Erweiterung des Einsatzgebiets

RCP-Hilfesystem und Enterprise-Wikis

Am Anfang unseres Projekts stand der Wunsch unserer Fachabteilung, ihre im Wiki gesammelten Informationen und Hilfetexte direkt mit unserer RCP-Anwendung zu verheiraten. Wir haben reflexartig entgegnet: „Kein Problem, mit Eclipse geht so etwas leicht!“. Wie sich im Projektverlauf herausstellte, ist dieses Unterfangen doch nicht ganz so trivial, wie es anfangs schien. Viele Kleinigkeiten standen dem eigentlich klaren Ziel im Weg, und die vorhandene Eclipse-Dokumentation schwieg sich zu diesem wichtigen Thema doch beharrlich aus. Wir möchten in diesem Artikel unseren Weg zu einer gar nicht mal so komplexen Lösung skizzieren und hoffen, die erwähnte Dokumentationslücke etwas zu schließen.

von Holger Grosse-Plankermann und Jörg Meister

Der Artikel wird sich mit der Erweiterung und Anpassung des Eclipse-Hilfesystems im Hinblick auf eine vorhandene Infrastruktur zur Dokumentation in Form eines Wikis befassen. Entstanden ist der Sourcecode in einem Projekt im Enterprise-Umfeld. Ein Deployment-Prozess ist im Enterprise-Umfeld oft mit hohem Aufwand verbunden. Es ist somit nicht unbedingt ratsam, für Änderungen in Hilfetexten einen solchen Prozess durchzuführen. Ein Abtrennen der Hilfetexte vom Programm ist somit notwendig. Anforderung war, dass Hilfetexte zu bestimmten Bereichen der Anwendung auf Knopfdruck verfügbar sein sollen. Die kontextsensitive Hilfe von Eclipse bietet hierfür bereits sehr gute Möglichkeiten. Da die Dokumentation im Wiki entwickelt wird, entstand der Wunsch, auf diese aus der Anwendung heraus zugreifen zu können. Im Idealfall hätte man somit eine Trennung von Hilfetexten und RCP-Anwendung, sodass die Texte unabhängig von der Anwendung pflegbar sind. Dies ist out-of-the-Box nicht direkt möglich. Zunächst haben wir versucht, Eclipse-Bordmittel und die mächtige Konfiguration des Hilfesystems zu verwenden, um uns diesen Anforderungen zu nähern. Das wird in den Punkten 1 und 2 beschrieben. Die beschriebenen Wege sind durchaus gangbar, haben uns allerdings nicht vollständig zufrieden gestellt. Schlussend-

lich haben wir einen programmatischen Ansatz gewählt, wie er in Punkt 3 skizziert ist. Dieser Artikel ist eher Erfahrungsbericht als Tutorial. Wir gehen davon aus, dass Sie dem hier vorkommenden Sourcecode leicht folgen können, wenn Sie sich schon ein wenig mit der RCP-Entwicklung beschäftigt haben und auch mit dem Eclipse-Hilfesystem in Berührung gekommen sind. Als gute Einführung in das Thema sei unter anderem auf [1] verwiesen.

1. Eingebaute Hilfe

Als erster Weg wurde die Standardhilfe verwendet, wie sie in diversen Artikeln und Büchern [1-5] beschrieben wird. Diese umfasst die aus der Eclipse IDE bekannte Help-Sidebar. Die Help-Sidebar kann eine Zusammenfassung des Hilfethemas und Links auf das Wiki enthalten. Das bedeutet, dass ein Nutzer bei Druck auf F1 auf einem Control zunächst diese Sidebar sieht und dann den Link auf das Wiki auswählen kann. Als Basis benutzten wir für diesen Artikel eine leicht angepasste Version des RCP-Mail-Templates. Dieses ist über den Wizard für ein neues Plugin-Project zu erstellen. Das Einbinden des Eclipse-Hilfesystems ist in der Literatur bereits ausführlich behandelt worden. Daher beschreiben wir an dieser Stelle den Weg nur schematisch (Kasten: „Die Beispielprojekte“).

