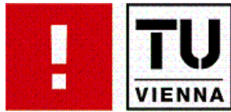

TU Wien

Institut für Softwaretechnik und Interaktive Systeme

Markus Demolsky

Entwickeln von Swing-Applikationen mit Spring RCP

PR Arbeit

Praktikumarbeit
Praktikum aus Software-Entwicklung

Betreuer: Dipl.Ing. Dr. Alexander Schatten

Wien, 16. November 2005

Inhaltsverzeichnis

1 Was ist Spring Rich Client 3
1.1 Ziele von Spring RCP . . . 3

2 Komponenten von Spring RCP 4
2.0.1 Application-Context . . . 6

3 GUI Hierarchie 11
3.1 Views in Spring RCP . . . 12
3.1.1 Erstellen von Views . . . 12
3.2 Formulare in Spring RCP . . . 13
3.2.1 Erstellen von Formularen . . . 13
3.2.2 Master/Detailformulare . . . 14

4 Binding Framework 16
4.1 Standard Swing Binding . . . 17
4.2 Benutzerdefiniertes Binding . . . 18

5 Warum Spring RCP einsetzen 21
5.1 Binding in Swing-Anwendungen . . . 22
5.2 Binding in Spring RCP . . . 23

6 Fazit 25

Abbildungsverzeichnis 26

Tabellenverzeichnis 27

Literaturverzeichnis 28

1 Was ist Spring Rich Client

Spring Rich Client ist ein Unterprojekt der Spring-Gruppe. Das J2EE (J5EE) Framework Spring ist auf einer sehr abstrakten Ebene in der Software Architektur angesiedelt. Es verfolgt den sogenannten Lightweight Ansatz der den großen Vorteil bietet, dass so wenig wie möglich eine Abhängigkeit vom Framework besteht. Dabei werden die einzelnen Komponenten einer J2EE Anwendung zusammengesetzt und zusätzliche Dienste, wie z.B Transatkionssteuerung, Sessionhandling, etc. deskriptiv konfiguriert.

Spring RCP nutzt das Applikationsframework Spring und hat sich zum Ziel gemacht, ein abstraktes Swing Framework zur Verfügung zu stellen.

1.1 Ziele von Spring RCP

- Ein Hilfswerkzeug zur Verfügung stellen um hochqualifizier- und konfigurierbare GUI Applikationen zu erstellen
- Reiche Bibliothek von UI Factories und Support Klassen
- Schnellere Integration mit bereits bestehenden Rich Client Projekten wie JGoodies-Forms und Table-Layout
- Grundideen vom Springframework übernehmen .. OO Design, Dokumentation und Testen
- Action-Framework um eine zentralisierte Verwaltung der Aktionen in der Applikation zu ermöglichen
- Multiple Window Management, Page Configuration und View Management
- Eine Ansammlung von Support-Klassen die dem Entwickler bei gebräuchlichen Rich Client Anforderungen unterstützen:
  - Well formed dialogs
  - Wizards
  - Input Validiation
  - Button Bars
  - Internationalisierung
  - Image/Icon Caching
  - Progress Monitoring
  - UI Threading
  - Tree Table/property sheet
  - usw.

2 Komponenten von Spring RCP

Das Spring RCP Framework setzt sich aus einer Menge von Komponenten zusammen, welche in Abbildung 2.1 dargestellt werden.

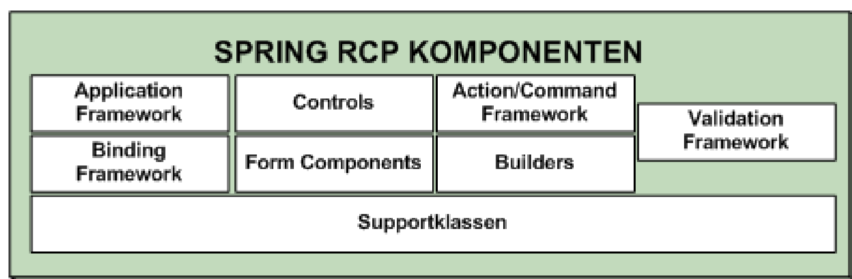

Abbildung 2.1: Komponenten von Spring RCP

Application Framework Diese Komponente bildet die oberste Schicht, das sogenannte Dach von Spring RCP und ist für das gesamte Zusammenspiel aller Komponenten im System zuständig. Die Application Komponente wird durch die Konfigurationsdatei *application-context.xml*, definiert. Das Application Framework bietet folgende Service an:

- Metadaten der gesamten Applikation
- Launcher für das Starten der Applikation
- Verwaltung applikationsweiter Dienste
- Window Manager

Action/Command Framework Das Action/Command Framework wird durch die Konfigurationsdatei *commands-context.xml* definiert. In dieser Datei wird die Grundstruktur des Menü und der Toolbar definiert. Das Framework stellt auf Basis der Konfigurationsdatei das Menü und die Toolbar zusammen und übernimmt das gesamte Eventhandling.

Validation Framework Die Validierung ist in der Softwareentwicklung ein sehr komplexes Thema. Vor allem bei Client-/Serveranwendungen stellt sich immer die Frage ob man nun am Client oder am Server validiert bzw. die Validierung aufteilt. Das Validation Framework von Spring RCP ist eine clientseitige Validierung und wird zur Laufzeit in der GUI ausgeführt.

Binding Framework Das Binding Framework zählt sicherlich zu den Kernstücken von Spring RCP. Es übernimmt das Binding zwischen den Objekten und den Form-Modellen, d.h. die Verbindung zwischen den GUI-Komponenten und den Objekteigenschaften.

Controls Spring RCP bietet eine große Bibliothek von bereits vordefinierten Komponenten an, welche nur mehr an die persönlichen Wünsche angepasst werden müssen. Hier nur die wichtigsten Komponenten:

- Standard Swing-Komponenten
- Tabellen
- Verschiedene Arten von Dialogen für Eingaben und Fehlermeldungen
- Verschiedene Auswahldialoge, unter anderem File-Chooser, Color-Chooser.

Form Components Die Formkomponenten umfassen verschiedene Arten von Formularen und Dialogen.

