` und `lifecycleAdvisor` nötig.

In der Bean `application` wird die oben schon erwähnte „Konstruktor-Injektion“ verwendet. Sie übergibt zwei Argumente an den Konstruktor der Klasse `Application`, wobei es sich in diesem Fall nicht um Werte („values“) sondern um referenzierte Beans innerhalb der XML-Datei handelt.

```
1 <bean id="application"
2   class="org.springframework.richclient.application.Application">
3   <constructor-arg index="0" ref="applicationDescriptor" />
4   <constructor-arg index="1" ref="lifecycleAdvisor" />
5 </bean>
```

Listing 5: Bean `application`

2 EIN „SPRING RCP“-BEISPIEL

Der Konstruktor der Klasse `Application` erwartet zwei Übergaben, deren Reihenfolge über das Attribut „`index`“ festgelegt ist.

Die Bean `applicationDescriptor` beinhaltet Informationen zur Anwendung wie z.B. Version, Autor und Titel, welche wieder über „`Properties`“ übergeben werden. Zur Ausführung der Klasse `DefaultApplicationDescriptor` wird eine weitere Bibliothek benötigt, die ebenfalls in den `BuildPath` der Anwendung eingebunden werden muss: `spring-binding-1.0.jar`.

```
1 <bean id="applicationDescriptor"
2   class="org.springframework.richclient.application.support.
3     DefaultApplicationDescriptor">
4   <property name="version" value="1.0" />
5   <property name="buildId" value="20081024-001" />
6   <property name="title"
7     value="Beispielanwendung für das SpringRCP" />
8 </bean>
```

Listing 6: Bean `applicationDescriptor`

Die Klasse `HskaLifecycleAdvisor` beinhaltet Zustände, die während der verschiedenen Lebenszyklen der Anwendung angenommen werden können wie z.B. das Starten und das Schließen der Applikation und was in diesem Status ausgeführt werden soll. Außerdem werden hier später über „`Properties`“ Bestandteile der Anwendung geladen wie „`Views`“, „`Menu-/Toolbar`“ u.ä. Darüber hinaus wird die Klasse dafür verwendet, Log-Informationen über den aktuellen Status der Anwendung auszugeben.

```
1 <bean id="lifecycleAdvisor"
2   class="de.hska.application.HskaLifecycleAdvisor" />
```

Listing 7: Bean `lifecycleAdvisor`

Beim Ausführen der Klasse `HskaApplication` wird ein `ApplicationLauncher` mit beiden XML-Dateien ausgeführt. Es öffnet sich nun eine Anwendung, welche lediglich einen Titel und eine „`StatusBar`“ enthält aber darüber hinaus über keine weiteren Funktionen verfügt.

2.3.3 Die „Menubar“

Um der Anwendung eine „`Menubar`“ zuzufügen, muss die Bean `lifecycleAdvisor` in der `richclient-application-definition.xml` um zwei „`Properties`“ ergänzt werden.

```
1 <property name="windowCommandBarDefinitions"
2   value="ui/window-command-bars.xml" />
3 <property name="menubarBeanName"
4   value="menuBar" />
```

Listing 8: „`Properties`“ in der Bean `lifecycleAdvisor`

2 EIN „SPRING RCP“-BEISPIEL

Die Eigenschaft `windowCommandBarDefinitions` erwartet einen Pfad zu einer XML-Datei, welche Beans zur Konfiguration der „Menubar“ enthält. Über die Eigenschaft `menubarBeanName` wird der Name der Bar bekannt gemacht, wobei es sich in diesem Fall auch gleich um die Id der entsprechenden Bean handelt.

Die `window-command-bars.xml`, die noch in das Projekt eingefügt werden muss, enthält neben der Bean für den Aufbau der „Menubar“ und dessen Untermenüs noch weitere Beans, welche später benötigt werden.

```
1 <bean id="menuBar"
2   class="org.springframework.richclient.command.
3     CommandGroupFactoryBean">
4   <property name="members">
5     <list>
6       <ref bean="fileMenu" />
7       <ref bean="editMenu" />
8       <ref bean="windowMenu" />
9       <ref bean="helpMenu" />
10    </list>
11  </property>
12 </bean>
13 <bean id="fileMenu"
14   class="org.springframework.richclient.command.
15     CommandGroupFactoryBean">
16   <property name="members">
17     <list>
18       <value>newCommand</value>
19       <value>separator</value>
20       <value>propertiesCommand</value>
21       <value>separator</value>
22       <bean class="org.springframework.richclient.command.support.
23         ExitCommand" />
24     </list>
25   </property>
26 </bean>
27 [...]
```

Listing 9: Beans für die „Menubar“ in der `window-command-bars.xml`

In dieser XML-Datei werden der Aufbau und die Sortierung bzw. die Navigationsmöglichkeiten der Anwendung gesteuert. Es ist zu sehen, dass der Eigenschaft `members` der Klasse `CommandGroupFactoryBean` nicht nur ein Wert sondern auch mehrere Werte in Form einer Liste übergeben werden können. Hierbei kann es sich z.B. um eine Referenz auf eine andere Bean, um ein Kommando, einen Separator oder auch eine Klasse handeln.

2 EIN „SPRING RCP“-BEISPIEL

Bevor die „Menubar“ und insbesondere die „Info“ getestet werden können, muss noch die Bibliothek *forms-1.0.7.jar* eingebunden werden. Das „Spring RCP“ greift hier auf „jgoodies“-Layout-Darstellungen zurück, die nicht in der Basis-Bibliothek vorhanden sind.

Der „Info“-Dialog lässt sich nun öffnen, leider fehlt jedoch noch die Internationalisierung, die in Kapitel 2.3.5 auf Seite 14 folgt. Die „Menubar“ dagegen hat die Spracheinstellungen des Rechners erkannt und die Ausgabe bereits entsprechend angepasst.

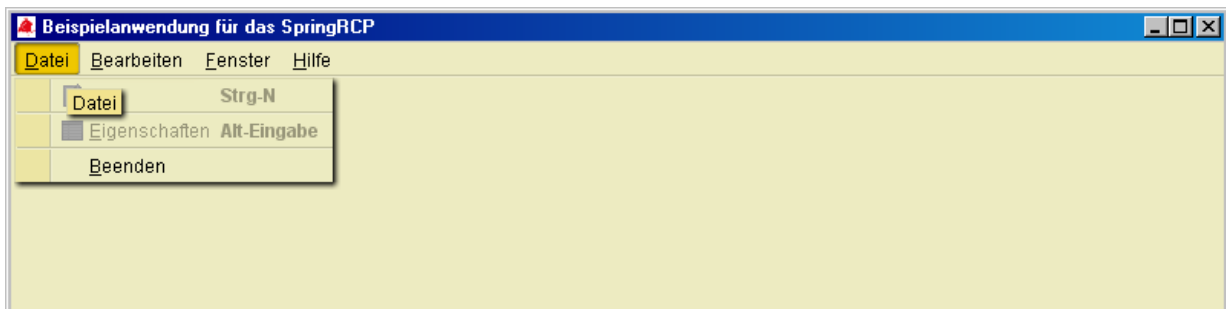


Abbildung 3: „Menubar“ der „Spring Rich“-Anwendung

Sehr schön ist auch die automatische Vergabe von „Tooltips“, „Mnemonics“ und „Shortcuts“, die das „Spring RCP“ übernimmt, ohne dass der Entwickler eingreifen müsste. Für die Navigation zu den „Views“ besteht später die Möglichkeit, in der Datei für die Internationalisierung entsprechende „Mnemonics“ und „Shortcuts“ einzufügen (siehe Kapitel 2.3.7 auf Seite 19).

2.3.4 Die „Toolbar“

Für die Anzeige einer „Toolbar“ muss die Bean `lifecycleAdvisor` wieder um ein „Property“ erweitert werden, um `toolbarBeanName`.

```
1 <property name="toolbarBeanName" value="toolBar" />
```

Listing 10: Neues „Property“ in der Bean `lifecycleAdvisor`

Die referenzierte Bean `toolBar` befindet sich ebenfalls in der `window-command-bars.xml`:

```
1 <bean id="toolBar"
2   class="org.springframework.richclient.command.
3     CommandGroupFactoryBean">
4   <property name="members">
5     <list>
6       <value>newCommand</value>
7       <value>separator</value>
8       <value>propertiesCommand</value>
9       <value>deleteCommand</value>
```

2 EIN „SPRING RCP“-BEISPIEL

```

9     </list>
10    </property>
11 </bean>

```

Listing 11: Bean *toolbar*

Da die Anwendung noch nicht um die Verarbeitung von Datensätzen ergänzt wurde, sind die „Buttons“ noch nicht ausführbar.