Das RCP-Mail-Template wurde so abgeändert, dass es ein Help-Plug-in enthält. Als Basis kann man hier ebenfalls den Wizard für ein neues Plugin-Project verwenden, indem man das Template Plugin With Sample HelpContent verwendet. Wir haben allerdings alle für unseren Zweck überflüssigen TOCs entfernt. Ausgehend von diesem RCP-Mail-Template haben wir einige Anpassungen vorgenommen. Für unseren Kunden war es ausreichend, dass es pro Bildschirmseite nur einen Hilfekontext gibt. Hier bietet es sich an, den Kontext an das oberste Composite (Top-Level-Composite) der Form zu binden. In unserem Beispiel wurde in *View.java* die Zeile *PlatformUI.getWorkbench().getHelpSystem().setHelp(top,"beispielApplikation.app.form_top")* hinzugefügt und somit *HelpSystem* und *Composite* verbunden. Nun wurden noch in der *contexts.xml* Kontexte angelegt:

```
<contexts>
  <context id="form_top" title="Eclipse">
    <description>Dies ist ein Link zu Eclipse in Wikipedia</description>
    <topic href="http://de.wikipedia.org/wiki/Eclipse_%28IDE%29"
                                                    label="Eclipse"/>
  </context>
</contexts>
```

Die Kontexte werden anschließend mittels der *plugin.xml* verdrahtet:

```
<plugin>
  <extension
      point="org.eclipse.help.contexts">
    <contexts
        file="contexts.xml"
        plugin="beispielApplikation.app">
    </contexts>
    <contexts
        file="contexts_missing.xml"
        plugin="org.eclipse.ui">
    </contexts>
  </extension>
</plugin>
```

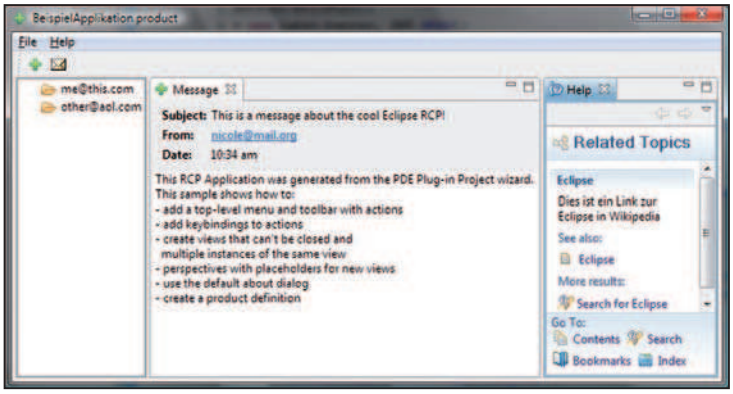

Abb. 1: Das Standardverhalten des Eclipse Help UI am Beispiel des Mail-Templates

Wie man hier sehen kann, gehen wir in unserem Beispiel noch ein kleines Stückchen weiter und richten einen Fallback-Kontext ein, den so genannten *Context_missing*. Dieser wird benutzt, wenn Eclipse bei Druck auf F1 keinen passenden Kontext findet. Er ist somit prädestiniert dafür, auf die Hilfestartseite zu verweisen. Selektiert man nun das Textfeld mit der E-Mail und drückt auf F1, öffnet sich die Sidebar mit dem Link auf Wikipedia (**Abb. 1**). Fokussiert man den linken Navigationsbereich und drückt F1, greift der *Context_missing*.

Die Standardhilfe bietet darüber hinaus weitere Möglichkeiten, zum Beispiel die eingebaute Suche und die Indizierung, die allerdings in unserem Projekt nicht er-

Die Beispielprojekte

Um dem Text leichter folgen zu können, haben wir zwei Beispielprojekte erstellt. Diese befinden sich als ZIP-Datei unter [10] und auf der beiliegenden CD. Die ZIP-Datei enthält zwei Verzeichnisse:

- *Abschnitt1* enthält die Sourcen von Abschnitt 1,
- *Abschnitt3* enthält die Sourcen von Abschnitt 3.

