Branding einer Rich-Client-Platform-Applikation

Look good – feel good!

VON MARTIN LIPPERT UND BERND KOLB

Mit Eclipse 3.0 hat ein enorm mächtiges Application Framework das Licht der Welt erblickt: die Eclipse Rich Client Platform. Mittlerweile basieren bereits einige Projekte auf dieser neuen Plattform. Viele neue Projekte entscheiden sich nicht zuletzt wegen des nativen Look & Feels des unter Eclipse liegenden UI-Frameworks SWT für die Rich Client Platform. Ein weiterer wichtiger Aspekt ist das so genannte Branding einer Applikation. Dazu gehören Dinge wie das Programm-Icon, ein About-Dialog oder auch der Splash Screen, welcher während des Startens der Applikation angezeigt wird.

In diesem Artikel wollen wir zeigen, was man tun muss, damit die von uns gewünschten Icons und Programmnamen überall erscheinen. Danach werden wir den About-Dialog auf unsere Bedürfnisse anpassen. Abschließend ändern wir noch den Splash Screen. Nebenbei exportieren wir unsere Anwendung so, dass sie sofort ausführbar ist. Wir werden in unserem Artikel die Version 3.1 M5a von Eclipse verwenden. Dennoch ist das hier beschriebene Branding einer RCP-Applikation auch größtenteils bereits bei den Eclipse-Versionen ab 3.0 möglich.

Eine Applikation entsteht

Als Einstieg in die RCP wollen wir kurz zeigen, wie eine Applikation erstellt wird. Auf dieser Applikation wird später unser Branding basieren. Eine Applikation muss das Interface *org.eclipse.core.runtime.IPlatformRunnable* aus *org.eclipse.core.runtime* implentieren. Das Interface definiert eine Methode: *public Object run(Object args) throws Exception*. Diese ist mit der Main-Methode einer Java-Applikation zu vergleichen (Listing 1).

Zunächst wird ein Display erstellt. Auf diesem Display wird dann mithilfe eines so genannten *WorkbenchAdvisor* die Workbench erstellt. Im *WorkbenchAdvisor* wird z.B. angegeben, welche Menüs vorhanden sein sollen oder ob es eine Toolbar oder eine Statuszeile geben soll. Beim Schließen der Workbench wird normalerweise ein Integer-Objekt zurückgegeben. Je nach Wert wird die Anwendung neu gestartet oder einfach beendet. Zum Schluss darf nicht vergessen werden, das erzeugte Display wieder zu zerstören und die Ressourcen an das Betriebssystem zurückzugeben.

Um unsere Applikation zu starten, müssen wir in der *plugin.xml* den Extension Point *org.eclipse.core.applications* erweitern:

```
<extension
  id="application"
    point="org.eclipse.core.runtime.applications">
  <application>
    <run class="brandingDemo.rcp.Application"/>
  </application>
</extension>
```

Listing 1

Quellcode einer Applikation

```
public Object run(Object args) throws Exception {
  Display display = PlatformUI.createDisplay();
  try {
    int returnCode =
      PlatformUI.createAndRunWorkbench(display,
        new ApplicationWorkbenchAdvisor());
    if (returnCode == PlatformUI.RETURN_RESTART) {
      return IPlatformRunnable.EXIT_RESTART;
    }
    return IPlatformRunnable.EXIT_OK;
    } finally {
      display.dispose();
  }
}
```