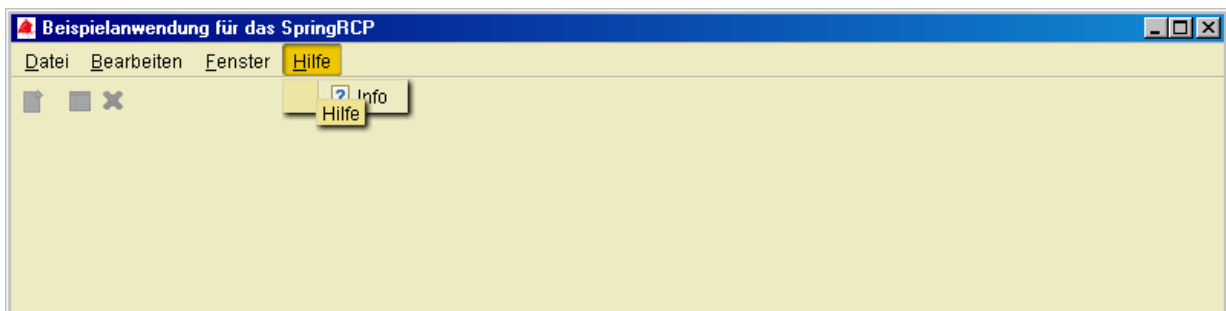


Abbildung 4: „Toolbar“ in der „Spring Rich“-Anwendung

2.3.5 Mehrsprachigkeit

Eigenschaften können im „Spring Framework“ auch mithilfe von „Properties“-Dateien einer Bean injiziert werden. Dies ist sinnvoll, wenn z.B. Text in mehreren Sprachen vorliegen soll.

Die Daten für die Mehrsprachigkeit werden in der Datei *messages.properties* abgelegt. Zu Beginn kann sie erst einmal leer bleiben. Die Id der dazugehörigen Bean in der *richclient-application-definition.xml* muss *messageSource* heißen. Die Property basenames übergibt als ersten Wert das „Package“ in Verbindung mit dem Namen, die Endung wird weggelassen.

```

1 <bean id="messageSource"
2     class="org.springframework.context.support.
3         ResourceBundleMessageSource">
4     <property name="basenames">
5         <list>
6             <value>ui.messages</value>
7             <value>org.springframework.richclient.application.messages</
8                 value>
9         </list>
10    </property>
11 </bean>

```

Listing 12: Bean *messageSource*

2 EIN „SPRING RCP“-BEISPIEL

Des Weiteren erhält die Klasse `ResourceBundleMessageSource` die `messages.properties`, welche schon in der „Spring RCP“ Bibliothek vorhanden ist. Sie enthält Standard-Texte in mehreren Sprachen wie u.a. Deutsch, Englisch und Französisch z.B. für die „Menubar“.



Abbildung 5: `messages.properties` in der „Spring RCP“ Bibliothek

Durch die Definition dieser Bean mit den beiden übergebenen „Properties“-Dateien wird in der „Info“-Ausgabe die Bezeichnung zu „Version“ und „Build ID“ gefunden. Darüber hinaus können in der selbst erstellten `messages.properties` noch weitere Informationen über die Anwendung gespeichert werden, die dann ebenfalls in der Ausgabe erscheinen.

```

1 applicationDescriptor.caption=\u00a9 Michaela Kieneke
2 applicationDescriptor.description=Entstanden im Seminar WS2008/09

```

Listing 13: Informationen über die Anwendung in der `messages.properties`

Mit der Ergänzung dieser beiden Zeilen, sieht die „Info“-Ausgabe schon wie in Abbildung 6.

Neben diesen, vom „Spring RCP“ selbständig eingefügten Texten, können auch eigene Bezeichnungen angelegt werden. Per Javacode kann darauf zugegriffen werden:

```

1 Application.instance().getApplicationContext().getMessage("text.in.
   message.properties", null, null));

```

Listing 14: Verwendung der Einträge in der `messages.properties`

Der Vorteil einer Internationalisierung besteht darin, dass für jede mögliche Sprache, die auf dem ausführenden PC eingestellt sein könnte, eine eigene `messages.properties` mit entsprechender Endung angelegt werden kann, die dann automatisch von der Anwendung geladen und angezeigt wird.

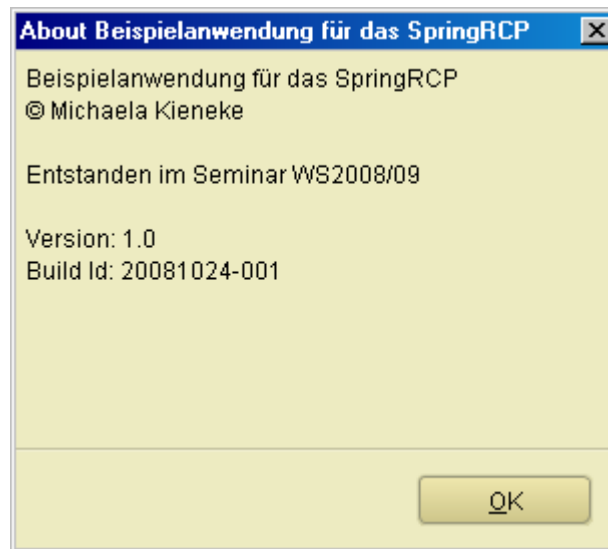


Abbildung 6: „Info“ der „Spring Rich“-Anwendung

2.3.6 Eine einfache „View“

Für eine erste „View“ wird die Klasse `InitialView` erstellt, welche einfache „Swing“-Elemente für eine erste Darstellung enthält. Wieder muss die Bean `lifecycleAdvisor` um ein „Property“ ergänzt werden, die `startingPageId`:

```
1 <property name="startingPageId" value="initialView" />
```

Listing 15: Neues „Property“ in der Bean `lifecycleAdvisor`

Die referenzierte Bean `initialView` wird ebenfalls in der `richclient-application-definition.xml` angelegt.

```
1 <bean id="initialView"
2   class="org.springframework.richclient.application.support.
   DefaultViewDescriptor">
3   <property name="viewClass" value="de.hska.ui.HskaInitialView" />
4 </bean>
```

Listing 16: Bean `initialView`

Mit der Klasse `DefaultViewDescriptor` wird eine einfache „View“ erzeugt. Später wird im Vergleich die Klasse `VLDockingViewDescriptor` verwendet, die z.B. das „Docking“ ermöglicht (siehe Kapitel 2.3.10 auf Seite 26).

Damit die „View“ instanziiert werden kann, müssen noch die Beans `serviceLocator` und `applicationServices` eingefügt werden. In der Klasse `ApplicationServicesLocator` wird die Bean der Klasse `DefaultApplicationServices` referenziert. Beide zusammen sind für die Anwendungslogik zuständig.

2 EIN „SPRING RCP“-BEISPIEL

```
1 <bean id="serviceLocator"  
2   class="org.springframework.richclient.application.  
   ApplicationServicesLocator">  
3   <property name="applicationServices" ref="applicationServices" />  
4 </bean>  
5  
6 <bean id="applicationServices"  
7   class="org.springframework.richclient.application.support.  
   DefaultApplicationServices">  
8 </bean>
```

Listing 17: Beans *serviceLocator* und *applicationServices*

Zur vollständigen Ansicht wird die *messages.properties* um ein „Label“ und die im „Swing“-Code verwendete Meldung erweitert. Der „Titel“ *kann, muss* aber nicht angegeben werden. Der Wert von *initialView.label* wird z.B. verwendet, um die Auswahl über das Menü anzuzeigen, *initialView.title* beinhaltet den Titel, der bei der „View“ angezeigt wird. Fehlt der zweite Wert, wird automatisch der erste verwendet.

```
1 initialView.label=&Startseite des HSKA Rich Clients  
2 #initialView.title=Erste View  
3 initialView.message=Dies ist die erste View!
```

Listing 18: Einträge für die „View“ in der *messages.properties*

Damit die „View“ über die „Menubar“ erreichbar ist, muss zudem noch die Bean *applicationObjectConfigurer* in der *richclient-application-definition.xml* eingefügt werden.

```
1 <bean id="applicationObjectConfigurer" depends-on="serviceLocator"  
2   class="org.springframework.richclient.application.config.  
   DefaultApplicationObjectConfigurer">  
3 </bean>
```

Listing 19: Bean *applicationObjectConfigurer*

Die Klasse *DefaultApplicationObjectConfigurer* erweitert die „Menubar“ unter „Fenster → Ansicht anzeigen“ um den unter „Label“ bzw. „Title“ in der *messages.properties* angegebenen Namen.

In der Konsole sieht man, welche Sprach-Quellen (theoretisch) noch vermisst werden. Dadurch entsteht jedoch kein Fehler und die Anwendung wird trotzdem problemlos ausgeführt.

```
1 [...]
2 13.11.2008 19:24:16 org.springframework.richclient.application.  
   config.DefaultApplicationObjectConfigurer loadMessage
3 INFO: The message source is unable to find message code [  
   initialView.title].Ignoring and returning null.
```

2 EIN „SPRING RCP“-BEISPIEL

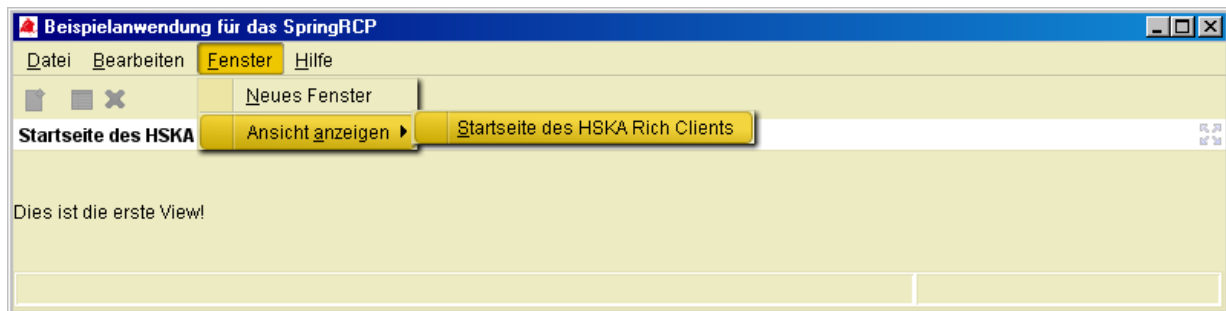


Abbildung 7: Erste „View“ der „Spring Rich“-Anwendung

```

4 13.11.2008 19:24:16 org.springframework.richclient.application.
   config.DefaultApplicationObjectConfigurer loadMessage
5 INFO: The message source is unable to find message code[initialView
   .caption].Ignoring and returning null.
6 13.11.2008 19:24:16 org.springframework.richclient.application.
   config.DefaultApplicationObjectConfigurer loadMessage
7 INFO: The message source is unable to find message code[initialView
   .description].Ignoring and returning null.
8 [...]

