Plus CD!

▶ Highlights von EclipseCon und JAX 2010 » 8

eclipse MAGAZIN

4.10 S&S

eclipse MAGAZIN

www.eclipse-magazin.de

EXKLUSIV AUF CD:

Eclipse MP3 Manager

RCP-Applikation gebaut mit Maven Tycho

- Eclipse SDK 4.0
- Eclipse Pulsar
- Eclipse SOA Platform
- Appcelerator Titanium 1.2
- ceno 0.0.4

Sinnkrise oder Innovationstreiber:

Quo vadis Eclipse?

Sonderdruck

„WIR MÜSSTEN BEI DER ECLIPSE FOUNDATION DEN GÜRTEL ENGER SCHNALLEN." » 24

▶ **Pulsar & Eclipse Mobile** » 28

Entwicklung für iPhone, Android & Co.

Datenträger enthält Info- und Lehrprogramme gemäß § 14 JuSchG

▶ **RCP-Testing**

Qualitätssicherung durch automatisierte SWTBot-Tests » 85

▶ **Navigationsbäume mit LunaRCP**

Baumkunde für Einsteiger » 90

▶ **Eclipse Modeling**

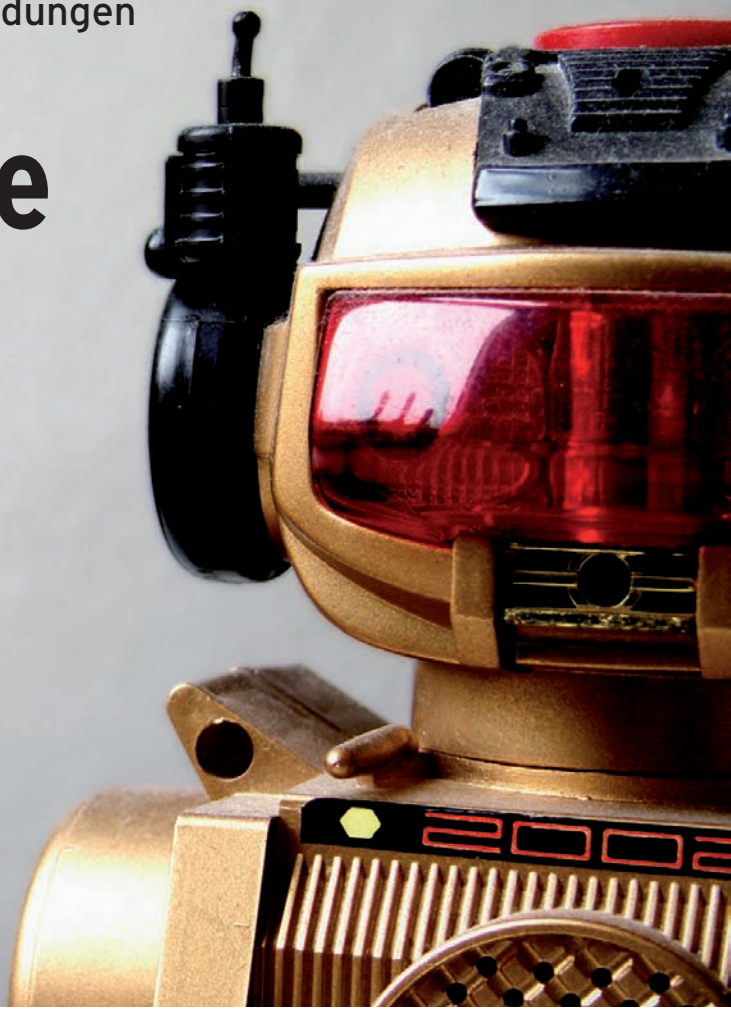
Model-2-Text-Transformationen » 42

▶ **Eclipse Target Platform**

Was bringt Eclipse Helios? » 68

Qualitätssicherung von Geschäftsanwendungen durch automatisierte SWTBot-Tests

Automatisierte SWTBot-Tests



>> DR. REIK OBERRATH

Automatisierte Tests, die mit dem Continuous Build ausgeführt werden, sind aus der Softwareentwicklung nicht mehr wegzudenken. Leider erschwert das Testframework der aktuellen Eclipse-Version 3.5 das Aufsetzen von automatisierten GUI-Tests sehr. Das Hauptproblem besteht darin, dass das Eclipse-Testframework der Version 3.5 unter JUnit4 nicht läuft [1]. Allerdings hat die Verwendung von JUnit3 zur Folge, dass SWTBot-Tests zunächst nicht ausgeführt werden können [2], [3]. Zusätzlich bringt die Verwendung von Log-in-Masken Schwierigkeiten mit sich. Dieser Artikel stellt als Beispiel für eine Geschäftsanwendung die Applikation „howareyou" zusammen mit ihrem Build-Prozess und SWTBot-Tests vor. Von diesen Voraussetzungen ausgehend wird gezeigt, wie die drei Technologien Eclipse RCP, SWTBot und JUnit3 „verheiratet" werden können, um SWTBot-Tests automatisiert mit dem Build-Prozess auszuführen.

Die hier beschriebene Beispielanwendung *howareyou* ist eine einfache RCP-Anwendung mit Loginmaske, Hauptmenü, einer View und einem Logging-Mechanismus (log4j). Beim Login erfolgt eine Userauthentifizierung. Die *howareyou*-View zeigt Daten des eingeloggten Benutzers, enthält eine Liste von Anwendern und einen Button. Der Inhalt der Liste