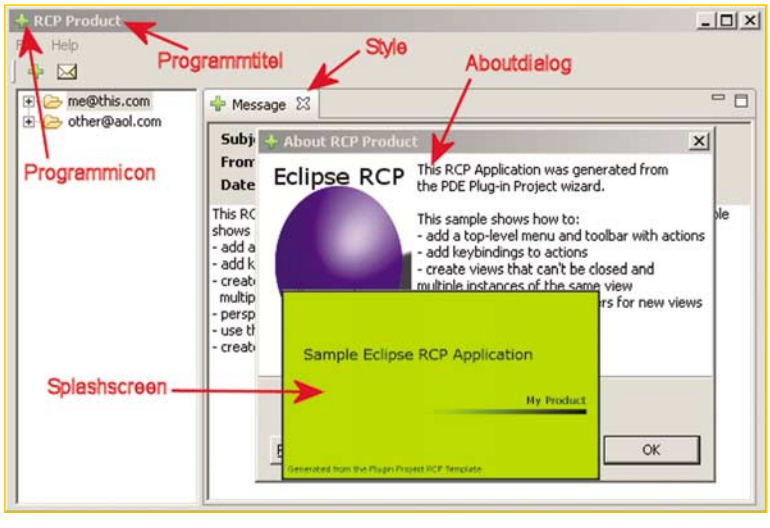

Abb. 1: Überblick über einige mögliche Anpassungen eines Produkts

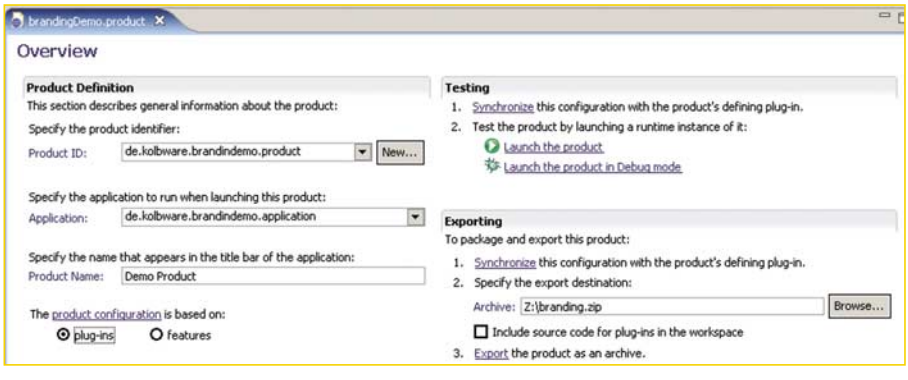

Abb. 2: Erste Seite des Product Configuration Editor – ein Überblick

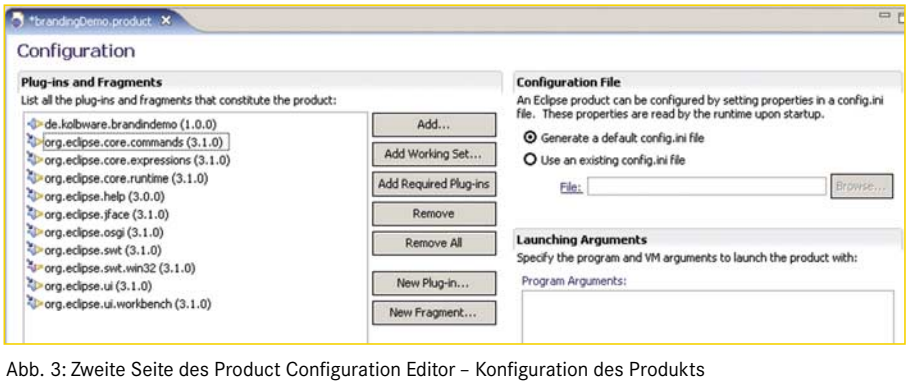

Abb. 3: Zweite Seite des Product Configuration Editor – Konfiguration des Produkts

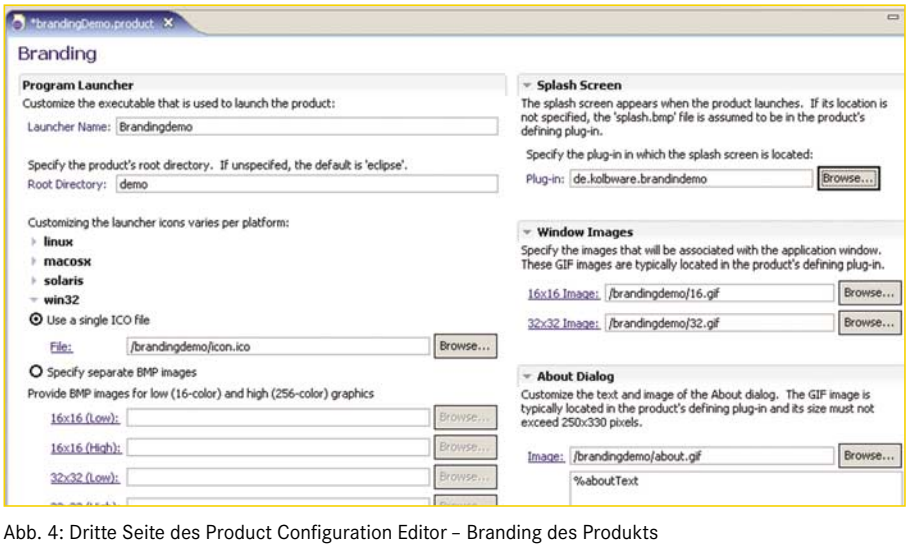

Abb. 4: Dritte Seite des Product Configuration Editor – Branding des Produkts

Mit dem Parameter *-application <pluginID>.<applicationID>*, in unserem Fall also *de.kolbware.brandindemo.application* kann die Anwendung gestartet werden.