```

Listing 20: Ausgabe in der Konsole

2.3.7 Eine „View“ mit Datensätzen

Das „Spring RCP“ bietet die Möglichkeit, mit vergleichsweise geringem Arbeitsaufwand Datensätze in einer Tabelle aufzulisten, neue Datensätze hinzuzufügen bzw. bestehende zu editieren. Die Erstellung der Tabelle und der Dialoge wird dabei weitestgehend von dem „Spring RCP“ übernommen, wodurch eine einheitliche und relativ „ordentliche“ Oberfläche garantiert ist.

Für die Anzeige von Datensätzen muss zunächst eine Klasse `Person` erstellt werden, die die einzelnen Personen instanziiert. Für ein einfaches und dauerhaftes Speichern der Datensätze wird die Datei `persons.data` angelegt. Die Klasse `PersonDataStore` kümmert sich um das Laden der Datei, den Zugriff, das Zufügen, das Löschen und das Speichern der einzelnen Datensätze.

Für die tabellarische Darstellung in der „View“ ist die Klasse `PersonTable` zuständig, welche von der Klasse `AbstractObjectTable` abgeleitet ist. Dazu muss die Bibliothek `spring-richclient-sandbox-1.0.0.jar` eingebunden werden, welche diese Klasse enthält. Darüber hinaus wird die Bibliothek `glazedlists_java15-1.7.0.jar` für den Zugriff auf ein bestimmtes Tabellen-Element benötigt. Der Konstruktor der Klasse `PersonTable` erstellt eine Tabelle mit einer Id und den gewünschten Spalten zur Datenanzeige. Danach wird der Datenspeicher zugewiesen.

2 EIN „SPRING RCP“-BEISPIEL

Die Klasse `PersonView` beinhaltet ähnlich wie die schon bekannte Klasse `InitialView` „Swing“-Elemente und bindet die Klasse `PersonTable` ein. Um sie zu laden, müssen in der `richclient-application-definition.xml` noch die Bean `personView` und referenzierte Bean `personDataStore` ergänzt werden:

```

1 <bean id="personView"
2   class="org.springframework.richclient.application.support.
   DefaultViewDescriptor">
3   <property name="viewClass" value="de.hska.ui.PersonView" />
4   <property name="viewProperties">
5     <map>
6       <entry key="personDataStore" value-ref="personDataStore" />
7     </map>
8   </property>
9 </bean>
10
11 <bean id="personDataStore"
12   class="de.hska.data.PersonDataStore" />

```

Listing 21: Bean `personView`

An dieser Stelle würde sich das Projekt schon fehlerfrei ausführen lassen, jedoch ist noch kein Zugriff auf die neue „View“ möglich. Hierzu muss die `messages.properties` ergänzt werden:

```

1 personView.label=&Übersicht der Mitarbeiter@ctrl M
2 personView.title=Übersicht der Mitarbeiter an der HSKA
3
4 firstName.label=Vorname
5 lastName.label=Nachname
6 room.label=Raum
7 personType.label=Typ
8 consultingHour.label=Sprechstunde

```

Listing 22: Einträge für die „View“ in der `messages.properties`

Der Zusatz `@ctrl M` erstellt den „Shortcut Strg + M“ für die neue „View“. So ist der Zugriff über die „Menubar“, „Mnemo“ oder die entsprechende Tastenkombination möglich. Die weiteren Zeilen sind für die korrekte Anzeige der Spaltennamen.

Soll jede einzelne Spalten eine bestimmte Breite besitzen, kann dies durch Überschreiben der Funktion `configureTable(JTable table)` in der Klasse `PersonTable` erreicht werden:

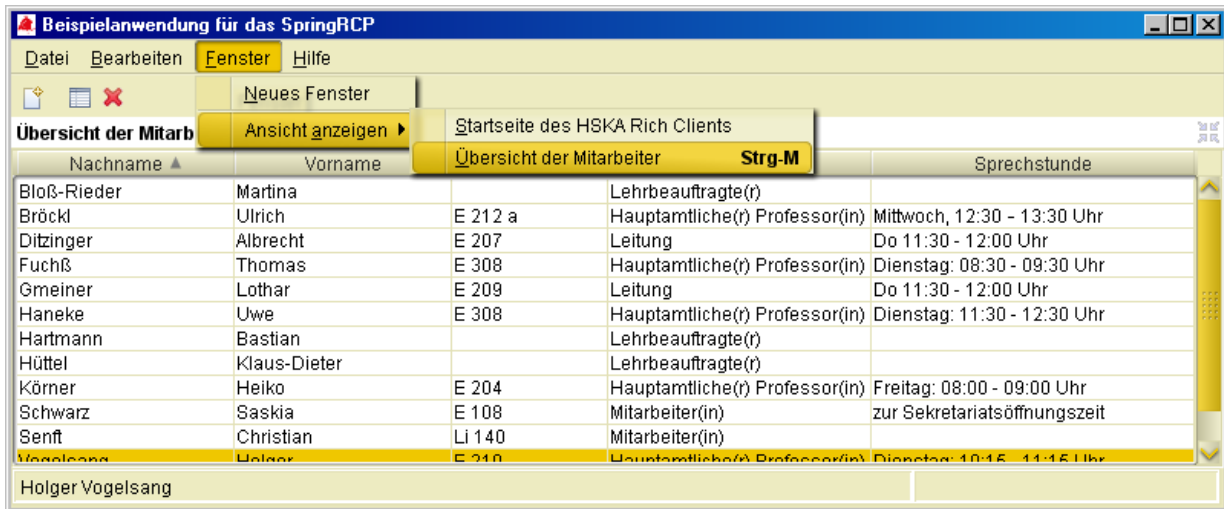
```

1 protected void configureTable(JTable table) {
2   TableColumnModel tcm = table.getColumnModel();
3   tcm.getColumn(0).setPreferredWidth(90);
4   [...]
5 }

```

Listing 23: Funktion `configureTable(JTable table)` in der Klasse `PersonTable`

2 EIN „SPRING RCP“-BEISPIEL



Nachname ▲	Vorname	Übersicht der Mitarbeiter	Strg-M	Sprechstunde
Bloß-Rieder	Martina		Lehrbeauftragte(r)	
Bröckl	Ulrich	E 212 a	Hauptamtliche(r) Professor(in)	Mittwoch, 12:30 - 13:30 Uhr
Ditzinger	Albrecht	E 207	Leitung	Do 11:30 - 12:00 Uhr
Fuchß	Thomas	E 308	Hauptamtliche(r) Professor(in)	Dienstag: 08:30 - 09:30 Uhr
Gmeiner	Lothar	E 209	Leitung	Do 11:30 - 12:00 Uhr
Haneke	Uwe	E 308	Hauptamtliche(r) Professor(in)	Dienstag: 11:30 - 12:30 Uhr
Hartmann	Bastian		Lehrbeauftragte(r)	
Hüttel	Klaus-Dieter		Lehrbeauftragte(r)	
Körner	Heiko	E 204	Hauptamtliche(r) Professor(in)	Freitag: 08:00 - 09:00 Uhr
Schwarz	Saskia	E 108	Mitarbeiter(in)	zur Sekretariatsöffnungszeit
Senft	Christian	Li 140	Mitarbeiter(in)	
Muggelberg	Ulrich	E 210	Hauptamtliche(r) Professor(in)	Dienstag: 10:15 - 11:15 Uhr
Holger Vogelsang				

Abbildung 8: Tabellenansicht einer „Spring Rich“-Anwendung mit angepasster Spaltenbreite

2.3.8 Neuer Datensatz bzw. Datensatz editieren

Für die Erstellung des Dialogs wird die Klasse `PersonForm` angelegt, welche von der Klasse `AbstractForm` abgeleitet ist. Zur Darstellung wird der `TableFormBuilder` verwendet, den das „Spring RCP“ zur Verfügung stellt. Über einfaches Hinzufügen der einzelnen Attribute der Klasse `Person` baut er automatisch einen ordentlichen dargestellten Dialog auf, der jedoch auf Wunsch auch ummodelliert und angepasst werden kann.

Zur Verarbeitung der Daten des Dialogs ist die Klasse `PersonPropertiesDialog` zuständig. In ihr können auch Anpassungen zur Textausgabe im Dialog vorgenommen werden.