Builders Die Erstellung von GUIs mit Tools hat zum Vorteil, dass man effizient und einfach graphische Oberflächen erstellen kann. Der generierte Code hingegen ist kaum wartbar und nur schwer zu lesen. Dafür stellt Spring RCP Builder zur Verfügung, die die Erstellung von GUIs einfach und schnell machen soll. Dabei soll der Code gut leserlich und leicht wartbar sein. Spring RCP verwendet dabei folgende Layout Manager:

- TableLayoutBuilder von JGoodies
- GridBagLayoutBuilder

Supportklassen Spring RCP stellt eine Menge von Supportklassen zur Verfügung die es erlaubt auf einer abstrakten Ebene zu arbeiten. So kann man sich zum Beispiel eine sortierbare Tabelle erstellen lassen.

2.0.1 Application-Context

Wie bereits erwähnt, wird der Context durch die Konfigurationsdatei *application-context.xml* definiert, welche im folgenden Abschnitt nun im Detail erklärt wird.

```
<!-- APPLICATION -->
<bean id="application"
  class="org.springframework.richclient.
         application.Application">
  <constructor-arg index="0">
    <ref bean="applicationDescriptor"/>
  </constructor-arg>
  <constructor-arg index="1">
    <ref bean="applicationAdvisor"/>
  </constructor-arg>
</bean>

<bean id="applicationDescriptor"
  class="org.springframework.richclient.application.
         support.DefaultApplicationDescriptor">
  <property name="version">
    <value>1.0</value>
  </property>
  <property name="buildId">
    <value>20041025001</value>
  </property>
</bean>

<!-- ADVISOR UND SEITENAUFBAU MDI -->
<bean id="applicationAdvisor" class="at.demolsky.lms.
          LMSLifecycleAdvisor">
  <property name="windowCommandBarDefinitions">
    <value> at/demolsky/lms/ui/commands-context.xml</value>
  </property>
  <property name="startingPageId">
    <value>customerOverview</value>
  </property>
</bean>
```

Abbildung 2.2: Ausschnitt aus der application-context.xml

Das Bean *Application* ist die Basis einer jeden Spring-RCP Anwendung, sie bildet die zentrale Schnittstelle des gesamten Systems. Der *ApplicationDescriptor* beinhaltet Metainformationen der Applikation und der *ApplicationAdvisor* ist für den LifeCycle der Anwendung zuständig. Dem

Advisor wird die Konfigurationsdatei *commands-context.xml*,(siehe Abbildung 2.3) übergeben, welche für den Aufbau des Menü und der Toolbar zuständig ist. Die *startingPageId* macht nichts anderes, als die Startseite der Applikation festzulegen.

```
..
<bean id="menuBar"
      class="org.springframework.richclient.
             command.CommandGroupFactoryBean">
    <property name="members">
      <list>
        <ref bean="fileMenu"/>
        <ref bean="windowMenu"/>
      </list>
    </property>
</bean>
<bean id="toolBar"
      class="org.springframework.richclient.
             command.CommandGroupFactoryBean">
    <property name="members">
      <list>
        <ref bean="showCustomerOverview"/>
        <ref bean="showLicenseOverview"/>
      </list>
    </property>
</bean>
<bean id="fileMenu"
      class="org.springframework.richclient.
      command.CommandGroupFactoryBean">
    <property name="members">
      <list>
        <bean
          class="org.springframework.richclient.
                 command.support.ExitCommand"/>
      </list>
    </property>
</bean>
..
```

Abbildung 2.3: Ausschnitt aus der commands-context.xml

Abbildung 2.3 stellt nur einen kleinen Ausschnitt der Konfigurationsdatei dar, enthält aber die wichtigsten Konstrukte. Das Bean *menuBar* definiert die Menüleiste und enthält in unserem Fall nur zwei Einträge, nämlich ein FileMenu und ein WindowMenu. Das *FileMenu* besteht nur aus dem Eintrag exit, welche die Anwendung schließt.

```
<bean id="lookAndFeelConfigurer"
      class="org.springframework.richclient.
             application.config.JGoodiesLooksConfigurer">
    <property name="theme">
      <bean class="com.jgoodies.looks.plastic.
                   theme.ExperienceBlue"/>
    </property>
</bean>
```

Abbildung 2.4: Ausschnitt aus der application-context.xml

Der *LookAndFeelConfigurer* ermöglicht ein leichtes Umstellen des Look and Feels der Applikation. Dieser Konfigurator ist jedoch nur für JGoodies ausgelegt. Möchte man zum Beispiel das Metal Look and Feel einstellen, so muss man dies über UIManager vornehmen, welches ebenfalls über die Konfigurationsdatei vorgenommen werden kann.

In Abbildung 2.5 werden alle benötigten Resourcen (Messages, Images und Icons) für die Anwendung geladen. Für diesen Zweck steht der *applicationObjectConfigurer* und die *componentFactory* zur Verfügung. Labelbezeichnungen, Meldungen, etc. erhält man über das Bean *messageSource*, siehe Abbildung 2.6, wo alle benötigten Message-Resourcen in eine Liste geladen werden. Dabei handelt es sich um standardmäßige Property-Dateien (messages.properties). Spring-RCP stellt eine standardmäßige Property-Datei zur Verfügung, welche immer eingebunden werden sollte, da hier Framework spezifische Fehlermeldungen definiert sind.
Zu guter letzt müssen nun auch die Images und Icons der Applikation zur Verfügung gestellt werden, siehe Abbildung 2.7. Bei den Images handelt es sich wieder um standardmäßige Property-Dateien, die der *imageResourceFactory* als Liste zur Verfügung gestellt werden.

```
<bean id="applicationObjectConfigurer"
      class="org.springframework.richclient.
             application.config.
             DefaultApplicationObjectConfigurer">
    <constructor-arg index="0">
      <ref bean="messageSource"/>
    </constructor-arg>
    <constructor-arg index="1">
      <ref bean="imageSource"/>
    </constructor-arg>
    <constructor-arg index="2">
      <ref bean="iconSource"/>
    </constructor-arg>
</bean>

<bean id="componentFactory"
      class="org.springframework.richclient.
             factory.DefaultComponentFactory">
    <property name="messageSource">
      <ref bean="messageSource"/>
    </property>
    <property name="iconSource">
      <ref bean="iconSource"/>
    </property>
</bean>
```

