Quellcode zum Download

SWTBot-Erweiterung für den Test von NatTable-Elementen

Automatisierte UI-Tests

In Zeiten von Continuous Delivery ist ein hohes Maß an Testautomatisierung unabdingbar. Für oberflächenzentrierte Anwendungen auf Basis der Eclipse Rich Client Platform sollte hierbei auch Funktionalität der Benutzeroberfläche automatisiert getestet werden. Für diesen Zweck gibt es bereits ausgereifte Frameworks oder Toolsets, die jedoch meist auf die Standardoberflächenelemente beschränkt sind. Dieser Artikel zeigt am Beispiel des Frameworks SWTBot, wie mit verhältnismäßig wenig Code und etwas Hintergrundwissen eine Erweiterung umgesetzt wurde, mit der Nebula-NatTable-Instanzen im Rahmen eigener Rich-Client-Anwendungen komfortabel getestet werden können.

von Jan-Paul Buchwald

SWTBot [1] ist ein Open Source Java-basiertes Tool für UI und funktionale Tests von SWT und Eclipse-basierten Anwendungen. Ziel ist eine möglichst einfache Entwicklung von automatisierten Tests durch Bereitstellung eines intuitiv bedienbaren API, das technische Details von SWT oder Eclipse weitgehend abstrahiert. SWTBot kann sowohl direkt aus der Eclipse Workbench, als auch über Ant oder Maven Tycho im Rahmen automatisierter Build- oder Deployment-Prozesse verwendet werden. Die Bibliothek bringt die Unterstützung der gängigen SWT Controls wie Formularelemente (Buttons, Combos, Checkboxen etc.), Menüs, Tables oder Tree-Elemente mit. Native Dialoge wie Dateiauswahl werden leider nicht unterstützt. Für die Anbindung eigener SWT Controls können verschiedene generische abstrakte Klassen erweitert werden. Dies wird in diesem Artikel am Beispiel eines NatTable-Tabellen-Controls gezeigt.

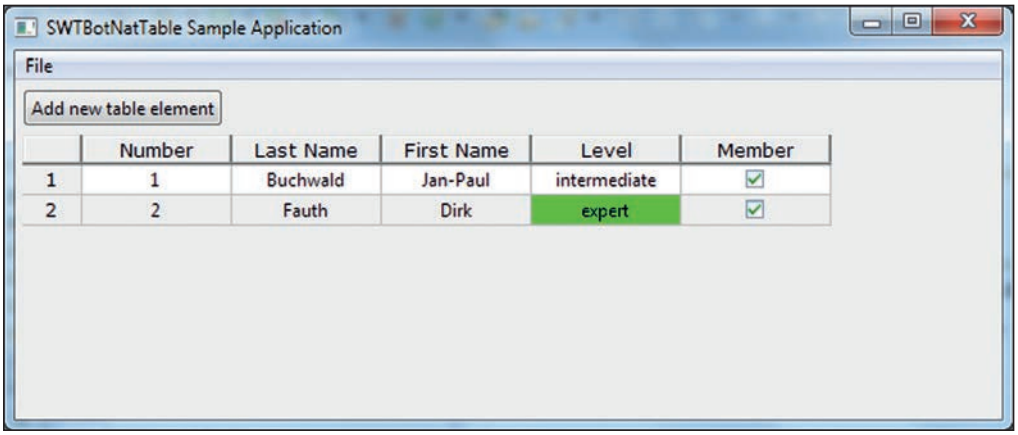

Abb. 1: Zu testende Beispiel-Eclipse-RCP-Anwendung

NatTable [2] ist ein Framework, um komplexe, hochperformante Tabellen, Grids und Trees zu erstellen. Es ist seit Anfang des Jahres als Stable Release innerhalb des Eclipse-Nebula-Projekts verfügbar [3]. NatTable bietet vor allem bezüglich Darstellung, Editiermöglichkeiten und Benutzeroptionen deutlich mehr Flexibilität und Optionen als das Standard SWT Table Control. Aus diesem Grund werden über NatTable Controls in Rich-Client-Anwendungen oft kritische Anwendungsfälle oder Benutzerinteraktionen realisiert, die auch in automatisierten Tests abgebildet sein sollten.