Zum Ausführen des neuen Anlegens bzw. Editierens eines Datensatzes müssen noch entsprechende `ActionCommandExecutor` in der Klasse `PersonView` bekannt gemacht werden:

```
1 private ActionCommandExecutor newExecutor = new NewExecutor();
2 private ActionCommandExecutor selectionExecutor = new
   SelectionExecutor();
```

Listing 24: `ActionCommandExecutor` in der Klasse `PersonView`

Darüber hinaus wird die Bean `lifecycleAdvisor` in der `richclient-application-definition.xml` wieder um ein „Property“ erweitert:

```
1 <property name="windowCommandManagerBeanName"
2         value="windowCommandManager" />
```

Listing 25: Neues „Property“ in der Bean `lifecycleAdvisor`

Automatisch hat die Klasse des „Spring RCPs“ die Bezeichnungen samt Eingabefeldern und Schaltflächen ordentlich angeordnet. Wie immer fehlt noch der Text für die Internationalisierung,

2 EIN „SPRING RCP“-BEISPIEL

lediglich die Attribute der Person sind bereits aus den Tabellenspalten-Bezeichnungen bekannt. Schade ist an dieser Stelle, dass ein Einfügen der optisch benötigten Doppelpunkte nicht möglich ist. Würden sie in der *messages.properties* eingefügt, stünden sie leider auch in der Spaltenbezeichnung der Tabelle.

Beinhaltet die Klasse *PersonForm* eine „ComboBox“ zur Auswahl einer Eingabe, muss im Projekt noch die Bibliothek *commons-collections-3.2.1.jar* ergänzt werden.

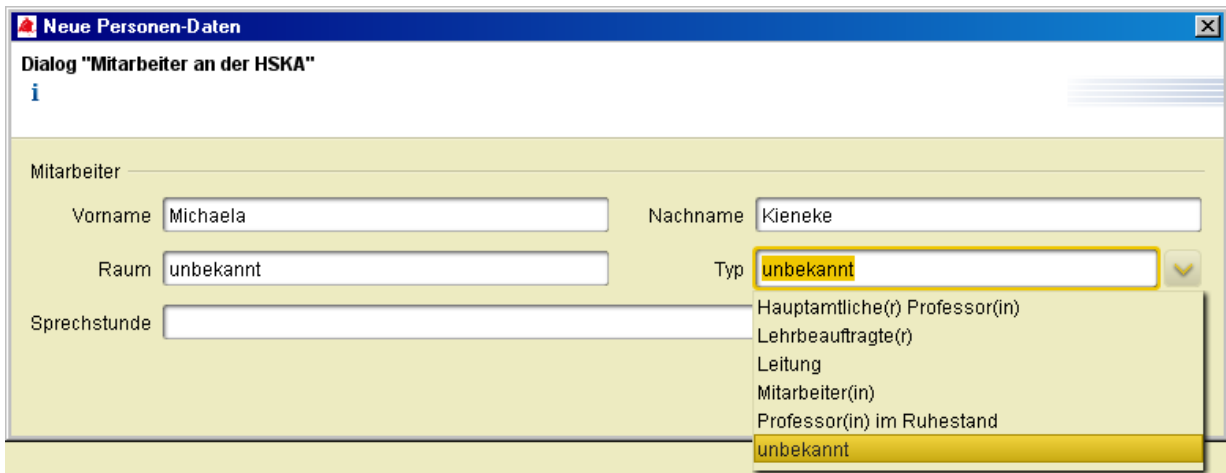


Abbildung 9: Neuer Datensatz

In der Klasse *PersonPropertiesDialog* können Anpassungen gemacht werden, ob es sich um einen neuen oder einen zu editierenden Datensatz handelt.

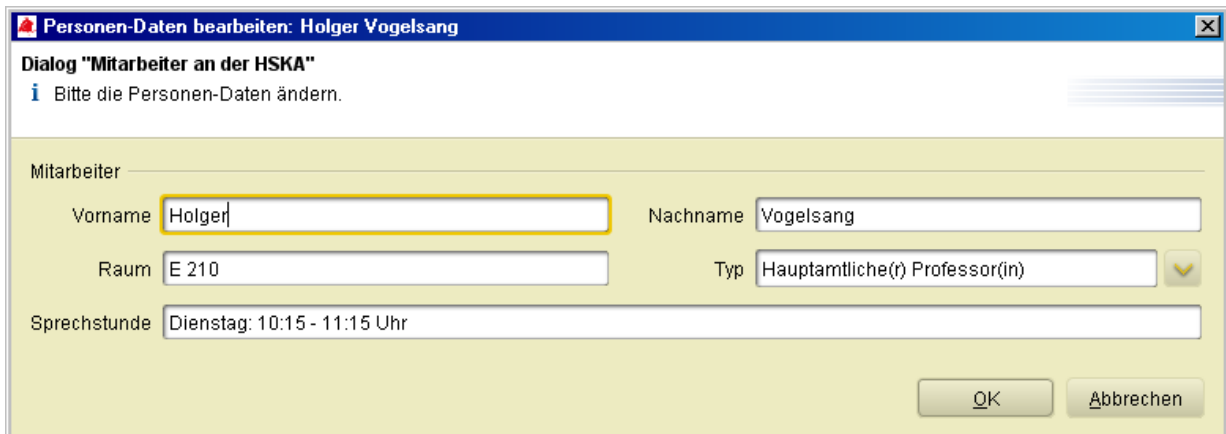


Abbildung 10: Zu editierender Datensatz

Die eingefügten Datensätze lassen sich nun auf Wunsch in der Tabelle sortieren, auswählen und bearbeiten. All dies ermöglicht das „Spring RCP“ mit vergleichsweise wenig Aufwand, da entsprechende Klassen zur Verfügung gestellt werden.

2 EIN „SPRING RCP“-BEISPIEL

```
1 private ActionCommandExecutor deleteExecutor
2     = new DeleteExecutor();
```

Listing 26: *ActionCommandExecutor deleteExecutor* in der Klasse *PersonView*

Um einen Datensatz zu löschen, wird auf die gleiche Weise ein *ActionCommandExecutor* in der Klasse *PersonView* erstellt. Das Löschen ist dann wie der Aufruf der Eigenschaften über die „Menu-“ und „Toolbar“ bzw. per Rechtsklick möglich.

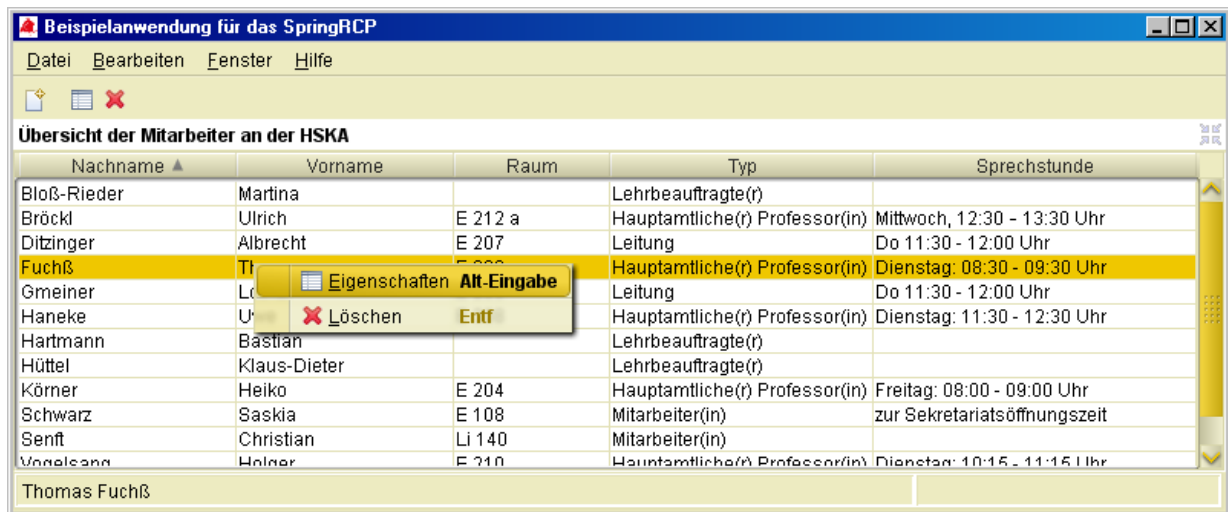


Abbildung 11: Das „Contextmenu“

2.3.9 Validierung der Dateneingaben

Zur Validierung der Dateneingabe wird die Klasse *PersonFormValidationRules* benötigt, welche von der Klasse *DefaultRulesSource* abgeleitet ist. Über einen regulären Ausdruck werden die gewünschten Eingaben festgelegt und den entsprechenden Eingabefeldern übergeben.

Um der Anwendung die Validierung bekannt zu machen, müssen die Bean *rulesSource* und die Bean *formComponentInterceptorFactory* in der *richclient-application-definition.xml* eingefügt werden.

```
1 <bean id="rulesSource"
2     class="de.test.data.PersonFormValidationRules" />
3
4 <bean id="formComponentInterceptorFactory"
5     class="org.springframework.richclient.form.builder.support.
6         ChainedInterceptorFactory">
7     <property name="interceptorFactories">
```

2 EIN „SPRING RCP“-BEISPIEL

```

7   <list>
8     <bean class="org.springframework.richclient.form.builder.
9       support.ColorValidationInterceptorFactory">
10    <property name="errorColor" value="255,245,245" />
11  </bean>
12  <bean class="org.springframework.richclient.form.builder.
13    support.OverlayValidationInterceptorFactory" />
14  <bean class="org.springframework.richclient.text.
15    TextComponentPopupMenuInterceptorFactory" />
16  <bean class="org.springframework.richclient.list.
17    ComboBoxAutoCompletionInterceptorFactory" />
18 </list>
19 </property>
20 </bean>

```

Listing 27: Beans *rulesSource* und *formComponentInterceptorFactory*

Die Klasse `ColorValidationInterceptorFactory` gibt die Farbe vor, in der eine Fehlermarkierung am entsprechenden Eingabefeld erstellt wird, solange die Eingabe nicht korrekt vorgenommen wurde. Die Klasse `OverlayValidationInterceptorFactory` zeigt diese Fehlermarkierung an. Mit der Klasse `TextComponentPopupMenuInterceptorFactory` wird das Eingabefeld um ein „Contextmenu“ erweitert, während eine Autovervollständigung mithilfe der Klasse `ComboBoxAutoCompletionInterceptorFactory` möglich ist.