Die Verzeichnisse enthalten jeweils zwei Projekte:

- *net.iksgmbh.beispielApplikation.app*
- *net.iksgmbh.beispielApplikation.help*

Das **help*-Plug-in enthält die Hilfetexte und das **app*-Plug-in die eigentliche Programmlogik. In einem echten Projekt würde man diese Aufteilung sicher noch feingranularer vornehmen. Die Projekte können Sie wie gewohnt über File | Import... laden. Beim Erstellen der Run Configuration ist darauf zu achten, dass auf dem Reiter Main unter Run a product die Product-ID *net.iksgmbh.beispielApplikation.app.product* ausgewählt ist und auf dem Reiter Plug-ins die beiden Plug-ins angehakt sind. Nach einem Klick auf Add required Plugins ist die Konfiguration abgeschlossen.

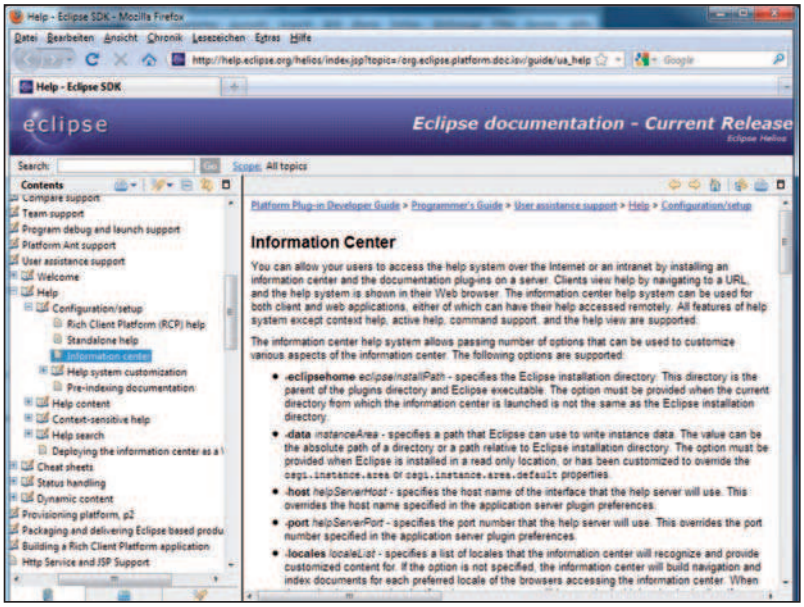

Abb. 2: Infocenter der Eclipse-Hilfe auf www.eclipse.org

wünscht waren. Zum einen sind sie schwer anpassbar, zum anderen sind viele Anwender ohne technischen Background von diesen Möglichkeiten oft schlichtweg überfordert. Dass der Nutzer nur über den Umweg des Hilfefensters zur Information im Wiki gelangt, wurde zunächst toleriert, richtig zufrieden war man mit der Lösung jedoch nicht. Eine Aktualisierung der Dokumentation ist hier ebenfalls schwieriger als nötig. Die betroffenen Kontexte müssen separat von der eigentlichen Dokumentation angepasst werden. Änderungen müssen dann auf irgendeine Art auf alle installierten Clients gelangen. Dies könnte über Featureupdates und den Eclipse-Bereitstellungsmechanismus p2 erledigt werden. Ein Update zu Programmstart ist sicherlich ein gangbarer Weg. Leider konnte p2 allerdings in unserem Projekt nicht eingesetzt werden, da eine bereits im Unternehmen vorhandene Softwareverteilung benutzt werden musste. Von diesem Umstand können sicherlich viele RCP-Entwickler ein Lied singen. Da wir p2 nicht einsetzen konnten, wäre bei Änderungen der Hilfetexte ein komplettes Deployment erforderlich gewesen. Dies war der Zeitpunkt, an dem wir uns nach Alternativen umschauen mussten.