Abbildung 2.5: Ausschnitt aus der application-context.xml

```
<bean id="messageSource"
      class="org.springframework.context.support.
             ResourceBundleMessageSource">
    <property name="basenames">
      <list>
        <value>org.springframework.richclient.
               application.messages</value>
        <value>at.demolsky.lms.ui.messages</value>
        <value>at.demolsky.lms.customer.ui.messages</value>
        <value>at.demolsky.lms.license.ui.messages</value>
        <value>at.demolsky.lms.contract.ui.messages</value>
      </list>
    </property>
</bean>
```

Abbildung 2.6: Ausschnitt aus der application-context.xml

```
<bean id="imageResourcesFactory"
    class="org.springframework.context.
           support.ResourceMapFactoryBean">
    <property name="locations">
      <list>
        <value>classpath:org/springframework/
               richclient/image/images.properties</value>
        <value>classpath:at/demolsky/lms/ui/
               images.properties</value>
        <value>classpath:at/demolsky/lms/license/ui/
               images.properties</value>
        <value>classpath:at/demolsky/lms/contract/ui/
               images.properties</value>
        <value>classpath:at/demolsky/lms/customer/ui/
               images.properties</value>
      </list>
    </property>
    <property name="resourceBasePath">
      <value>images/</value>
    </property>
</bean>

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

<bean id="iconSource"
      class="org.springframework.richclient.
             image.DefaultIconSource">
    <constructor-arg index="0">
      <ref bean="imageSource"/>
    </constructor-arg>
</bean>
```

Abbildung 2.7: Ausschnitt aus der application-context.xml

3 GUI Hierarchie

Das Spring Rich Client Framework verwendet eine Reihe von GUI Abstraktionen

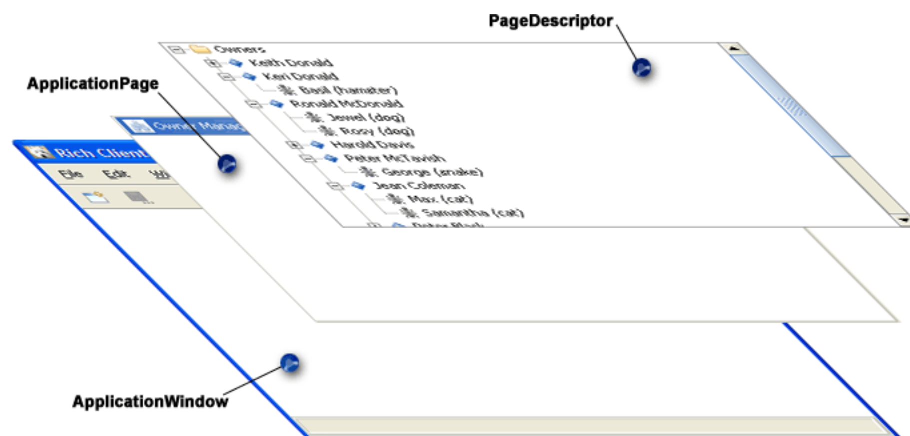

Abbildung 3.1: GUI Hierarchie in Spring RCP

Application Window Dieses Interface bildet die unterste Schicht der GUI Ebenen und ist für den grundlegenden Applikationsaufbau zuständig. Spring RCP stellt hierfür eine Standardimplementierung DefaultApplicationWindow zu Verfügung, welche ein Hauptfenster bestehend aus:

- Menüleiste,
- Toolbar,
- Arbeitsbereich und
- Statusleiste

bereitstellt.Die Menüleiste, sowie die Toolbar werden vom User über die Konfigurationsdateien *commands-context.xml* definiert.

Application Page Die ApplicationPage bildet den Arbeitsbereich (=Container) im Hauptfenster. Der Container ermöglicht es, auch mit mehreren Fenstern (= Views) zu arbeiten.

Page Descriptor Der PageDescriptor liefert Metadaten einer Seite.

3.1 Views in Spring RCP

> *A view is a panel-like component displayed within an area on the page associated with an application window. There can be multiple views per page; a single view can only be displayed once on a single page. View instances encapsulate the creation of and access to the visual presentation of the underlying control. A view's descriptor – which is effectively a singleton – can be asked to instantiate new instances of a single view for display within an application with multiple windows. In other words, a single view instance is never shared between windows* [1]

Eine View ist also eine Komponente, welche in einem Container des ApplicationWindow angezeigt werden kann. Üblicherweise werden Views für Übersichten von Datensätzen verwendet, wie zum Beispiel eine Kundenliste.

3.1.1 Erstellen von Views

Eigene Views werden von der abstrakten Klasse AbstractView abgeleitet. Die Abbildung 3.2 auf 12 zeigt das dazugehörige UML Diagramm.

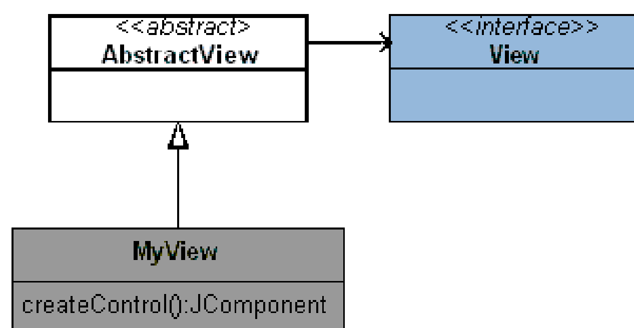

Abbildung 3.2: Erstellen einer View

Die einzige Methode welche implementiert werden muss ist *createControl*. In dieser Methode wird die Form der View zusammengestellt. Diese JComponent wird von Spring RCP in einen passenden Container gepackt und im Arbeitsbereich des Application Window angezeigt.