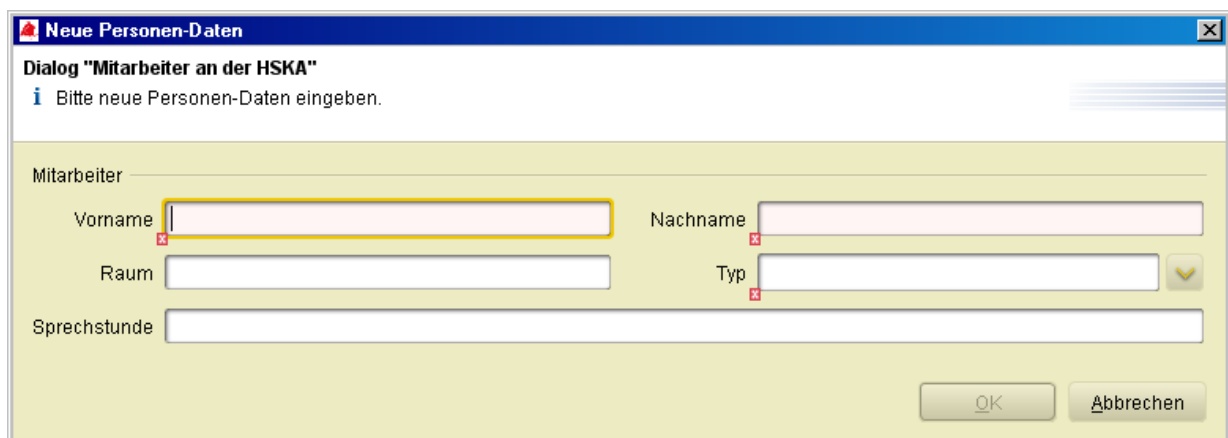


Abbildung 12: Validierung - Pflichtfelder

Die entsprechenden Fehlermeldungen werden von dem „Spring RCP“ selbst erzeugt und sind zum Teil etwas lustig, was wohl an der Übersetzung ins Deutsche liegt.

2 EIN „SPRING RCP“-BEISPIEL



Abbildung 13: Erweiterung der Validierung um ein „Contextmenu“

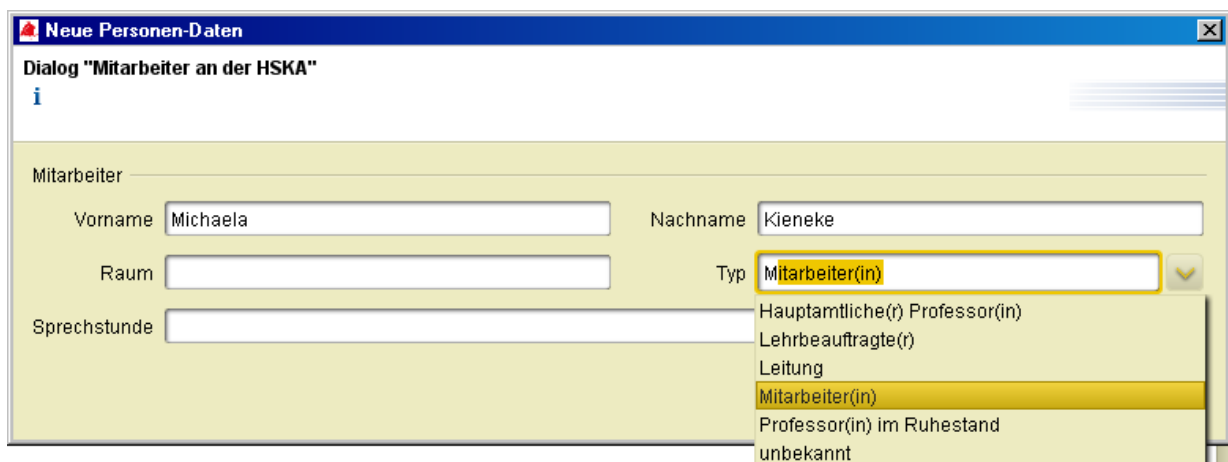


Abbildung 14: Autovervollständigung in der „Combobox“

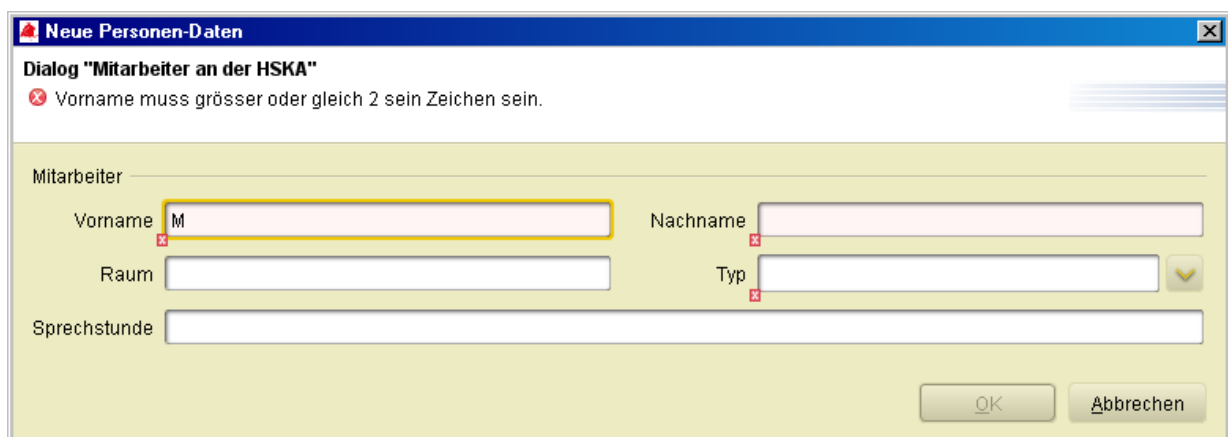


Abbildung 15: Validierung - Name muss länger als zwei Zeichen sein

2 EIN „SPRING RCP“-BEISPIEL

Abbildung 16: Validierung - Vorname nicht gleich Nachname

2.3.10 Mehrere „Views“ - „Docking“

Eine „Spring RCP“-Anwendung kann mehrere „Views“ beinhalten, die nicht nur getrennt voneinander sondern auch parallel betrachtet werden können. Neben unterschiedlichen Anordnungen nebeneinander können einzelne „Views“ sogar aus der Anwendung heraus gezogen und auf dem Desktop herum geschoben werden. Existieren mehrere Seiten in verschiedenen Fenstern, kann die „View“ mehrmals in diesen geöffnet sein, jedoch immer nur einmal pro Seite.

Zur Realisierung müssen die Klassen der „Views“, von denen sie erben, in der *richclient-application-definition.xml* von `DefaultViewDescriptor` auf `VLDockingViewDescriptor` geändert werden. Für das Verhalten der „Views“ können noch verschiedene Eigenschaften übergeben werden.

```

1 <bean id="personView"
2   class="org.springframework.richclient.application.docking.
3     vldocking.VLDockingViewDescriptor">
4   <property name="viewClass" value="de.hska.ui.PersonView" />
5   <property name="viewProperties">
6     <map>
7       <entry key="personDataStore"
8         value-ref="personDataStore" />
9     </map>
10  </property>
11  <property name="autoHideEnabled" value="true" />
12  <property name="closeEnabled" value="true" />
13  <property name="floatEnabled" value="true" />
14  <property name="maximizeEnabled" value="true"/>
15 </bean>