2. Infocenter

Die zweite Idee war das Verwenden des Infocenters (**Abb. 2**). Das Infocenter ist eine abgespeckte Eclipse-Version, die mittels des eingebauten Jetty als Webserver fungiert [6]. Somit kann der Nutzer mit einem Webbrowser über Intra- oder Internet auf die Hilfeseiten zugreifen. Allerdings bietet das Infocenter nicht alle Features des Built-in-Help-Systems. Es fehlt die Unterstützung für *Context Help*, *Active Help* und *Command Support* [7]. Vorteil des Infocenters ist das separate Deployment des Hilfesystems auf einem Application-Server. Die Anwendung selber ist somit unabhängig von Hilfe-Updates. Eine Änderung der Hilfe kann also separat ausgeliefert werden und ist für alle Nutzer sofort verfügbar.

Das Infocenter kam den Anforderungen im Unternehmen schon recht nahe, da bei Änderungen an den Hilfetexten keine Neuauslieferung der Clientsoftware erforderlich ist. Der Nachteil, dass die Kontexthilfe vom Infocenter nicht out-of-the-Box unterstützt wird, war für die Nutzer allerdings so gravierend, dass wir diese Lösung verwerfen mussten.

3. Programmatische Anpassung des Hilfesystems

Nachdem uns die beiden Standardansätze nicht zufrieden stellten, fingen wir an, einen Weg zu suchen, das Verhalten des Hilfesystems stärker beeinflussen zu können. Die Internetrecherche war leider nicht sehr erfolgreich. So blieb (wie so häufig) nur die Möglichkeit, den Sourcecode zu analysieren und schrittweise zu durchlaufen. Erneut hilfreich beim Verständnis der Funktionsweise der Eclipse-Hilfe war hier die Möglichkeit des Tracings in der Run-Configuration, um herauszufinden, welche die am Hilfesystem beteiligten Klassen

Hilfearten in Eclipse RCP

Hier sind die unterschiedlichen Wege der Aktivierung der Eclipse-Hilfe zusammengefasst, die einem Eclipse-Entwickler zur Verfügung stehen, wenn er die Hilfe wie skizziert mit der Anwendung verbindet.

Begriff	Erklärung
Help	Hilfe, die beim Betätigen der Hilfetaste oder eines Menüpunkts erscheint. Üblicherweise öffnet sich dann die Hilfeübersicht in einem Fenster der RCP-Anwendung.
Context/Kontext	Umgebung, in der ein Hilfebereich gültig ist. Oft ein bestimmtes Control, das den Focus besitzt.
Context Help	Hilfe, die bei Druck auf die Hilfetaste (unter Windows: F1) den Kontext berücksichtigt.
Dynamic Context Help	Context Help, deren Kontexte dynamisch aufgebaut werden. So können beispielsweise einzelnen Elementen einer Combobox spezielle Hilfethemen zugeordnet werden. Nicht Thema dieses Artikels.
Active Help/Command Support	Möglichkeit, Code aus dem Inhalt der Hilfe aufzurufen.

Tabelle 1: Die Möglichkeiten der Hilfe in Eclipse RCP

und Plug-ins sind (**Abb. 3**). Als Einstieg boten sich die Plug-ins *org.eclipse.help* und *org.eclipse.help.ui* an.

Wir stießen so relativ schnell auf die Klasse *WorkbenchHelpSystem*. Dort findet sich mit dem *WorkbenchHelpListener* und der Methode *helpRequested(HelpEvent event)* der Einstiegspunkt in das Eclipse-Hilfesystem. Über die Methoden *displayHelp* und *getHelpUI* kamen wir dem Mysterium Eclipse-Hilfe mehr und mehr auf die Schliche. Die Standard-UI-Funktionalität der Hilfe verbirgt sich in der Klasse *DefaultHelpUI*. Das JavaDoc der Basisklasse *AbstractHelpUI* erklärte die Funktionsweise der Hilfe dann schließlich. Der UI-Teil der Hilfe ist über den Extension-Point *org.eclipse.ui.helpSupport* austauschbar. Wichtig ist, dass es nur *eine* Help-UI-Implementierung im System geben darf. Die Standardimplementierung ist also irgendwie zu entfernen.