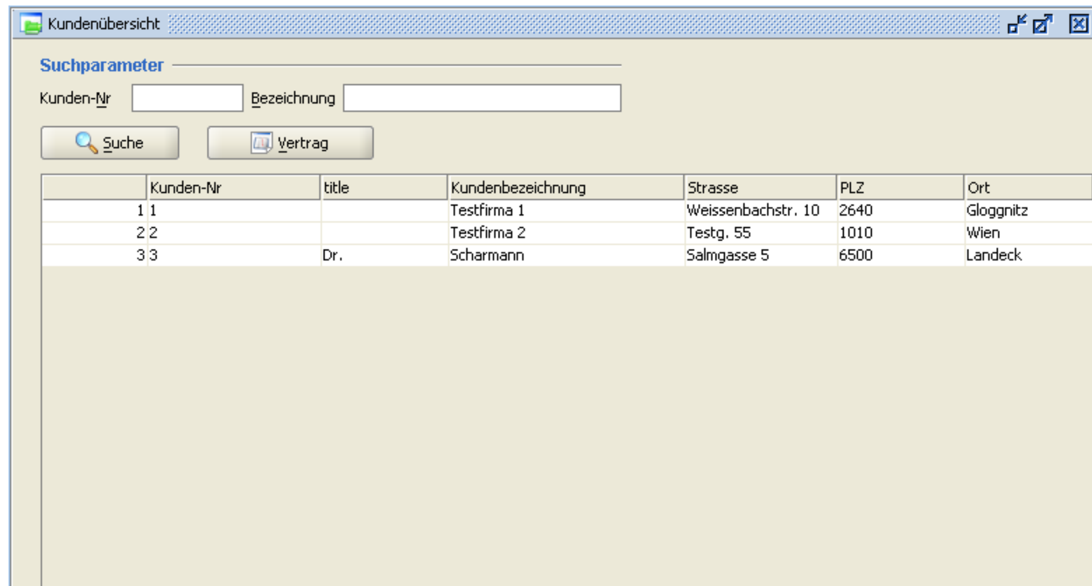

Abbildung 3.3: Beispiel einer View

3.2 Formulare in Spring RCP

Formulare stellen kleine wiederverwendbare Komponenten dar, welche in der gesamten Anwendung und auch von anderen Formularen wieder verwendet werden können. Ein Formular steht meistens mit einem POJO (Plain Old Java Object) in Verbindung. Das ist das sogenannte Binding.

3.2.1 Erstellen von Formularen

Eigene Formulare werden im Normalfall von der abstrakten Klasse AbstractForm abgeleitet. Die Abbildung 3.4 auf 13 zeigt das dazugehörige UML Diagramm.

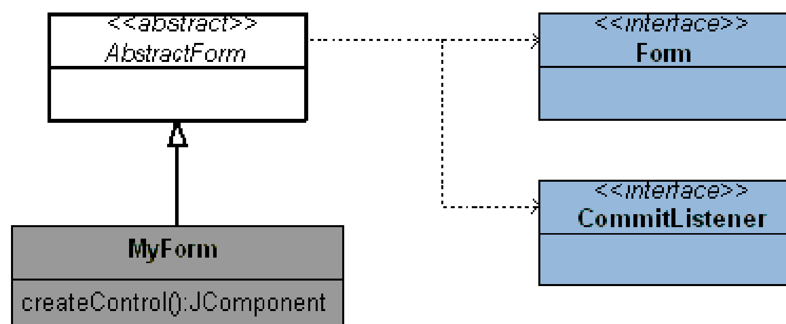

Abbildung 3.4: Erstellen einer Form

Die einzige Methode welche implementiert werden muss ist *createControl*. In dieser Methode wird die eigentliche Formularkomponente zusammengestellt. Diese erstellte Komponente kann man danach in einen beliebigen Container packen. Der Sourcecode in Abbildung 3.5 zeigt ein einfaches Beispiel für die Erstellung einer Formularkomponente und Abbildung 3.6 auf 14 ist das Ergebnis.

```
protected JComponent createFormControl(){
  TableFormBuilder fb = new TableFormBuilder(getBindingFactory());
  fb.add("publisher"); fb.add("publisherArtNr");
  fb.row();
  fb.add("publisherArtTitle");
  fb.add("publisherArtPrice");
  fb.row();
  fb.add("compArtNr");
  fb.getLayoutBuilder().cell();
  fb.row();
  return fb.getForm();
}
```

Abbildung 3.5: Beispielcode für die Erstellung einer einfachen Form

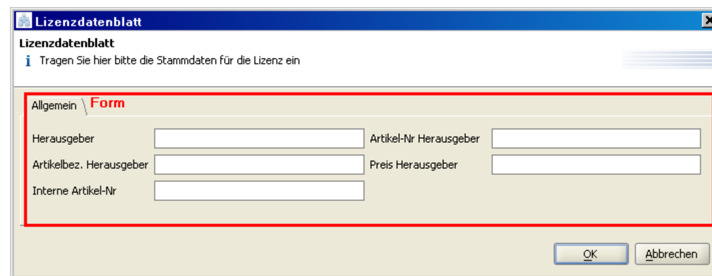

Abbildung 3.6: GUI der erstellten Form

3.2.2 Master/Detailformulare

Diese Art von Formularen erlauben eine einfache Erstellung von Master-/Detailinformationen. Dabei werden die Masterinformationen oft in Form einer Tabelle oder eines Baumes ausgegeben. Die Detailinformationen sind bearbeitbare Attribute, wie Eingabefelder, Comboboxen, Checkboxen, etc.
Für diesen Zweck stellt das Framework die Master/Detail Komponente zur Verfügung, bestehend aus:

- AbstractMasterForm
- AbstractDetailForm
- AbstractTableMasterForm

Die Abbildung 3.7 zeigt das UML Diagramm für die Erstellung einer Master/-Detailansicht. MyMasterForm wird von der abstrakten Klasse AbstractMasterTableForm abgeleitet. Diese Klasse stellt ein Template für die Master/Detaildarstellung in Form einer Tabelle zur Verfügung. MyMasterForm muss dabei nur die beiden Methoden:

- **getColumnPropertyNames**
  Legt die anzuzeigenden Spalten in der Mastertabelle fest. Die Spaltenbezeichnungen werden aus den message.properties ausgelesen.

- **createDetailForm**
  Stellt den Detailbereich dar und erhält nur einen Verweis auf MyDetailForm. Die eigentliche Erstellung der Detailkomponente erfolgt in MyDetailForm.