hängt von der Rolle des eingeloggten Benutzers ab, wobei lediglich der Administrator alle Benutzer sehen kann. Nach Selektion eines Anwenders in der Liste und einem Klick auf den Button How are you? öffnet sich ein Userdialog mit einem Text, der von der vorherigen Selektion abhängig ist. Unter dem Menüpunkt File gibt es die Möglichkeit, sich mit einem anderen User einzuloggen (Relogin). Für den Administrator zeigt das Hauptmenü den zusätzlichen Menüpunkt Administration an (Abb. 1).

Der Kern der Anwendung ist autogeneriert und wurde mit dem Wizard „New Plugin-Project" und der Einstellung Would you like to create a new rich client application? sowie dem Template RCP application with a view erzeugt. Die Loginmaske ist als Dialog implementiert und wird im *ApplicationWorkbenchAdvisor* aufgerufen (Listing 1). Beim Relogin wird die Me-

thode *updateGuiWithUserPermission()* aufgerufen.

Da beim ersten Login noch keine Shell vorhanden ist, wird zur Initialisierung des Logindialogs immer eine neue Shell erzeugt (Listing 2). Im Fall eines Relogins wird die Shell der Anwendung unsichtbar gemacht.

Galileo ermöglicht mit mäßig großem Aufwand, einen Ant-Build-Prozess aufzusetzen, mit dessen Hilfe eine RCP-Anwendung automatisch (headless) gebaut werden kann [4], [5]. In einem Build-Verzeichnis müssen alle Eclipse-Projekte, die zur Anwendung gehören, nach Plug-ins und Features sortiert in gleichnamigen Unterverzeichnissen vorliegen. Das *howareyou*-Build-Skript erzeugt diese Verzeichnisstruktur und außerdem auch ein spezielles Target-Verzeichnis für den Build-Prozess (Tabelle 1).

Zu diesem Artikel gehören die Sourcen der Beispielanwendung *howareyou*, die unter [6] heruntergeladen werden können. Diese Sourcen beinhalten die Beispielanwendung in drei verschiedenen Entwicklungsstadien. Tabelle 2 gibt darüber einen Überblick.

Die *howareyou1*-Sourcen beinhalten vier Projekte. Das Projekt *com.iks_gmbh.howareyou.app* (später *app*-Plug-in genannt) beinhaltet den Quellcode der Anwendung und die Produktkonfiguration. Dazu gibt es ein Featureprojekt, daneben ein Build-Projekt für den automatischen Build-Prozess und ein Environment-Projekt, aus dessen *zip*-Dateien der Build-Prozess das Target-Verzeichnis erzeugt, falls es nicht schon vorhanden ist.