Wie ist nun weiter vorzugehen? Zur Erinnerung: Die Aufgabe besteht darin, bei Druck auf F1 auf einem fokussierten Control einen Wiki-URL in einem Webbrowser aufzurufen. Wie wir gesehen haben, ist die Verwaltung des Hilfesystems (*WorkbenchHelpSystem*) getrennt von der Präsentation (*DefaultHelpUI*). Das spielt uns an dieser Stelle in die Karten: Die Funktionalität der Kontexte und das Finden von F1 sind schließlich durchaus erwünscht (Kasten: „Das F1-Problem“).

Als Ausgangspunkt für unsere Klasse *BrowserHelpUI* kopierten wir die Klasse *DefaultHelpUI* aus dem Plug-in *org.eclipse.help.ui* und entfernten allen für uns unnötigen Ballast. Es mussten noch die Plug-ins *org.eclipse.help* und *org.eclipse.help.base* als Dependency hinzugefügt werden. Übrig geblieben in *BrowserHelpUI* ist die innere Klasse *ExternalWorkbenchBrowser*, die einen kleinen Wrapper um die Browserfunktionalität darstellt, sowie einige Methoden, die wir entsprechend unseren Wünschen angepasst haben und im Folgenden beschreiben.

Die Methode *displayHelp* wird für das statische Aufrufen des Hilfesystems verwendet. Dieses ist in unserem Use Case ja durchaus erwünscht. An dieser Stelle sollte möglichst die Startseite des Wikis aufgerufen werden.

```
public void displayHelp() {
  displayHelpResource("http://de.wikipedia.org");
}
```

Die Methode *displayContext* wird eigentlich dafür verwendet, den übergebenen Hilfekontext an der ebenfalls übermittelten Position anzuzeigen. Die Position interessiert uns an dieser Stelle nicht, da die Hilfe außerhalb des Anwendungsfensters der Applikation dargestellt wird. Interessant ist hier der Zugriff auf die Konfiguration der Hilfekontexte. An dieser Stelle zur Erinnerung ein Ausschnitt aus einer *contexts.xml*-Datei:

```
<contexts>
  <context id="form_top" title="Eclipse">
    <description>Dies ist ein Link zu Eclipse in Wikipedia</description>
    <topic href="http://de.wikipedia.org/wiki/Eclipse_%28IDE%29"
                                                    label="Eclipse"/>
  </context>
</contexts>
```

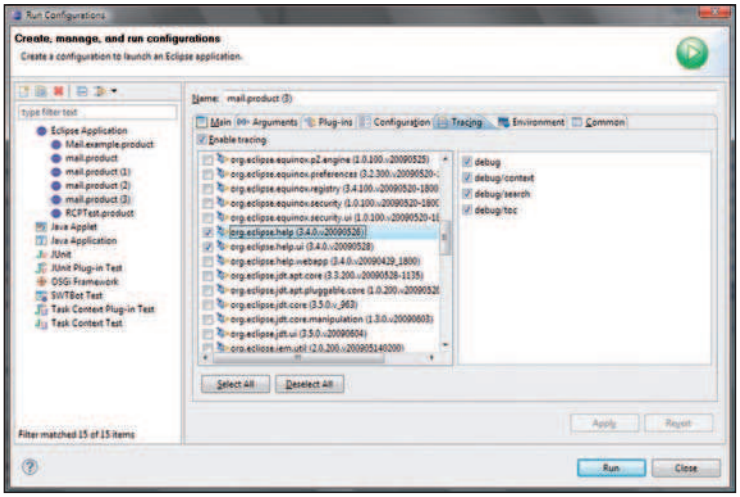

Abb. 3: An den Run Configurations hat man die Möglichkeit, Tracings einzurichten

Für das Hilfesystem ist hier zunächst nur die ID wichtig, auf die später im Code referenziert wird (siehe hierzu die Hilfeverknüpfung in *View.java* im ersten Abschnitt). Der *title* und die *description* sowie das *label* werden nur für die Darstellung in der Hilfe-Sidebar verwendet. Wir interessieren uns hier nur für den Knoten *<topic href = " "/>*. Hier kann zu jedem Topic ein URL hinterlegt werden. An dieser Stelle verknüpfen wir also den Kontext mit dem im Browser anzuzeigenden URL. Dabei muss man etwas aufpassen: Wie man deutlich im Code sieht, ist es einfach möglich, mehrere *relatedTopics* für einen Kontext zu hinterlegen. Unsere momentane Implementierung sieht vor, einfach alle zu öffnen. Wir empfehlen allerdings, es bei einem *relatedTopic* in der Konfiguration zu belassen und die Wikiseite mit entsprechenden Links auszustatten:

```
public void displayContext(IContext context, int x, int y) {
  IHelpResource[] relatedTopics = context.getRelatedTopics();
  for (int i = 0; i < relatedTopics.length; i++) {
    String href = relatedTopics[i].getHref();
    displayHelpResource(href);
  }
}
```