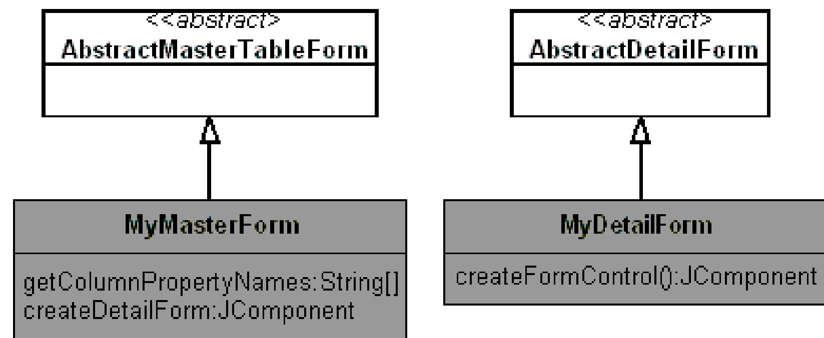

Abbildung 3.7: UML für Master/Detailform

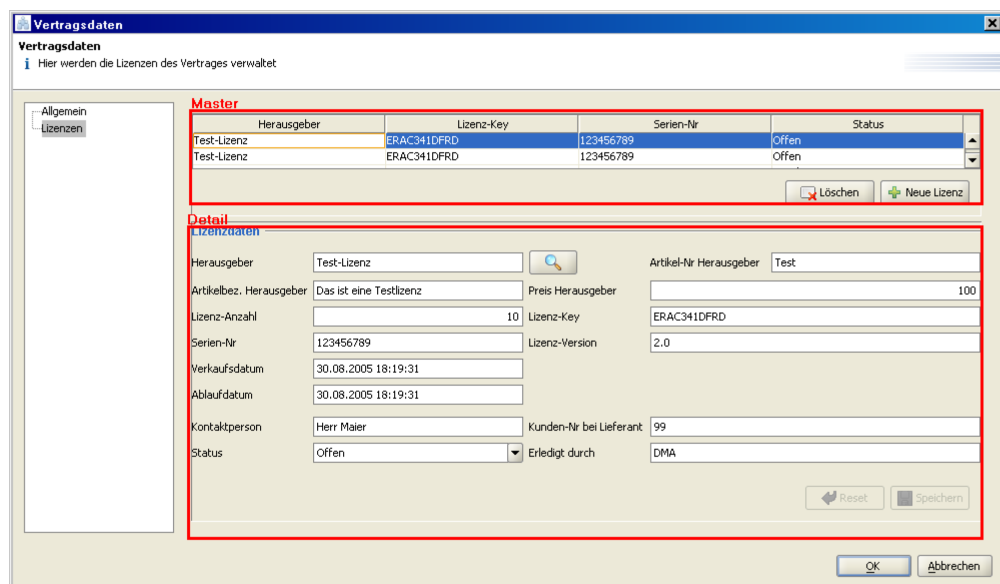

Abbildung 3.8: GUI für Master/Detailform

4 Binding Framework

Das Binding Framework ist für die Verbindung zwischen Objektattributen und graphischen Formularkomponenten zuständig. Für das automatische Binding ist die Schnittstelle FormModel zuständig. Ein Formmodell repräsentiert den Zustand und das Verhalten der Form unabhängig von dem User Interface welches das FormModel verwendet. Erst wenn ein Commit im FormModel erfolgt, werden die Daten der graphischen Formularkomponenten in das FormModel geschrieben, unter der Bedingung, dass diese den definierten Validierungsregeln entsprechen.

Die FormModel Schnittstelle wird durch weitere abgeleitete Schnittstellen:

- ConfigurableFormModel und
- HierarchicalFormModel

erweitert. Abbildung 4.1 auf 16 zeigt das passende UML Diagramm.

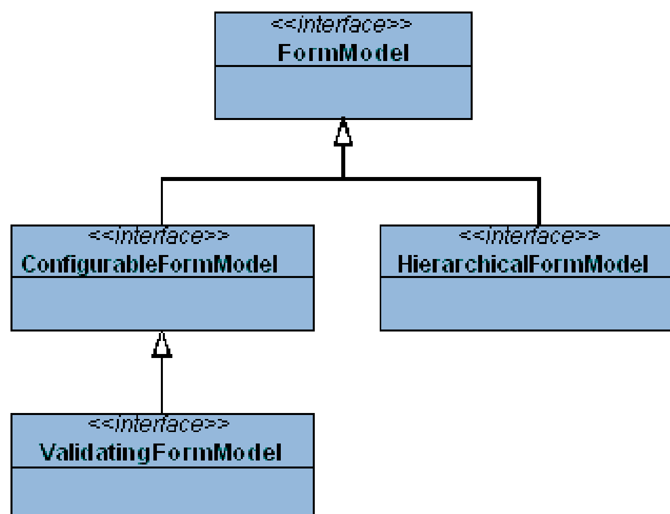

Abbildung 4.1: FormModel

ConfigurableFormModel Ist ein Sub-Interface von FormModel und erlaubt es das FormModel frei zu konfigurieren.

HierarchicalFormModel Ist ein Sub-Interface von FormModel und stellt einen Teil einer FormModel Hierarchie dar. Man hat dadurch die Möglichkeit, dass ein FormModel aus mehreren FormModel besteht. Abbildung 4.2 auf 17 zeigt den Sachverhalt. Man bezeichnet dies als eine Parent-Child Beziehung:

- Der Zustand (Aktiv/Inaktiv) des Child-Modell hängt vom Zustand des Parent-Modell ab. Hingegen hat der Zustand des einzelnen Child-Modell keinen Einfluss auf jenen des Parent-Modell.
- Werden Änderungen in einem Child-Modell gemacht, so wurde auch das Parent-Modell bearbeitet. Umgekehrt hat es jedoch keine Auswirkung.