Die manuellen SWTBot-Tests

Die *howareyou2*-Sourcen beinhalten drei weitere Projekte: ein Testprojekt, mit dessen Hilfe alle Tests im Workspace zusammengefasst werden, außerdem ein Test-Base-Projekt, das Basisfunktionalität zur Implementierung von Tests zur Verfügung stellt, und schließlich ein Testfragmentprojekt, das die eigentlichen *howareyou*-Tests beinhaltet. Die Architektur des *howareyou*-Testframeworks beruht auf der Verwendung von Fragmenten [7]. Bei dieser Architektur bekommt jedes zu testende Plug-in ein Fragment, das alle Plug-in-spezifischen Test-Sourcen beinhaltet. Alle Testklassen des Fragments werden zu einer Test-Suite vereinigt und die Suites der einzelnen Testfragmente werden in einer globalen Suite vereinigt, die sich in einem speziellen Test-Plug-in befindet. Der Start der globalen Suite kann so alle vorhandenen Tests im Workspace ausführen. Voraussetzung dafür ist, dass die benötigten SWTBot-Sourcen im Target-Verzeichnis der Entwicklungsumgebung enthalten sind. Dabei handelt es sich um den Inhalt der *org.eclipse.swtbot.eclipse*-Zip-Datei, die sich im Environment-Projekt der *howareyou2*-Sourcen befindet.

Die Implementierung der SWTBot-Tests wurde mit Page-Objekten realisiert [8]. Außerdem leiten alle *howareyou*-Testklassen von der gemeinsamen Stammklasse *AbstractSWTBotTest* ab. Der Konstruktor dieser Stammklasse prüft, ob die Loginmaske zu bedienen ist und führt gegebenenfalls den Login mit einem Standarduser durch. Auf diese Weise braucht nicht jeder Einzeltest den Login zu implementieren und kann trotzdem einzeln ausgeführt werden.

Nach dem Login selektieren die *howareyou*-SWTBot-Tests einen Benutzer in der Liste und betätigen den HOWAREYOU-

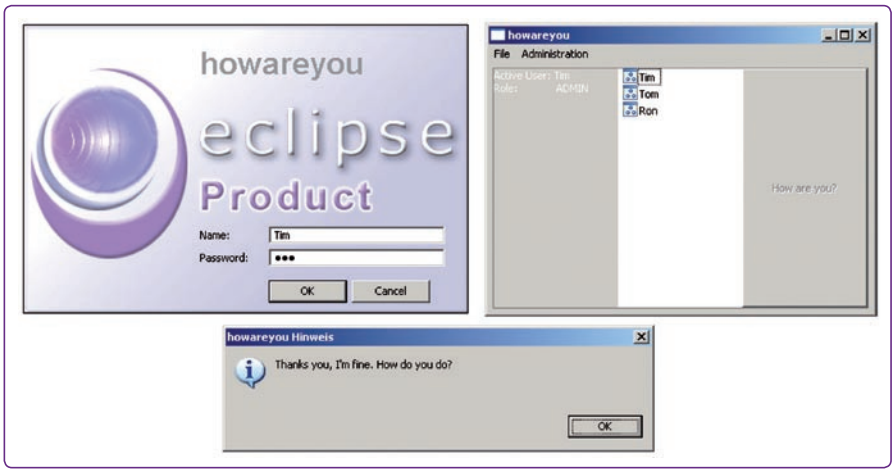


Abb. 1: Loginmaske, View und Userdialog der „howareyou"-Anwendung

buildHowareyouApplication	führt *initBuild* aus und startet dann den eigentlichen Build
initBuild	erzeugt das Build-Verzeichnis, führt *buildTargetIfNotPresent* und *initBuildOutputDir* aus
buildTargetIfNotPresent	prüft Target-Verzeichnis und führt ggf. *BuildTargetFromEnvironmentDir* aus
buildTargetFromEnvironmentDir	erzeugt das Target-Verzeichnis und entpackt die Zip-Files
initBuildOutputDir	kopiert alle Feature- und Plug-in-Projekte in die *feature*- und *plugin*-Unterverzeichnisse des Build-Verzeichnisses