```

Listing 28: Neue „Properties“ in der Bean *personView*

2 EIN „SPRING RCP“-BEISPIEL

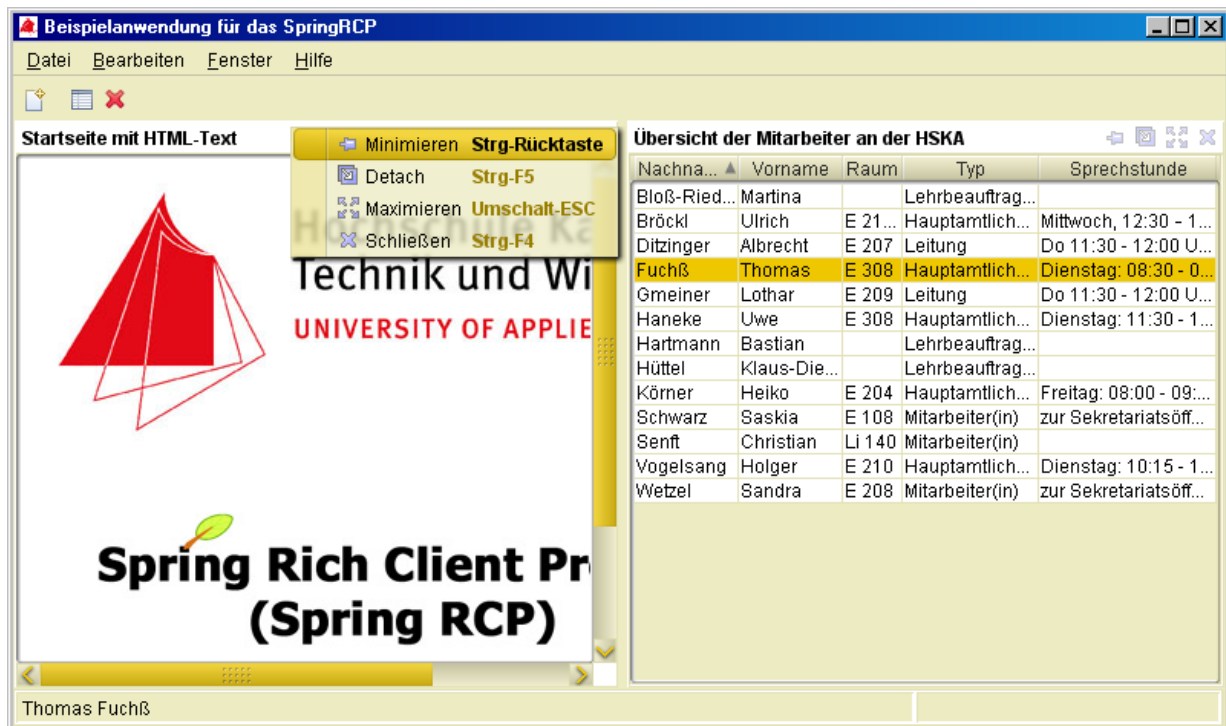


Abbildung 17: Zwei „Views“ nebeneinander

Zusätzlich muss noch die Bean `applicationPageFactory` eingefügt werden, die zur Verwaltung der „docking“-fähigen „Views“ benötigt wird.

```

1 <bean id="applicationPageFactory" depends-on="serviceLocator"
2   class="org.springframework.richclient.application.docking.
3     vldocking.VLDockingApplicationPageFactory">

```

Listing 29: Bean `applicationPageFactory`

Das „Docking“ erfordert zwei weitere Bibliotheken im `BuildPath`: `vldocking-2.1.4.jar` und `spring-richclient-docking-1.0.0.jar`.

Durch diese kleinen Änderungen erhält die entsprechende „View“ die Eigenschaften eines „Floating Windows“ und kann außerdem innerhalb der „Spring RCP“-Anwendung an unterschiedlichen Positionen „angedockt“ werden.

2.3.11 Die Oberfläche: „Swing“

Die Oberfläche des „Spring RCPs“ wird mit Elementen der „Swing“-Bibliothek erstellt. Somit besteht die Möglichkeit, bekannte *Look&Feel*-Bibliotheken zu verwenden. Hierzu wird die ent-

2 EIN „SPRING RCP“-BEISPIEL

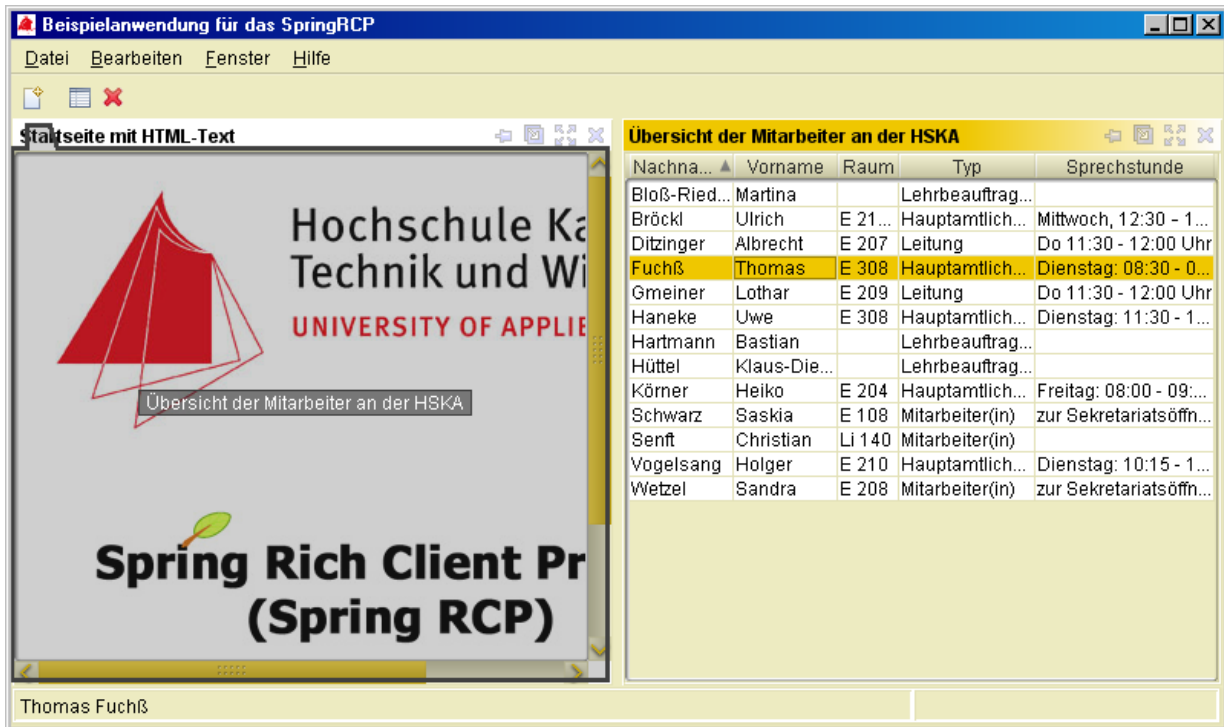


Abbildung 18: Eine „View“ verschieben

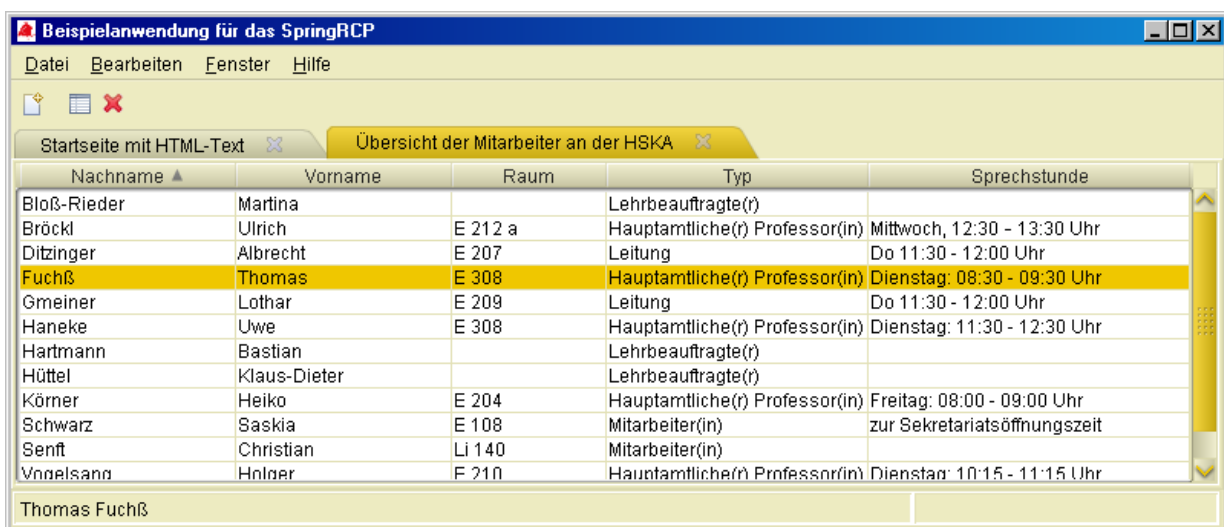
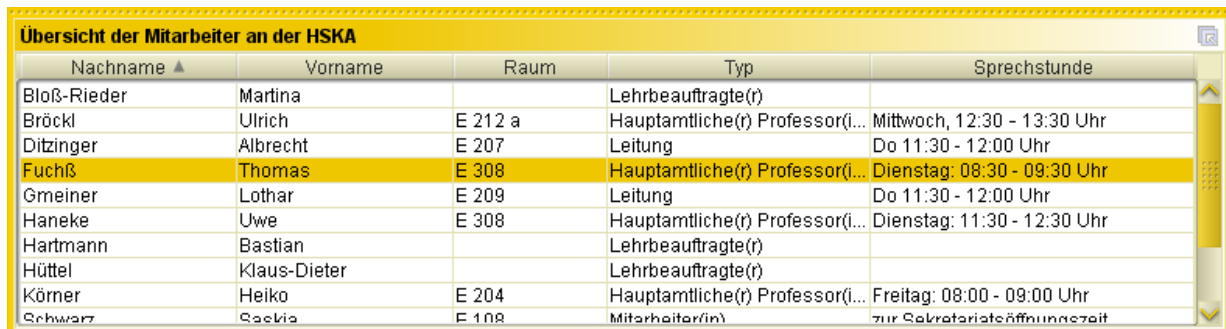