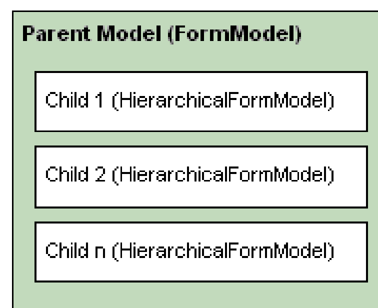

Abbildung 4.2: Beispiel für eine Hierarchie eines FormModel

ValidatingFormModel Ist ein Sub-Interface von FormModel und ermöglicht es die einzelnen Formeigenschaften zu validieren.

Spring stellt das DefaultFormModel zur Verfügung. Es ist eine Standardimplementierung die ein konfigurierbares, hierarchisches und validierbares FormModel zur Verfügung stellt.

Abbildung 4.3 zeigt ein einfaches Beispiel für ein Binding zwischen einem Objekt und einem Formular.

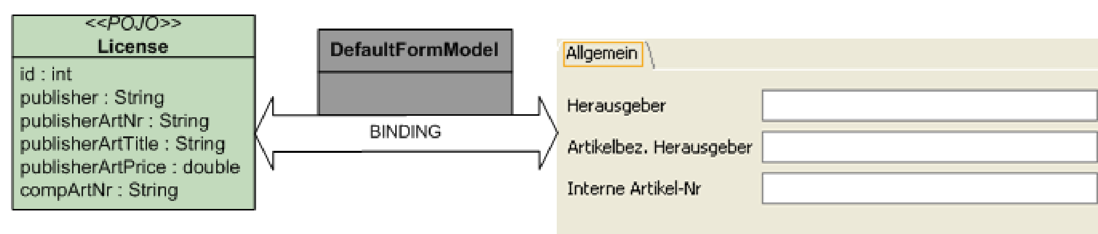

Abbildung 4.3: Beispiel für ein einfaches Binding

4.1 Standard Swing Binding

Spring RCP stellt Standardkomponenten für das Binding von Swing-Komponenten zur Verfügung. So wird beispielsweise ein String immer in einer Textbox angezeigt, oder ein Boolean in Form einer Checkbox dargestellt. Diese Komponenten befinden sich alle im Package org.springframework.richclient.form.binding.swing Die SwingBindingFactory ist eine passende Implementierung der abstrakten Klasse BindingFactory und

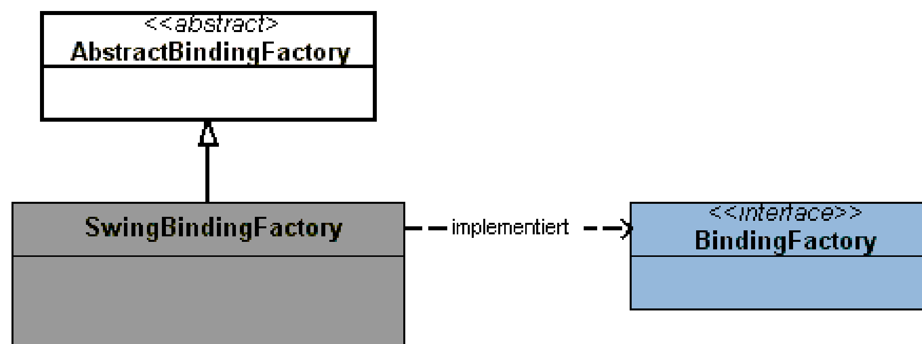

Abbildung 4.4: Swing Binding Factory

Attribut	Typ	SwingBinding Factory	GUI Element
id	Integer	createBoundTextField(id)	Textfeld
title, title1	String	createBoundTextField(title)	Textfeld
contract	Boolean	createBoundCheckBox(contract)	Checkbox
customerType	String	createBoundComboBox(customerType, Customer.getCustomerTypes())	Combobox mit den Typen von CustomerTypes. Wert wird aber im Attribut customer-Type gespeichert.

Tabelle 4.1: Standard Swing Binding Factory

bietet eine reiche Ansammlung von Mehtoden an, um Objektattribute mit Swing-Komponenten zu verbinden. Der folgende Abschnitt widmet sich nun ein paar kleinen Beispielen, um das Binding in Spring RCP zu demonstrieren. Als Beispiel verwenden wir das POJO in Abbildung 4.5 auf 18.

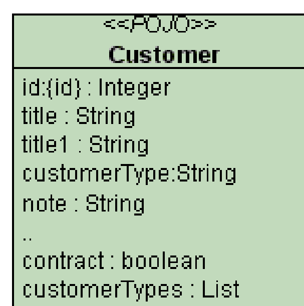

Abbildung 4.5: POJO Customer

4.2 Benutzerdefiniertes Binding

Das benutzerdefinierte Binding von Swing-Komponenten kommt dann zum Einsatz wenn man eigene Komponenten in Swing entwickelt. Dies betrifft zum Beispiel Textfelder, welche für bestimmte Zahlenformate ausgerichtet sind oder Datumsauswahldialoge, etc. Spring-RCP führt bietet die Möglichkeit das Binding

auf Datentyp- und Propertyebene zu definieren. Um nun die benutzerdefinierten Komponenten einzuführen muss die SwingBinderSelectionStrategy geändert werden. Abbildung 4.6 auf 19 zeigt das UML Diagramm. In der benutzerdefinier-

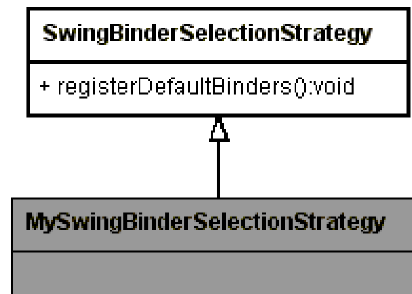

Abbildung 4.6: Benutzerdefinierte Swing Selection Strategy

ten MySwingBinderSelectionStrategy wird nun ein Mapping durchgeführt, welche Datentypen mit welchem Binder gemapped werden. Die abgeleitete Klasse muss nur die Methode registerDefaultBinders überschreiben. Der Code 4.7 auf 19 zeigt den passenden Code.

```
protected void registerDefaultBinders() {
 registerBinderForPropertyType(Integer.class,
    new RightJustifiedTextComponentBinder());
 registerBinderForPropertyType(double.class,
    new RightJustifiedTextComponentBinder());
 registerBinderForPropertyType(float.class,
    new RightJustifiedTextComponentBinder());
 registerBinderForPropertyType(int.class,
    new RightJustifiedTextComponentBinder());
 registerBinderForPropertyType(Date.class,
    new DatePickerBinder());
 super.registerDefaultBinders();
}
```

Abbildung 4.7: Binder registrieren

Die Methode *doBind* muss von jedem Binder implementiert werden und ist für das Binding zwischen dem Attribut des FormModel und der grafischen Komponente zuständig. In diesem Codebeispiel wird ein JTextField mit rechtsbündiger Ausrichtung definiert. Am Schluss wird lediglich die modifizierte JTextField-Komponente an den TextComponentBinder weitergegeben, der die restliche Arbeit übernimmt.