Die Methode *displayHelpResource* ist dann unspektakulär: „Reiche URL an den Browser-Wrapper weiter“:

```
public void displayHelpResource(String href) {
  try {
    externalWorkbenchBrowser.displayURL(href);
  } catch (Exception e) {
    throw new RuntimeException("Fehler beim Öffnen des Browsers", e);
  }
}
```

Das *DefaultHelpUI* bzw. dessen Oberklasse *AbstractHelpUI* bietet noch eine weitere Funktionalität an, die für

uns keinen Nutzen hat. Eine Suche ist über den Browser und im Wiki selber möglich, also ist das Implementieren der Methoden rund um die Suche überflüssig (*search* und *displaySearch*). Dynamic Context Help [8] ist von uns noch nicht umgesetzt worden. Es ist wahrscheinlich auch nicht sinnvoll, *Dynamic Context Help* mit dem Browser zu verbinden. Das Öffnen eines Browserfensters bei jedem Feuern eines *SelectionChanged*-Events einer Drop-Down-Box ist wahrscheinlich nicht praktikabel. Somit implementieren wir die Methode *displayDynamicHelp* nicht. Schlussendlich finden wir in der Klasse *DefaultHelpUI* noch eine ganze Reihe Methoden vor, die sich mit der Darstellung der Hilfe in der RCP-Anwendung befassen. Diese benötigen wir aufgrund unserer Anforderungen nicht. Alles in allem wird unsere resultierende Klasse *BrowserHelpUI* schön schlank. Unser neues *HelpUI* muss noch dem System bekannt gemacht werden. Dazu wird der Extension-Point *org.eclipse.ui.helpSupport* bedient:

```
<extension
      point="org.eclipse.ui.helpSupport">
   <config
      class="net.iksgmbh.beispielApplikation.app.BrowserHelpUI">
   </config>
</extension>
```

Das Plug-in *org.eclipse.help.ui* muss dann als Abhängigkeit entfernt werden, da es den Extension Point *org.eclipse.ui.helpSupport* blockiert. Wie bereits erwähnt, darf es davon nur eine Implementierung in der Applikation geben. Die Abhängigkeiten findet man im Manifest des *App*-Plug-ins. Auch aus der Run-Config ist sie zu entfernen, sofern vorhanden. Unser gestelltes Ziel ist somit erreicht: Das Hilfesystem ist nun vollständig vom Deployment-Zyklus unserer Applikation getrennt, und wir ermöglichen, wie in Teil 1 erwähnt, den Zugriff pro Top-Level-Composite auf eine Hilfeseite im Wiki.