Abbildung 19: „Views“ hintereinander mit „Tabs“

2 EIN „SPRING RCP“-BEISPIEL



Nachname ▲	Vorname	Raum	Typ	Sprechstunde
Bloß-Rieder	Martina		Lehrbeauftragte(r)	
Bröckl	Ulrich	E 212 a	Hauptamtliche(r) Professor(i...	Mittwoch, 12:30 - 13:30 Uhr
Ditzinger	Albrecht	E 207	Leitung	Do 11:30 - 12:00 Uhr
Fuchß	Thomas	E 308	Hauptamtliche(r) Professor(i...	Dienstag: 08:30 - 09:30 Uhr
Gmeiner	Lothar	E 209	Leitung	Do 11:30 - 12:00 Uhr
Haneke	Uwe	E 308	Hauptamtliche(r) Professor(i...	Dienstag: 11:30 - 12:30 Uhr
Hartmann	Bastian		Lehrbeauftragte(r)	
Hüttel	Klaus-Dieter		Lehrbeauftragte(r)	
Körner	Heiko	E 204	Hauptamtliche(r) Professor(i...	Freitag: 08:00 - 09:00 Uhr
Schwarz	Saskia	E 108	Mitarbeiter(in)	zur Sekretariatsöffnungszeit

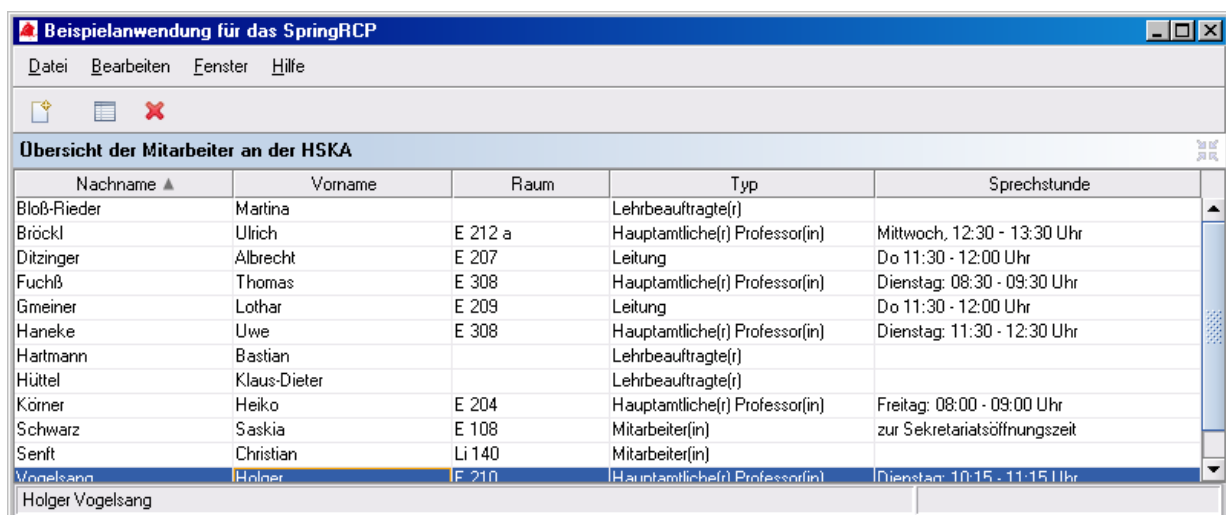
Abbildung 20: Tabellenansicht als „Floating Window“ auf dem Desktop

sprechende Jar-Datei im *BuildPath* bekannt gemacht und die entsprechende Klasse bei den *VM arguments* übergeben:

```
1 -Dswing.defaultlaf=com.nilo.plaf.nimrod.NimRODLookAndFeel
```

Listing 30: Aufruf des *Look&Feel* über die *VM arguments*

Dabei ist darauf zu achten, dass in der *richclient-application-definition.xml* keine Bean mehr mit dem Namen *lookAndFeelConfigurer* vorhanden ist, da diese sonst das übergebene Argument beim Laden der Anwendung überschreibt.



Nachname ▲	Vorname	Raum	Typ	Sprechstunde
Bloß-Rieder	Martina		Lehrbeauftragte(r)	
Bröckl	Ulrich	E 212 a	Hauptamtliche(r) Professor(in)	Mittwoch, 12:30 - 13:30 Uhr
Ditzinger	Albrecht	E 207	Leitung	Do 11:30 - 12:00 Uhr
Fuchß	Thomas	E 308	Hauptamtliche(r) Professor(in)	Dienstag: 08:30 - 09:30 Uhr
Gmeiner	Lothar	E 209	Leitung	Do 11:30 - 12:00 Uhr
Haneke	Uwe	E 308	Hauptamtliche(r) Professor(in)	Dienstag: 11:30 - 12:30 Uhr
Hartmann	Bastian		Lehrbeauftragte(r)	
Hüttel	Klaus-Dieter		Lehrbeauftragte(r)	
Körner	Heiko	E 204	Hauptamtliche(r) Professor(in)	Freitag: 08:00 - 09:00 Uhr
Schwarz	Saskia	E 108	Mitarbeiter(in)	zur Sekretariatsöffnungszeit
Senft	Christian	Li 140	Mitarbeiter(in)	
Vogelsang	Holger	E 210	Hauptamtliche(r) Professor(in)	Dienstag: 10:15 - 11:15 Uhr

Abbildung 21: Look&Feel „ExperienceRoyale“

Sollte dies jedoch gewünscht sein, kann ein *Look&Feel* mit folgendem Code eingebunden werden:

```
1 <bean id="lookAndFeelConfigurer"
2   class="org.springframework.richclient.application.config.
   JGoodiesLooksConfigurer">
```

2 EIN „SPRING RCP“-BEISPIEL

```
3 <property name="popupDropShadowEnabled" value="false" />
4 <property name="theme">
5   <bean class="com.jgoodies.looks.plastic.theme.ExperienceRoyale"
6     />
7 </property>
</bean>
```

Listing 31: Bean *lookAndFeelConfigurer*

2.3.12 Die „Icons“

Ähnlich wie bei den *messages.properties* können der Anwendung über *images.properties* einheitliche „Icons“ übergeben werden. Dies beinhaltet nicht nur das „Icon“ in der Titelzeile sondern geht über die Bildchen in der „Menu-“ bzw. „Toolbar“ bis zu den Anzeigen in den Dialogboxen. Hierzu muss in der *richclient-application-definition.xml* die Bean *imageResourcesFactory* angelegt werden. Wie schon bei den *messages.properties* wird auch hier zum Einen auf Bilder verwiesen, die über die „Spring Rich“-Bibliothek zugewiesen werden. Zum Anderen wird auf die Datei verwiesen, die projekteigene Pfade enthält. Werden in dieser Datei „Icons“ zugewiesen, die bereits in der „Spring“-Datei zugewiesen wurden, werden diese damit überschrieben.

```
1 <bean id="imageResourcesFactory"
2   class="org.springframework.context.support.
3     ResourceMapFactoryBean">
4   <property name="locations">
5     <list>
6       <value>classpath:org/springframework/richclient/image
7         /images.properties</value>
8       <value>classpath:images/images.properties</value>
9     </list>
10  </property>
</bean>
```

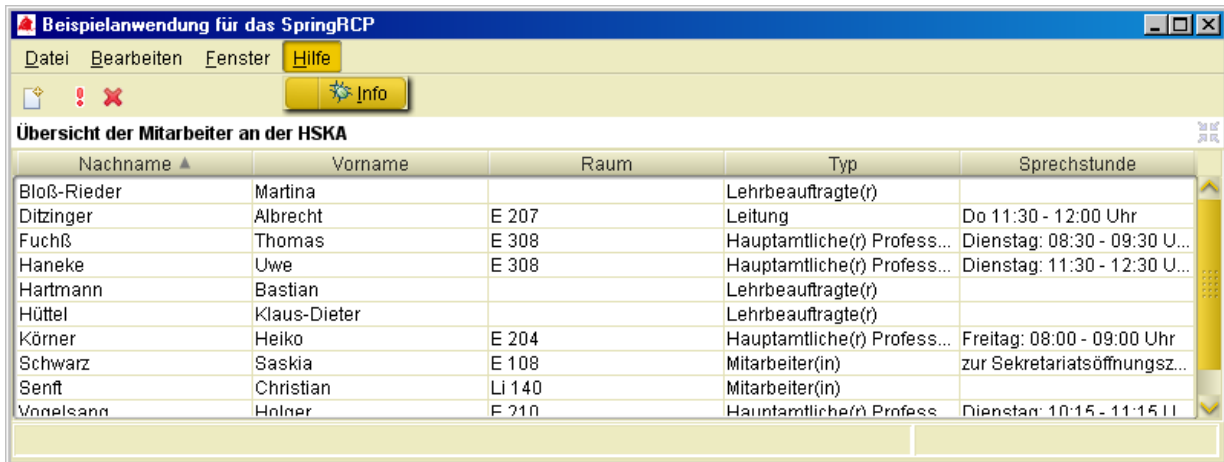
Listing 32: Bean *imageResourcesFactory*

Es besteht außerdem mittels der Bean *imageSource* und der Klasse *DefaultImageSource* die Möglichkeit, ein „Default“-Bild anzuzeigen, falls einmal ein Bild nicht gefunden wird.

```
1 <bean id="imageSource"
2   class="org.springframework.richclient.image.DefaultImageSource">
3   <constructor-arg index="0" ref="imageResourcesFactory" />
4   <property name="brokenImageIndicator"
5     value="/org/springframework/richclient/images/alert/error_obj.
6     gif" />
</bean>
```

Listing 33: Bean *imageSource*

2 EIN „SPRING RCP“-BEISPIEL



Nachname ▲	Vorname	Raum	Typ	Sprechstunde
Bloß-Rieder	Martina		Lehrbeauftragte(r)	
Ditzinger	Albrecht	E 207	Leitung	Do 11:30 - 12:00 Uhr
Fuchß	Thomas	E 308	Hauptamtliche(r) Profess...	Dienstag: 08:30 - 09:30 U...
Haneke	Uwe	E 308	Hauptamtliche(r) Profess...	Dienstag: 11:30 - 12:30 U...
Hartmann	Bastian		Lehrbeauftragte(r)	
Hüttel	Klaus-Dieter		Lehrbeauftragte(r)	
Körner	Heiko	E 204	Hauptamtliche(r) Profess...	Freitag: 08:00 - 09:00 Uhr
Schwarz	Saskia	E 108	Mitarbeiter(in)	zur Sekretariatsöffnungs...
Senft	Christian	Li 140	Mitarbeiter(in)	
Vogelsann	Holger	E 210	Hauptamtliche(r) Profess...	Dienstag: 10:15 - 11:15 U...