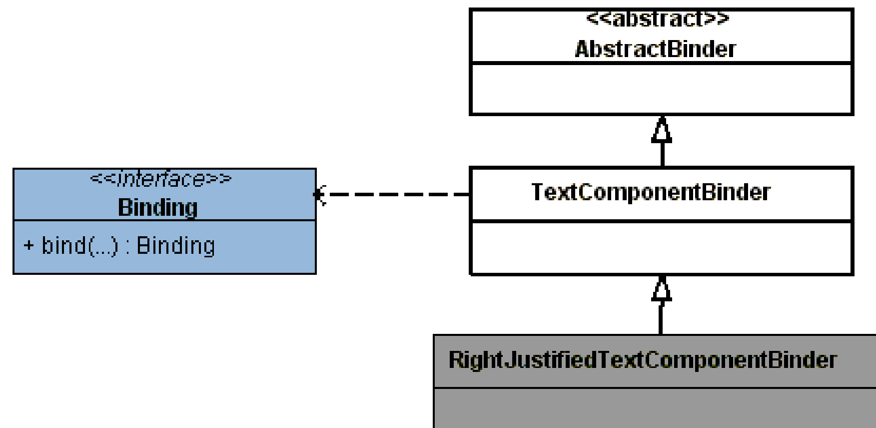

Abbildung 4.8: Benutzerdefinierter Binder

```
protected Binding doBind(JComponent control,
          FormModel formModel,
          String formPropertyPath,
            Map context) {
  //Integer Werte in einem Textfeld immer rechtsbündig anzeigen
  JTextField textComponent = (JTextField) control;
  textComponent.setHorizontalAlignment(JTextField.RIGHT);
  textComponent.setColumns(0);
  return super.doBind(textComponent,
                      formModel,
                      formPropertyPath,
                      context);
}
```

Abbildung 4.9: Beispielcode für einen rechtsbündigen Binder

5 Warum Spring RCP einsetzen

Am Schluss stellt sich nun die Frage, warum soll man gerade Spring RCP für seine zukünftigen GUI basierten Projekte einsetzen. Es gibt bereits eine große Anzahl an GUI Frameworks für Java Applikationen, doch der Erfolg des Springframeworks spricht für sich. Da es sich beim Spring Rich Client Project um ein Subprojekt der Spring-Gruppe handelt, kann man die Stärken von beiden Frameworks nutzen. Anhand eines kurzen Beispiels soll demonstriert werden welchen enormen Vorteil der Einsatz von Srping RCP in einem Projekt hat. Als Anwendungsfall nehmen wir das Binding zwischen Objekten und Formularelementen her, welches eines der stärksten Features in Spring RCP ist. Zu den wichtigen Features zählen auch die Master-/Detailansicht, das gesamte Window-Handling, usw.
In unserem Beispiel wird die herkömmliche Weise, wie man das Binding in Swing-Anwendungen durchführt, mit jener von Spring RCP gegenübergestellt. Als Basis verwenden wir das Objekt License, siehe Abbildung 5.1.

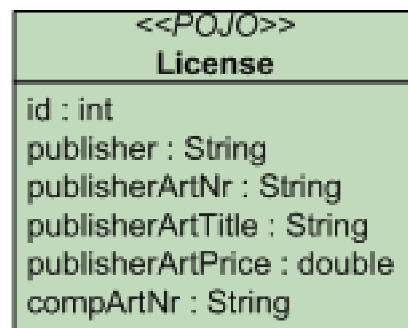

Abbildung 5.1: Beispielobjekt License

5.1 Binding in Swing-Anwendungen

Der erste Schritt besteht in der Erstellung einer passenden GUI für unser License-Objekt. Dies kann man sehr einfach mit einem Designer machen, der im Hintergrund den passenden Java Code wegschreibt. Der entstandene Code ist jedoch von Designer zu Designer unterschiedlich, was beim Austausch schon mal zu Problemen führen kann. Abbildung 5.2 zeigt eine einfache Erstellung einer GUI. Mit diesem Source werden jediglich die Komponenten für die Eingabe angezeigt unter Verwendung eines Standard-Layout Manager. Abbildung 5.3 demonstriert das Binding zwischen dem License-Objekt und den Swing-Komponenten. Die Methode *displayLicenseData* bekommt das License-Objekt und weist den Komponenten die einzelnen Werte zu und die Methode *getLicenseData* liefert das License-Objekt mit den aktuellen Formularwerten. Bei Objekten mit geringer Attributanzahl ist das nicht so schlimm, doch bei komplexen Objekten, kann dies schnell zu einer langweiligen Schreibarbeit werden. Denn man muss immer in zwei Richtungen denken:

- Objekt nach Formularkomponente
- Formularkomponente nach Objekt

```
public JPanel getForm(){
  JTextField txf_id = new JTextField();
  JTextField txf_publisher = new JTextField();
  JTextField txf_publisherTitle = new JTextField();
  JTextField txf_publisherArtPrice = new JTextField();
  JTextField compArtNr = new JTextField();

  txf_publisherArtPrice.setHorizontalAlignment(JTextField.RIGHT)

  JPanel form = new JPanel();
  form.add(txf_id);
  form.add(txf_publisher);
  form.add(txf_publisherArtNr);
  form.add(txf_publisherArtTitle);
  form.add(txf_publisherArtPrice);
  form.add(txf_compArtNr);
}
```