Tabelle 1: Liste der Targets des „howareyou"-Build-Skripts

Listing 1

```
public class ApplicationWorkbenchAdvisor extends
                                    WorkbenchAdvisor {
...
 @Override
 public boolean openWindows() {
 LoginDialog.setApplicationWorkbenchAdvisor(this);
  LoginDialog.login();
  updateMainMenu();
  return super.openWindows();
 }

 public void updateGuiWithUserPermissions() {
  updateMainMenu();
  View.updateContent();
 }
}
```

Listing 2

```
public class LoginDialog extends Dialog {
...
 public static boolean login() {
  final Display d =
            PlatformUI.getWorkbench().getDisplay();
  final Shell shell = new Shell(d);
if (applicationShell != null)
                applicationShell.setVisible(false);
  (new LoginDialog(shell)).open();
  return true;
 }
}
```

Button. Anschließend wird der Inhalt des angezeigten Userdialogs auf Richtigkeit überprüft. Beim Ausführen der SWTBot-Tests ist es wichtig, im Launcher die Option Run in UI Thread zu deaktivieren und unter der Option Run a product das *howareyou*-Produkt auszuwählen. Tut man das mit den *howareyou1*-Sourcen, steht man vor einem Thread-Problem.

Bevor die SWTBot-Test-Sourcen ausgeführt werden, wird die zu testende RCP-Anwendung gestartet. Für den Start der Anwendung muss die Methode *Workbench.init()* ausgeführt werden, die wiederum die Methode *ApplicationWorkbenchAdvisor.openWindows()* aufruft. Diese Methode wartet aber wegen der Loginmaske auf eine Usereingabe (Listing 1). Da deshalb der *howareyou*-Start stehen bleibt, laufen die Tests nicht an. Die Loginmaske braucht also ihren eigenen Thread. Listing 3 zeigt, wie das geht.

Damit das Applikationsfenster erst nach dem Login erzeugt wird, wird die Methode *WorkbenchAdvisor.openWindows()* jetzt nicht mehr in der Methode *ApplicationWorkbenchAdvisor.openWindows()* aufgerufen (Listing 1), sondern nach erfolgreichem Login vom Logindialog über die Methode *ApplicationWorkbenchAdvisor.reallyOpenWindows()* (Listing 4).

Auf diese Weise startet das Eclipse-Framework die *howareyou*-Anwendungen ohne Applikationsfenster (aber mit Logindialog). Danach beginnt der SWTBot-Test und stellt bei seiner Initialisierung fest, dass eine Loginmaske bedient werden muss. Der SWTBot-Test füllt die Loginmaske aus und klickt den OK-Button. Das führt dazu, dass die *howareyou*-Anwendung ihr Applikationsfenster erzeugt. Parallel dazu beginnt die Testausführung, die ein bestimmtes Widget sucht. Sobald das Applikationsfenster zur Verfügung steht, findet der SWTBot-Test das Widget, und die Testausführung beginnt.

Das zweite Build-Skript enthält Ant Targets für die Testausführung.

Die automatischen SWTBot-Tests

Für die automatische Ausführung der Tests werden noch zusätzliche Quellcodes benötigt. Im Wesentlichen handelt es sich dabei um die Eclipse-Test-Framework-Sourcen (ETF) und spezielle SWTBot-Sourcen (*org.eclipse.swtbot.eclipse.test*). Das Environment-Projekt der *howareyou3*-Sourcen enthält alle benötigten Quellcodes für die automatische Testausführung.

Diese zusätzlichen Sourcen müssen im Target der Eclipse-IDE zur Verfügung gestellt werden. Deshalb muss das Ant Target *buildTargetFromEnvironment* im Build-Skript entsprechend erweitert werden. Außerdem müssen alle benötigten Test-Sourcen in die Anwendung eingebaut werden. Aus diesem Grund enthalten die *howareyou3*-Sourcen ein neues Testfeatureprojekt, das alle Test-Plug-ins beinhaltet. Da es wünschenswert ist, die Anwendung auch ohne Test-Sourcen bauen zu können, ist jetzt eine zweite Produktkonfiguration hilfreich, die neben dem *app*-Feature für die Plug-ins der eigentlichen Anwendung auch das Testfeature beinhaltet. Diese Produktkonfiguration heißt in den *howareyou3*-Sourcen *howareyou.test.product* und befindet sich, wie die erste Produktkonfiguration, im *app*-Plug-in.