Listing 1

```
public void displayContext(IContext context, int x, int y) {

  try {
    IWorkbenchWindow window =
                        PlatformUI.getWorkbench().getActiveWorkbenchWindow();
    Shell activeShell = PlatformUI.getWorkbench().getDisplay().getActiveShell();
    // Haben wir gerade eine Shell?
    if (window != null && activeShell != null &&
                                          activeShell.equals(window.getShell())) {
      if (window.getActivePage() != null) {
        Control control = window.getShell().getDisplay().getFocusControl();
        tryAndShowHelp(control);
      }
    }
  } catch (Exception e) {
    e.printStackTrace();
  }
}

private void tryAndShowHelp(Control control) throws Exception, IOException {
  // Abfrage des Help-Hooks
  String data = (String) control.getData("org.eclipse.ui.help");
  boolean found = false;
  if (data != null && !data.isEmpty()) {
    // HelpSystem nach dem Context befragen
    IContext context = HelpSystem.getContext(data);
    if (context != null) {

      IHelpResource[] relatedTopics = context.getRelatedTopics();
      for (int i = 0; i < relatedTopics.length; i++) {
        String href = relatedTopics[i].getHref();
        // Steckt hinter dem URL auch eine HTML-Seite?
        if (checkUrl(href)) {
          // Anzeige der Seite
          displayHelpResource(href);
          found = true;
          break;
        }
      }
    }
  }

  if (!found) {
    // Keine Hilfeseite gefunden, dann rekursiv die übergeordneten
    // Controls durchlaufen
    Composite parentControl = control.getParent();
    if (parentControl != null) {
      tryAndShowHelp(parentControl);
    } else {
      //letzter Versuch mit dem Context_Missing
      IContext context = HelpSystem.getContext("org.eclipse.ui.missing");
      if (context != null) {
        IHelpResource[] relatedTopics = context.getRelatedTopics();
        for (int i = 0; i < relatedTopics.length; i++) {
          String href = relatedTopics[i].getHref();
          displayHelpResource(href);
        }
      }
    }
  }
}

private boolean checkUrl(String href) throws HttpException, IOException {
  HttpClient client = new HttpClient();

  // Eine HTTP-Get-Methode erzeugen
  GetMethod method = new GetMethod(href);
  method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
      new DefaultHttpMethodRetryHandler(3, false));

  try {
    // GET ausführen
    int statusCode = client.executeMethod(method);
    // Wurde die Seite gefunden, wird TRUE zurückgegeben
    return statusCode == HttpStatus.SC_OK;
  } finally {
    // Freigeben der Verbindung.
    method.releaseConnection();
  }
}
```

Doch befriedigend ist die Lösung noch nicht. Ist der Hilfetext umfangreich, wird es für den Benutzer unübersichtlich und er muss den für ihn interessanten Teil erst suchen. Abhilfe würde die Vorgehensweise schaffen, für jedes Control einen eigenen Kontext zu erstellen. Wir fanden es allerdings wenig praktikabel, für jeden Kontext auch eine eigene Seite im Wiki zur Verfügung zu stellen. Also suchten wir einen Weg, wie dies umgangen werden konnte. Hierbei mussten wir zwischen zwei Varianten wählen. Bei der ersten Möglichkeit wird im Kontext des Controls derselbe URL angegeben wie bei dem Top-Level-Composites, ergänzt durch einen Anker:

```
<context id="wichtiges_thema" title="Wichtiges Thema">
  <description>Dies ist ein Link zu Eclipse in Wikipedia</description>
  <topic href="http://www.eclipse.org#WichtigesThema" label="Wichtiges
                                                        Thema "/>
```

Auf diese Weise wird dem Benutzer im Browser sofort die richtige Stelle im Hilfetext angezeigt. Allerdings führt dies unter Umständen zu sehr langen Hilfeseiten.

Bei der zweiten Variante kann jedes Control einen Kontext mit eigenem URL bekommen. Jedoch überprüft die Klasse *BrowserHelpUI*, ob die Seite existiert. Dazu änderten wir die *displayContext*-Methode (Listing 1) in der Art ab, dass wir zuerst überprüfen, ob hinter dem angegebenen URL im Kontext überhaupt eine HTML-Seite vorhanden ist. Dazu wird per HTTPClient von Apache-Commons [9] eine Verbindung aufgebaut und ein *GET* auf die Hilfeseite aufgerufen. Anhand des Rückgabewerts der GET-Methode (HTTP 200 OK) kann festgestellt werden, ob der URL valide war. Ist die gewünschte Wikiseite nicht vorhanden, wird das Control gesucht, in das das Ausgangs-Control (Parent) eingebettet ist. Von diesem Control wird beim *HelpSystem* der Kontext erfragt und wiederum der Help-URL überprüft. Hat dies keinen Erfolg, wird es weiter rekursiv aufgerufen. Irgendwann landet man bei einem Control, das eine existierende Seite repräsentiert. Wird kein entsprechendes Control gefunden, wird vom Help-System der URL des *Context_Missing* geholt und aufgerufen.