Abbildung 5.2: Formularkomponen

```
public void displayLicenseData(License license){
  txf_id.setText(Integer.toString(license.getId()));
  txf_publisher.setText(license.getPublisher());
  txf_publisherTitle.setText(license.getPublisherTitle());
  txf_publisherArtPrice.setText(Float.toString(
      license.getPublisherArtPrice()));
  txf_compArtNr.setText(license.getCompArtNr());
}

public License getLicenseData(){
  License license = new License();
  license.setId(Integer.parseInt(txf_id.getText()));
  license.setPublisher(txf_Publisher.getText());
  license.setPublisherTitle(txf_PublisherTitle.getText());
  license.setPublisherartPrice(Float.parseFloat(
      txf_Publisher.getText()));
  license.setCompArtNr(txf_compArtNr.getText());
  return license;
}
```

Abbildung 5.3: Binding zwischen Objekt und Komponenten

5.2 Binding in Spring RCP

Die entscheidenden Komponenten beim Binding und Erstellen von Formularen in Spring RCP sind in Abbildung 5.4 und 5.5 ersichtlich. Grundlage für das Binding bildet das Formmodell, welches basierend auf einem Objekt vom Framework erstellt wird. Dieses Modell bekommt als Parameter das Objekt, in unserem Fall das License-Objekt. Dieses Modell wird danach an das GUI Formular über den Konstruktor übergeben. Somit weiß das GUI-Formular mit welchem Objekt es verbunden ist (über das Formmodell).
Abbildung 5.5 demonstriert das Erstellen der GUI mit gleichzeitigem Binding. Das Binding-Konzept ist vergleichbar mit jenem von Web-Frameworks. Die Zeile *fb.add(publisher)* erstellt zum Beispiel ein JTextField und dieses Feld ist mit dem Attribut publisher des License-Objektes verbunden. Es werden somit die get und set-Methoden des Objektes in Anspruch genommen.
Als positiver Nebeneffekt ist die klare Darstellung des Formularaufbaus, was es uns ermöglicht ohne Formulardesigner leistungsfähige Formulare zu gestalten.

```
<bean id="license"
      class="at.demolsky.lms.license.bo.License"
      singleton="false"/>

<bean id="licenseFormModel"
      class="org.springframework.binding.
             form.support.DefaultFormModel"
      singleton="false">
  <constructor-arg index="0">
    <ref bean="license"/>
   </constructor-arg>
</bean>

<bean id="licenseForm"
      class="at.demolsky.lms.license.ui.LicenseForm"
      singleton="true">
  <constructor-arg index="0">
    <ref bean="licenseFormModel"/>
   </constructor-arg>
   <constructor-arg index="1">
     <value>generalPage</value>
   </constructor-arg>
</bean>
```

Abbildung 5.4: Konfiguration

```
protected JComponent createFormControl() {
  //Deklaration
  TableFormBuilder fb = new TableFormBuilder(getBindingFactory());
  fb.add("publisher");
  fb.add("publisherArtNr");
  fb.row();
  fb.add("publisherArtTitle");
  fb.add("publisherArtPrice");
  fb.row();
  fb.add("compArtNr");
  fb.getLayoutBuilder().cell();
  fb.row();
  return fb.getForm();
}
```

Abbildung 5.5: Binding in Spring RCP

6 Fazit

Bis dato gibt es noch kein offizielles Release von Spring RCP, sondern ist nur über ein CVS Repository zugänglich. Die Funktionsweise des Frameworks wird durch das mitgelieferte Beispiel PetClinic demonstriert. Auch für Dokumentation wurde bereits gesorgt, welche aber an manchen Stellen sehr dürftig ausfällt. Da es noch kein Release von Spring RCP gibt muss man sich der Entwickler immer auf laufende API Änderungen einstellen. In manchen Bereichen ist das Framework schon recht ausgereift, wie zum Beispiel das Action-Framework, Window-Management oder das Binding. Die Validierung weist noch einige Fehler auf.
Für die Zukunft wird sich Spring RCP mit Sichereit als ein sehr mächtiges Swing-Framework durchsetzen. Hat man vor seine Swing-Applikation mit dem Applikationsframework Spring zu erstellen, dann ist man ganz guter Dinge wenn man auch das Spring RCP einsetzt.
Weitere Swing-Frameworke im OpenSource Bereich:

- Eclipse RCP
- Genuine

Abbildungsverzeichnis

2.1 Komponenten von Spring RCP 4
2.2 Ausschnitt aus der application-context.xml 6
2.3 Ausschnitt aus der commands-context.xml 7
2.4 Ausschnitt aus der application-context.xml 8
2.5 Ausschnitt aus der application-context.xml 9
2.6 Ausschnitt aus der application-context.xml 9
2.7 Ausschnitt aus der application-context.xml 10

3.1 GUI Hierarchie in Spring RCP 11
3.2 Erstellen einer View 12
3.3 Beispiel einer View 13
3.4 Erstellen einer Form 13
3.5 Beispielcode für die Erstellung einer einfachen Form 14
3.6 GUI der erstellten Form 14
3.7 UML für Master/Detailform 15
3.8 GUI für Master/Detailform 15

4.1 FormModel 16
4.2 Beispiel für eine Hierarchie eines FormModel 17
4.3 Beispiel für ein einfaches Binding 17
4.4 Swing Binding Factory 18
4.5 POJO Customer 18
4.6 Benutzerdefinierte Swing Selection Strategy 19
4.7 Binder registrieren 19
4.8 Benutzerdefinierter Binder 20
4.9 Beispielcode für einen rechtsbündigen Binder 20

5.1 Beispielobjekt License 21
5.2 Formularkomponen 22
5.3 Binding zwischen Objekt und Komponenten 23
5.4 Konfiguration 24
5.5 Binding in Spring RCP 24

Tabellenverzeichnis

4.1 Standard Swing Binding Factory 18

Literaturverzeichnis

[1] Spring Rich Client. http://www.springframework.org/richclient.

[2] Genuine A client framework for Java Swing Applications. http://genuine.sourceforge.net/.

[3] Eclipse Rich Client Project. http://www.eclipse.org/rcp.

[4] Springforum. http://forum.springframework.

[5] Springframework. http://www.springframework.org.