Das bisherige Build-Skript bleibt bis auf die oben genannte Änderung im *buildTargetFromEnvironment* Target unverändert und kann benutzt werden, um die *howareyou*-Anwendung ohne Test-Sourcen als Release Candidate zu bauen. Im Verzeichnis *test* des Build-Projekts findet sich in den *howareyou3*-Sourcen ein zweites Build-Skript, das das erste Build-Skript benutzt, um die *howareyou*-Anwendungen als automatisch testbare Version zu bauen. Dieses zweite Build-Skript enthält Ant Targets für die Testausführung und hat außerdem eine eigene Properties-Datei, die die neue Produktkonfiguration *howareyou.test.product* referenziert. Jedes Build-Skript benutzt also eine eigene Produktkonfiguration.

Die Testausführung besteht aus drei Phasen. In der ersten Phase wird eine so genannte Testumgebung gebaut. Für den Bau der Testumgebung wird die *howareyou*-Zip-Datei (das Resultat des vorangegangenen Build-Prozesses) entpackt. Zusätzlich zu diesen Applikations-

	howareyou1	howareyou2	howareyou3	Beschreibung
app	X	X	X	Plug-in mit den Sourcen der Anwendung
app.test.fragment	-	X	X	Fragment für die Test-Cases des *app*-Plug-ins
app.feature	X	X	X	Feature mit allen Plug-ins, die für ein Release nötig sind
build	X	X	X	Sourcen des automatischen Build-Prozesses
environment	X	X	X	Speicherort aller nötigen *jar*- und *zip*-Dateien, um eine Runtime- oder Entwicklungsumgebung einzurichten
test	-	X	X	Plug-in für die globale Test-Suite
test.base	-	X	X	Plug-in für die Basisfunktionalität der Tests
test.feature	-	-	X	Feature mit allen Plug-ins, die zur Testausführung nötig sind
Beschreibung	Anwendung ohne Tests	Anwendung mit manuellen Tests	Anwendung mit automatischen Tests	

Tabelle 2: Übersicht über die „howareyou"-Sourcen und die darin enthaltenen Eclipse-Projekte

Sourcen werden verschiedene andere Sourcen (im wesentlichen SWTBot- und ETF-Sourcen) dazu kopiert. Die fertige Testumgebung findet sich im Build-Verzeichnis unter *testOutput\eclipse*. In der zweiten Phase werden die eigentlichen Tests ausgeführt und in der dritten die Ergebnisse zusammengefasst, analysiert und in einer XML-Datei (*results.xml*) bereitgestellt. Diese Ergebnisdatei findet sich nach der Testdurchführung im Build-Verzeichnis unter *testOutput/results*. Für die Phasen zwei und drei halten die Eclipse-Test-Framework-Sourcen in der Datei *library.xml* verschiedene Ant Targets bereit. Diese Targets wurden für *howareyou* angepasst und stehen in der gleichnamigen Datei im Build-Projekt unter *test* zur Verfügung. Wichtigste Änderung in dieser Datei ist ein spezielles Target zum Ausführen der SWTBot-Tests (Listing 5).