Abbildung 22: Veränderte „Icons“

Es können eigene Bilder und auch Bilder aller eingebundenen Bibliotheken verwendet werden.

```

1 applicationInfo.image=/images/hska_pyramide_icon.gif
2 propertiesCommand.icon=/org/springframework/richtclient/images/alert
  /hprio_tsk.gif
3 aboutCommand.icon=/org/springframework/richtclient/images/dev/
  debug_exc.gif

```

Listing 34: Bilder-Pfade in der *images.properties*

Die Angaben in dieser *images.properties* ändern das Icon des Titels in ein selbst erstelltes Bild, bei den Eigenschaften wird ein rotes Ausrufezeichen angezeigt und die Hilfe findet man über den grünen „Debug-Käfer“ von Eclipse.

2.3.13 Die „Hilfe“

Mithilfe des „Spring RCP“ lässt sich die „Menubar“ um eine „Hilfe“ zur Anwendung erweitern. Diese besteht aus den Dateien *simple.hs* und *simple.jhm* welche in einem neuen „Package“ *help* abgelegt werden. Verwendete HTML-Seiten werden im „Package“ *help.HTML* gespeichert.

In der *simple.hs* ist festgelegt, welche HTML-Seite beim Start der Hilfe aufgerufen wird.

```

1 <maps>
2   <homeID>Overview</homeID>
3   <mapref location="simple.jhm" />
4 </maps>
5
6 <view>
7   <name>favorites</name>

```

2 EIN „SPRING RCP“-BEISPIEL

```
8 <label>Favorites</label>
9 <type>javax.help.FavoritesView</type>
10</view>
```

Listing 35: *simple.hs*

Hier wird die Anzeige der verschiedenen „View“ geregelt. Im Beispiel gibt es nur die Möglichkeit, Favoriten anzulegen. Darüber hinaus wird auf die *simple.jhm* verwiesen, welche die einzelnen HTML-Seiten verwaltet. Diese Seiten können mit bekannten HTML-Tags gestaltet werden.

```
1 <mapID target="ContactsView" url="HTML/personView.htm"/>
2 <mapID target="InitialView" url="HTML/startView.htm"/>
3 <mapID target="Overview" url="HTML/overView.htm"/>
```

Listing 36: HTML-Seiten in der *simple.jhm*

Um die Hilfe aufrufen zu können, müssen in der *window-command-bars.xml* in der Bean *helpMenu* noch die entsprechende Referenz und die dazugehörige Bean *helpContentsCommand* ergänzt werden.

```
1 <bean id="helpMenu"
2   class="org.springframework.richclient.command.
3     CommandGroupFactoryBean">
4   <property name="members">
5     <list>
6       <ref bean="helpContentsCommand" />
7       [...]
8     </list>
9   </property>
10 </bean>
11 <bean id="helpContentsCommand"
12   class="org.springframework.richclient.command.support.
13     HelpContentsCommand">
14   <property name="helpSetPath" value="help/simple.hs" />
15 </bean>
```

Listing 37: Beans *helpMenu* und *helpContentsCommand*

Außerdem wird die Bibliothek *jhelp-2.0.jar* benötigt, die noch in den *BuildPath* mit eingebunden werden muss.

2.3.14 Die „ProgressBar“

In der erstellten Anwendung benötigt kein Vorgang soviel Zeit, dass es nötig bzw. überhaupt möglich wäre, einen Fortschrittsbalken in der „StatusBar“ der Anwendung anzuzeigen. Daher

2 EIN „SPRING RCP“-BEISPIEL



Abbildung 23: Hilfe für die Anwendung

wurde über die „Menubar“ eine Art „Dummy“ eingefügt, um auch diese Möglichkeit des „Spring RCPs“ zu zeigen.

Benötigt wird die Klasse `StatusBarDemoCommand`, die von der Klasse `ApplicationWindowAwareCommand` abgeleitet ist. Der Anwendung wird ein `ProgressMonitor` zugefügt. Den „Fortschritt“ darin zeigt der `SwingWorker` an, welcher einfach von 0 bis 100 in 10er-Schritten hochzählt.

In der `window-command-bars.xml` muss noch die entsprechende Bean `demoCommand` zum Aufruf der Klasse erstellt und diese im „Menu“ eingefügt werden.

```
1 <bean id="demoCommand"
2   class="de.hska.data.StatusBarDemoCommand" />
```

Listing 38: Bean `demoCommand` in der `window-command-bars.xml`

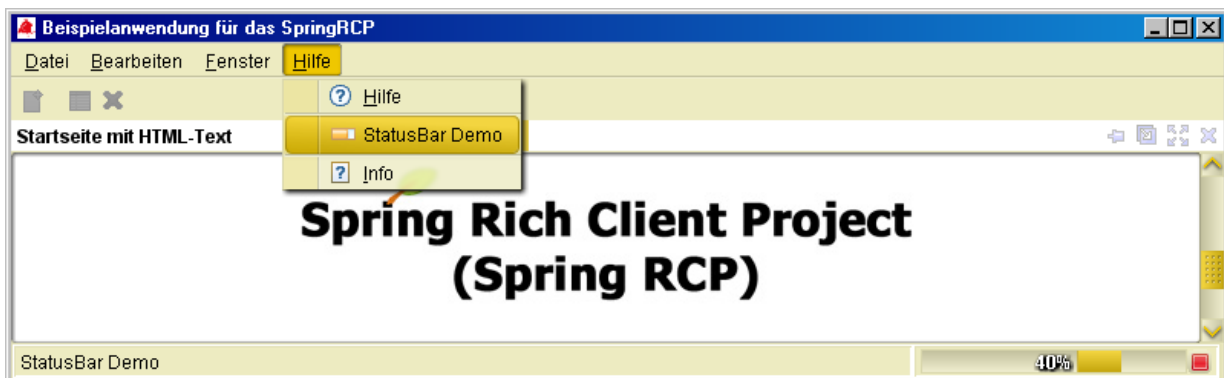


Abbildung 24: ProgressBar-Demo

2.4 Fazit

Das „Spring RCP“ ermöglicht eine relativ einfache Erstellung einer Anwendung mit ansprechender Oberfläche und vielen Funktionen. Da jedoch wenige Informationen über den Aufbau des „Spring RCPs“ bzw. kaum Dokumentationen vorhanden sind, ist es relativ arbeitsintensiv, diese „einfachen Wege“ zu finden und zu realisieren.

3 Ausblick in die Zukunft

Nachdem sich eineinhalb Jahre scheinbar nichts bei der Entwicklung des Spring RCP getan hatte, kam am 17. März 2008 endlich die Version 1.0.0 heraus. Jedoch sind die noch geplanten Erweiterungen des „Spring Rich Client Projects“ mit dem momentan vorliegenden Projekt nicht möglich, weswegen die Entwickler ein völlig neues Projekt begonnen haben, was im Oktober 2008 unter dem Namen „Spring Desktop“ herauskommen sollte¹¹.

¹¹The Spring Framework: „Spring richclient - Home“,
<http://spring-rich-c.sourceforge.net/1.0.0/index.html> [Stand 12.11.2008]

Literatur

- [GSwSRCP] Geertjan Wielenga: „Getting Started with Spring RCP“,
<http://netbeans.dzone.com/news/spring-rcp-tutorial?page=0%2C0>
- [PiJ] Sigrid Kersken-Canbaz: “Programmieren in Java“ - „Spring Rich Client Project“,
<http://www.programmieren-in-java.de/de/content/spring-rich-client-project>
- [RCPDok] Alessandro Melandri: Pronux Wiki - „RCPDokumentation“,
<http://www.pronux.de/Wiki/Wiki.jsp?page=RCPDokumentation>
(z.T. etwas überholt, aber für kleinere Tipps durchaus brauchbar)
- [SRCP] Spring Rich Client Project (RCP) - User Documentation,
<http://opensource.atlassian.com/confluence/spring/display/RCP/Home>
- [Spr2] Daniel Oltmanns, Stefan Edlich: „Spring 2 für Grünschnäbel“,
Books an Demand GmbH, Norderstedt; 1. Auflage 2007