Am Anfang steht das Produkt

Für ein Branding reicht eine Applikation nicht aus; es ist immer an ein so genanntes Produkt gebunden. Ein solches wird erstellt, in dem man den Extension Point Produkt aus *org.eclipse.core.runtime* „implementiert". Ein Produkt besteht aus einigen Konfigurationseinstellungen sowie dem Verweis auf die Applikation (*org.eclipse.core.runtime.applications*), die mit dem Produkt gestartet werden soll. Seit Eclipse 3.1 M5 gibt es einen Editor, der uns bei der Erstellung des Brandings behilflich ist. Wir werden diesen Wizard benutzen und uns parallel anschauen, welche Auswirkungen dies auf unsere *plugin.xml* und andere Dateien hat.

In unserem Plug-in-Projekt legen wir mithilfe des Product Configuration Wizards eine neue Datei mit der Endung *product* an. In dieser Datei werden wir nun unser Eclipse-Produkt konfigurieren. Wir wählen die Option CREATE A CONFIGURATION FILE WITH BASIC SETTINGS. Daraufhin öffnet sich ein dreiseitiger Multi-Page-Editor.

Zuerst müssen wir die Produkt-ID wählen, für die wir das Branding erstellen wollen. Da wir bisher noch kein Produkt haben, erzeugen wir ein neues, indem wir den entsprechenden Knopf im Editor drücken. Dazu müssen wir das Plug-in wählen, in dem das Produkt definiert werden soll, sowie die ID des Produkts. Danach wählen wir die gewünschte Applikation aus. Bei Abschluss des Wizards wird Folgendes in der *plugin.xml* hinzugefügt:

```
<extension id="demo"
  point="org.eclipse.core.runtime.products">
 <product application="BrandingDemo.application"/>
</extension>
```

Wir erweitern jetzt also den Extension Point *products*. Zu beachten ist, dass die ID des Produkts auf oberster Ebene definiert wird und nicht wie bei den meisten anderen Extension Points erst im Produkt selbst. Auf unserem *Product*-Element fehlt noch ein weiteres, benötigtes Attribut: *Name*. Dieses wird gesetzt, wenn wir in unserem Konfigurationseditor das Feld *Product Name* füllen. Dieser Name wird später in der Titelzeile unserer Applikation erscheinen. Um unsere Einstellungen in die *plugin.xml* zu übernehmen, klicken wir in der rechten unteren Hälfte unseres Editors auf SYNCHRONIZE. Damit wird die Konfiguration (und damit die später generierte *config.ini*) unserer Anwendung mit der *plugin.xml* abgeglichen.

Bevor wir unser Product zum ersten Mal testen können, müssen wir noch bestimmen, aus welchen Plug-ins oder Features das Produkt bestehen soll. In diesem Artikel geben wir der Einfachheit halber die benötigten Plug-ins an und verzichten auf Features. In einer größeren Applikation sollten jedoch Features gewählt werden. Diese Option spiegelt sich später beim Export des Produkts wieder. Nur wenn das Produkt auf Features basiert, können

wir den Update-Mechanismus von Eclipse verwenden, um später einmal neue Versionen des Produkt zu verteilen.

Auf der zweiten Seite des Editors wählen wir unser Plug-in und drücken mit ADD REQUIRED PLUGINS alle benötigten Plug-ins hinzu. Jetzt können wir unsere Applikation das erste Mal starten. Dazu klicken wir auf der ersten Seite des Editors in der Sektion Testing auf LAUNCH THE PRODUCT. Das Fenster öffnet sich, doch leider passt der Titel unserer Applikation noch nicht. Um dies zu ändern, gehen wir in unseren Workbench Advisor und setzten in der Methode *preWindowOpen* als Titel den Name des Produkts.

```
public void preWindowOpen() {
 IWorkbenchWindowConfigurer configurer =
                                  getWindowConfigurer();
 //...
 configurer.setTitle(org.eclipse.core.runtime.Platform.
  getProduct().getName());
}
```

Nach einem Neustart ist das Problem behoben. Durch dieses Vorgehen ist es möglich, ein und dieselbe Applikation für unterschiedliche Kunden anders aussehen zu lassen. Es sind dazu keine Änderungen im Code nötig. Es reicht aus, die entsprechende *plugin.xml*-Datei anzupassen.