Die Umstellung auf JUnit3

Würde Eclipse 3.5 und das zugehörige Testframework JUnit4 richtig unterstützen, wäre eine automatische Testausführung jetzt möglich. Das ist leider nicht der Fall. Die Probleme mit JUnit4 äußern sich darin, dass in den Testklassen keine Testmethoden gefunden werden, oder dass eine *ClassCastException* ausgegeben wird, weil bei der Testausführung versucht wird, JUnit3- und JUnit4-Klassen aufeinander zu casten. Ein weiteres Problem ist, dass der OSGi-Container bei der Testausführung die Test-Fragment-Plug-ins nicht laden kann, weil nicht näher benannte Constraints mit dem Host existieren. Da JUnit4 für automatische Tests also nicht zur Verfügung steht, muss die Testausführung auf JUnit3 umgestellt werden. Diese Umstellung ist leider aufwändig. Zunächst müssen in den *howareyou*-Projekten alle JUnit4-Abhängigkeiten durch JUnit3 ersetzt werden. Das betrifft folgende Projekte:

- *com.iks_gmbh.howareyou.test*
- *com.iks_gmbh.howareyou.test.base*
- *com.iks_gmbh.howareyou.test.feature*
- *com.iks_gmbh.howareyou.app.test.fragment*

Diese Umstellung hat folgende Codeanpassungen zufolge:

- *AbstractSWTBotTest* von *junit.framework.TestCase* ableiten
- Importanweisungen für *org.junit.Assert.assertEquals* entfernen

> Abhängigkeiten zu den SWTBot-Sourcen sind unbedingt nötig.

- JUnit4-Annotierungen und ihre Importanweisungen entfernen

Weitere JUnit4-Abhängigkeiten, die die Testausführung unter JUnit3 verhindern, finden sich in den JDT-Sourcen und den SWTBot-Sourcen. Mögliche Abhängigkeiten der durch das Build-Skript automatisch gebauten Anwendung zu den Eclipse-Java-Development-Tools (JDT) müssen vollständig gelöscht werden. Die Abhängigkeiten zu den SWTBot-Sourcen sind aber unbedingt nötig. Deshalb muss der SWTBot-Quellcode ausgecheckt und genauso wie die *howareyou*-Sourcen wie eben beschrieben auf JUnit3 umgestellt werden. Wie Auschecken und Kompilieren funktionieren, ist unter [9] und [10] beschrieben. Anschließend wird der SWTBot-Sourcecode neu kompiliert und dann der *howareyou*-Anwendung wieder zur Verfügung gestellt.

Die *howareyou3*-Sourcen beinhalten auf diese Weise modifizierte SWTBot-Sourcen für den SWTBot-Build 433. Wer eine jüngere Version von SWTBot benötigt, muss diese JUnit-Modifizierung selbst vornehmen. Allerdings stehen für das SWTBot-Eclipse-Test-Zip-File bereits verschiedene Versionen für JUnit3 und JUnit4 zur Verfügung, sodass zukünftig nur noch das SWTBot-Eclipse-Zip-File für JUnit3 modifiziert werden muss.

Nach diesen Umstellungen laufen die SWTBot-Tests allerdings immer noch nicht automatisch. Auch die manuell gestarteten Tests funktionieren nach diesen Umstellungen nicht mehr. Der Fehler ist im manuellen wie im automatischen Fall derselbe: Die SWTBot-Sourcen können die Klasse *SWT* nicht finden, und es wird eine *ClassNotFoundException* geworfen. Dieses Problem kann nicht durch Zuweisen eines SWT-Plug-ins als Abhängigkeit gelöst werden. Allerdings kann es sehr einfach durch folgenden Eintrag in die Manifest-Datei des *test*.

Listing 3

```
public static boolean login() {
 final Display d =
          PlatformUI.getWorkbench().getDisplay();
 final Shell shell = new Shell(d);
 Runnable loginRunnable = new Runnable() {
  public void run() {
   (new LoginDialog(shell)).open();
if (applicationShell != null)
                applicationShell.setVisible(false);
  }
 };
 d.asyncExec(loginRunnable);
 shell.forceActive();
 return true;
}
```

Listing 4

```
public class ApplicationWorkbenchAdvisor extends
                                WorkbenchAdvisor {
...
 @Override
 public boolean openWindows() {
 LoginDialog.setApplicationWorkbenchAdvisor(this);
  LoginDialog.login();
  return true;
 }

 public void reallyOpenWindows() {
  super.openWindows();
  updateMainMenu();
 }
...
}
```