Geschafft! Wir haben somit erreicht, dass die Hilfeinhalte nicht mehr in der Anwendung selber, sondern im Browser dargestellt werden. Wie erwähnt, gibt es ein paar kleinere Einschränkungen, aber diese können vom verwendeten Wiki aufgefangen werden (Suchfunktion, Indizierungen). Die einfache Integration des Hilfesystems in die Anwendung (Stichwort F1) und das Verwenden von Kontexten bleiben aber bestehen. Somit stellt dieser Ansatz für uns die zurzeit optimale Lösung dar.

Das F1-Problem

Noch nicht allgemein bekannt ist die Tatsache, dass die Hilfetaste F1 unter Windows fest mit der Kontexthilfe verdrahtet ist. Da wir SWT verwenden und SWT direkt auf OS-Ressourcen zugreift, gibt es leider keine Chance, dies auf elegante Art und Weise zu ändern. In der Eclipse-Implementierung wird dem Control, dem wir unseren Kontext zuweisen, ein *WorkbenchHelpListener* hinzugefügt. Das Control ruft dann bei Bedarf die Methode *helpRequested* auf. In den RCP Sourcen ist uns keine weitere Möglichkeit aufgefallen, dies anzupassen. Schlussendlich haben wir nach dem Studium der Klasse *org.eclipse.ui.internal.actions.DynamicHelpAction* und der Methode *appendAccelerator* aufgegeben. Der Kommentar besagt ebendies. Anders sieht es bei einem einfachen Aufruf der Hilfefunktion ohne Kontext aus, dies lässt sich über Key Binding anpassen.

4. Fazit

Wie man gesehen hat, war es mit relativ wenigen Handgriffen möglich, das Eclipse-Hilfesystem unseren Anforderungen anzupassen. Dies hat mehrere Gründe: Zum einen bringt die Standardfunktionalität schon einen vernünftigen Satz an Möglichkeiten mit, die mittels Konfiguration sehr weit angepasst werden können. Zum anderen können durch Austausch des UI-Teils der Hilfe per Extension Point relativ einfach Veränderungen am Look and Feel der Hilfe vorgenommen werden. Unsere Lösung mag einfach erscheinen, der Weg dahin allerdings war recht steinig. Die Dokumentation über die Anpassung des Hilfesystems im Web und in Eclipse ist verbesserungswürdig. Vielleicht kann dieser Artikel diesen Punkt etwas abmildern.

Holger Grosse-Plankermann, Diplom-Informatiker, ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme mbH tätig. Er beschäftigt sich seit vielen Jahren mit verteilten JEE-Systemen, Web-2.0-Technologien sowie Rich-Client-Architekturen und deren Umsetzung mit Eclipse RCP. Kontakt: h.grosse-plankermann@iks-gmbh.com

Jörg Meister arbeitet seit vielen Jahren als freiberuflicher Softwareentwickler und Berater. Seine Schwerpunkte sind Rich-Client-Architekturen und JEE-Systeme. Kontakt: jm@joergmeister.de

Links & Literatur

[1] Imrie, Alexandra; Lörke, Achim (Teil 2 des Tutorials „RCP-Usability“): „Hilfe naht“, in: Eclipse Magazin 3.10

[2] Eclipse Rich Client Platform: Designing, Coding, and Packaging Java Applications, ISBN: 978-0321603784

[3] Die Eclipse Rich Client Platform: Entwicklung von erweiterbaren Anwendungen mit RCP, ISBN: 978-3939084914

[4] Adding Help Support to a Rich Client Platform (RCP) Application: http://www.eclipse.org/articles/article.php?file=Article-AddingHelpToRCP/index.html

[5] Eclipse-Hilfe: http://www.ralfebert.de/rcpbuch/help/

[6] http://download.eclipse.org/eclipse/downloads/drops/R-3.6-201006080911/index.php#RCPRuntime

[7] http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/ua_help_setup_infocenter.htm

[8] Dynamic Context Help: http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/ua_help_context_dynamic.html

[9] http://hc.apache.org/httpclient-3.x

[10] Die Beispielprojekte: http://www.iks-gmbh.com/rcp