Durch das Drücken des LAUNCH THE PRODUCT-Links wurde eine Launch Configuration angelegt. Diese können wir uns unter RUN | RUN… ansehen. Dabei ist zu sehen, dass im Tab MAIN im mittleren Feld (PROGRAM TO RUN) nun nicht mehr eine Applikation gestartet wird, sondern unser Produkt.

Das Window Image und der About-Dialog

Auf der letzten Seite des Editors können die Programm-Icons sowie das *aboutImage* und der *aboutText* gesetzt werden. Diese Angaben werden beim Speichern des Editors als Properties des Produkts in der *plugin.xml* eingetragen.

```
<product
   application=" BrandingDemo.application "
   name="Hello World">
  <property
    name="windowImages"
    value="16x16.gif,32x32.gif"/>
  <property
    name="aboutImage"
    value="about.gif"/>
  <property
    name="aboutText"
    value="This is my branded product :-)"/>
</product>
```

Der *about*-Dialog kann in das Menü eingebunden werden, indem die Workbench Action *About* dem Menü im *ActionBarAdvisor* hinzugefügt wird (Listing 2).

Splash Screen

Bevor wir unser Product das erste Mal exportieren, legen wir fest, in welchem Plug-in unser Splash Screen zu finden ist. Dabei ist darauf zu achten, dass keine Datei angegeben wird, sondern nur ein Pfad zu dem Plug-in, in dem unserer Splash Screen liegt. Das Bild muss dabei vom Typ BMP sein und auf den Namen *splash* lauten. Der Name sowie das Dateiformat sind nicht wählbar. Abschließend machen wir dann auf der letzten Seite des Editors Angaben über den Launcher. Wir können dort den Namen und die Icons für das Executable

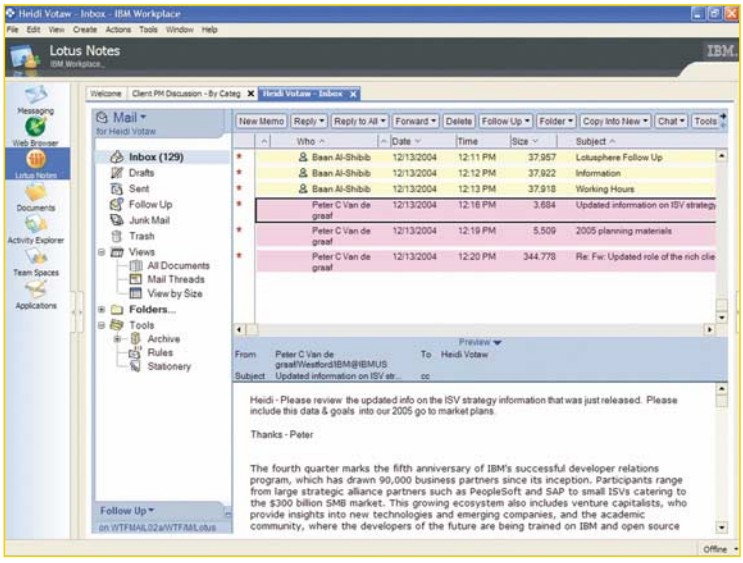

Abb. 5: Nichts mehr von Eclipse zu sehen – IBM Lotus Workplace

Listing 2

About Action in ein Menü einhängen

```
protected void makeActions
                    (IWorkbenchWindow window) {
  about = ActionFactory.ABOUT.create(window);
}

protected void fillMenuBar(IMenuManager menuBar) {
  //..
  MenuManager aMenu = new MenuManager("A Menu");
  menuBar.add(aMenu);
  //..
  aMenu.add(about);
  //..
}

public void dispose() {
  about.dispose();
  super.dispose();
}
```