Listing 5

```
<target name="swtbot-test"
description="Launches SWTBot
                                tests." depends="init">
<echo> Running SWTBot tests... </echo>
 <antcall target="eclipse-test">
<param name="application"
value="org.eclipse.swtbot.eclipse.junit3.headless
                            .swtbottestapplication"/>
<param name="testApplication" value=
    "com.iks_gmbh.howareyou.app.application"/>
 </antcall>
</target>
```

base-Plug-ins gelöst werden: *Eclipse-RegisterBuddy: org.eclipse.swtbot.swt.finder*. Dieser Eintrag ermöglicht den SWTBot-Sourcen, auf die *test.base*-Sourcen zuzugreifen und damit die *SWT*-Klasse zu finden.

Mit dieser letzten Anpassung laufen die SWTBot-Tests unter JUnit3 sowohl manuell im Eclipse Launcher als auch automatisch im Build-Prozess. Bei der Konfiguration des Launchers sind folgende drei Punkte zu beachten:

- Im Tab '*test*' muss der JUnit-Runner für JUnit3 gewählt werden
- Im Tab '*test*' muss die Option RUN IN UI THREAD deaktiviert werden
- Im Tab '*main*' muss die Option RUN A PRODUCT gewählt und ein *howareyou*-Produkt ausgewählt werden

Automatische GUI-Tests im Entwickleralltag

Die in diesem Artikel dargestellten Probleme und ihre Lösungen waren Teil der Qualitätssicherung einer Geschäftsanwendung, die aus einem RCP-Client und einem EJB3-basierten Server besteht. Der Datenaustausch zwischen Client und Server erfolgt über RMI. Der Client kann so gebaut werden, dass er mithilfe von Mock-Remote-Objekten selbst den Server simuliert. Auf diesem so genannten Democlient werden im Continuous Build automatische SWTBot-Tests ausgeführt. Der Continuous Build startet mit jeder Änderung im Repository. Zusätzlich gibt es einen Nigthly Build, der die gleichen SWTBot-Tests auf dem richtigen Client (ein Client ohne Mock-Remote-Objekte) ausführt. Dieser Client kontaktiert einen Testserver, hinter dem eine leere Testdatenbank steht. Die SWTBot-Tests erzeugen, laden, bearbeiten und löschen Geschäftsobjekte und führen damit Systemtests automatisch durch, weil hinter dem getesteten Client ein voll funktionsfähiges System steht.

> Unter Helios sollte automatisiertes Testen deutlich einfacher sein.

Ausblick

Ab Eclipse 3.6 soll das Eclipse-Testframework mit JUnit4 funktionieren [1]. Unter Helios sollte das Aufsetzen von automatischen Tests für RCP-Anwendungen also deutlich einfacher sein. Wir werden sehen, ob das auch zutrifft.

Dr. Reik Oberrath ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme tätig. Er beschäftigt sich seit vielen Jahren mit der Entwicklung von individuellen Geschäftsanwendungen, speziell unter Swing und RCP. Besonderen Fokus legt er auf Automatisierung in der Qualitätssicherung. Kontakt: r.oberrath@iks-gmbh.com

>> Links & Literatur

[1] https://bugs.eclipse.org/bugs/show_bug.cgi?id=153429

[2] https://bugs.eclipse.org/bugs/show_bug.cgi?id=220812

[3] http://swtbot.org/bugzilla/show_bug.cgi?id=79

[4] http://www.vogella.de/articles/EclipsePDEBuild/article.html

[5] http://help.eclipse.org/help32/index.jsp?topic=/org.eclipse.pde.doc.user/guide/tasks/pde_product_build.htm

[6] Quellcodes der Beispielanwendung howareyou: Eclipse Magazin 4.10: http://it-republik.de/jaxenter/eclipse-magazin-ausgaben/

[7] Dilger, Martin: „RCP-Tests mit Fragmenten", in Eclipse Magazin 4.09, S. 80

[8] http://it-republik.de/jaxenter/artikel/2275

[9] http://wiki.eclipse.org/SWTBot/Contributing

[10] http://wiki.eclipse.org/SWTBot/Maintaining_SWTBot_Versions_For_Your_Team