Listing 3

Generierte *config.ini* für unser Produkt

```
osgi.bundles=de.kolbware.brandingDemo,
  org.eclipse.core.commands,
  org.eclipse.core.expressions,
  org.eclipse.core.runtime@2\:start,
  org.eclipse.help,
  org.eclipse.jface,
  org.eclipse.swt,
  org.eclipse.swt.win32,
  org.eclipse.ui,
  org.eclipse.ui.workbench
osgi.bundles.defaultStartLevel=4
eclipse.product= de.kolbware.brandingDemo.demo
osgi.splashPath=
  platform\:/base/plugins/ de.kolbware.brandingDemo
```

angeben, mit dem unser Produkt gestartet werden kann. Zusätzlich geben wir noch das Basisverzeichnis an, in das unsere Anwendung später extrahiert werden soll.

Export und Installation des Produkts

Auf der ersten Seite des Editors wählen wir das Archiv, in das unsere Applikation gepackt werden soll, und exportieren diese, indem wir den entsprechenden Hyperlink anklicken. Ein Blick unter die Haube verrät uns, was der Wizard beim Export noch zusätzlich für uns erledigt hat: Nachdem wir die Zip-Datei entpackt haben, werfen wir einen Blick in die Datei *config.ini* (Listing 3) im Ordner *configuration*. Neben einigen OSGi-Einstellungen für den Start finden sich in dieser Datei die Angaben über das zu startende Produkt sowie den Pfad zu unserem Splash Screen.

Der erste Eintrag in der *config.ini* gibt an, welche Plug-ins benötigt werden und damit beim Start geladen werden müssen. Danach werden das zu startende Produkt und der dafür notwendige Splash Screen festgelegt. Das Ausführen des Launchers startet unser Produkt mit dem von uns erzeugten Branding.

Eine weitere Möglichkeit, unsere Anwendung anzupassen, bietet das Presentation API, das es seit Eclipse 3.0 gibt. Mit diesem API kann das Layout der gesamten Workbench verändert werden. Ein Beispiel dafür ist z.B. Lotus Workplace von IBM. Leider gibt es dazu noch keine ausführliche Dokumentation. Wer sich für dieses API interessiert, kann sich aber die Implementierung des 2.1 Theme innerhalb von Eclipse 3.x ansehen.

Aber auch ohne das Presentation API bietet Eclipse kleinere Anpassungsmöglichkeiten, die ohne viel Aufwand machbar sind. Wenn wir in unserem Plug-in, in dem die Applikation definiert wurde, eine Datei namens *plugin_customization.ini* öffnen (Listing 4), können wir u.a. das Layout der Taps von eckig auf rund ändern.

Wie zu sehen ist, lässt sich mit wenigen Handgriffen eine Anwendung so ändern, dass sie sich in das Corporate Design einer Firma einfügt. Je nach Ausbau des Brandings ist es nicht mehr möglich zu erkennen, dass es sich um eine Eclipse-basierte Anwendung handelt.

Listing 4

Datei *plugin_customization.ini aus org.eclipse.platform*

```
# plugin_customization.ini
# sets default values for plug-in-specific preferences
# keys are qualified by plug-in id
# e.g., com.example.acmeplugin/myproperty=myvalue
# java.io.Properties file (ISO 8859-1 with "\" escapes)
# "%key" are externalized strings defined in plugin_
                        customization.properties
# This file does not need to be translated.

# Property "org.eclipse.ui/defaultPerspectiveId"
# controls the perspective that the workbench
# opens initially
org.eclipse.ui/defaultPerspectiveId=
                    org.eclipse.ui.resourcePerspective

# new-style tabs by default
org.eclipse.ui/SHOW_TRADITIONAL_STYLE_TABS=false

# put the perspective switcher on the top right
org.eclipse.ui/DOCK_PERSPECTIVE_BAR=topRight
```

Bernd Kolb ist freiberuflicher Berater und Coach. Er ist (Mit-)Autor von diversen Artikeln sowie eines Buches und Sprecher auf Konferenzen. Seine Schwerpunkte liegen auf modellgetriebener Softwareentwicklung sowie Eclipse. Kontakt: b.kolb@kolbware.de.

Martin Lippert ist Senior-IT-Berater bei it-agile. Er arbeitet dort als Coach und Berater für agile Softwareentwicklung, Refactoring und Eclipse-Technologie. Kontakt: martin.lippert@it-agile.de.