Im Folgenden wird gezeigt, wie ein automatisierter UI-Test für NatTable Controls mit dem *SWTBotNatTable*, einer SWTBot-Erweiterung mit Operationen und nützlichen Hilfsmethoden zur automatisierten Bedienung einer NatTable-Instanz realisiert wurde. Anhand eines einfachen Beispiels wird zunächst verdeutlicht, wie ein Testfall mit der Lösung aussieht. Im Anschluss werden die allgemeinen Erweiterungspunkte von SWTBot und die Implementierung und Benutzung des *SWTBotNatTable* beleuchtet. Abschließend wird auf die Ausführung von SWTBot-Tests innerhalb der Eclipse-IDE und Möglichkeiten zur Integration in einen automatisierten Build-Prozess eingegangen.

Beispielanwendung und Testfall

Als Beispiel und Anschauungsobjekt für den automatisierten Test der NatTable mit dem *SWTBotNatTable* wird eine einfache Eclipse-Rich-Client-Anwendung verwendet. Diese beinhaltet eine View mit einem Button zum Hinzufügen neuer Tabellenzeilen, sowie einem Tabellen-Grid mit Zeilen- und Spaltenheadern auf Basis der NatTable (**Abb. 1**). Die Zellen der Tabelle sind über typische Eingabemöglichkeiten wie die direkte Eingabe von Integerzahlen und Text per Tastatur sowie die Auswahl aus einer ComboBox und das Ankreuzen einer Checkbox editierbar. In der Spalte „Level“ ist eine bedingte Formatierung hinterlegt. Mit Klick auf den Button wird eine neue Zeile mit Dummyeinträgen der Tabelle hinzugefügt, nach Rechtsklick auf den Header der Zeile erscheint ein Kontextmenü, über das sich der Eintrag löschen lässt. Für die Zellen im Spaltenheader und Body-Bereich ist ein Tooltip gesetzt, der jeweils eine String-Ausgabe des aktuellen Zellenwerts anzeigt.

Listing 1 zeigt einen einfachen SWTBot-Testfall für die Beispielanwendung. Zum Verständnis des Codes ist zunächst wichtig, dass in SWTBot jedes Control oder Widget der zu testenden Anwendung über ein so genanntes Bot-Objekt angesprochen wird. Das ist eine Art Proxy für das eigentliche Widget und stellt Hilfsmethoden für Abfragen, Zugriff oder Interaktionen bereit. Der Testfall erweitert den *SWTBotEclipseTestCase* und kann somit über die Variable *bot* auf das übergreifende Testobjekt für die Eclipse-Workbench zugreifen. Dies ermöglicht den Zugriff auf die Basiselemente wie Perspektiven, Views, Dialoge etc. Im Test wird zuerst eine *SWTBotNatTable*-Instanz erzeugt und mit dem ersten und einzigen vorhandenen Widget vom Typ NatTable initialisiert. *SWTBotNatTable* ist die selbst implementierte Bot-Klasse für den Zugriff auf Instanzen des NatTable Controls. Nach einer Prüfung der anfänglichen Zeilen- und Spaltenanzahl wird über den Button ein neues Element hinzugefügt. Die anschließende Prüfung, ob sich die Zeilenanzahl um 1 erhöht hat, erfolgt über eine Warteoperation, da der Refresh der Tabelle asynchron im UI-Thread stattfindet. Im weiteren Verlauf des Tests werden die Werte des neuen Eintrags gesetzt bzw. geändert, und das Ergebnis sowie weitere Eigenschaften wie der Tooltip-Inhalt jeweils verifiziert. Zuletzt werden per Rechtsklick auf den Zeilenheader der ersten Zeile das erste Element aus der Tabelle entfernt und erneut die Zeilenzahl sowie die Werte der übrig gebliebenen Einträge geprüft.

Mit diesen (für Java-Verhältnisse) wenigen Zeilen Code ist ein lesbarer, aber dennoch nicht trivialer Testfall entstanden, der auch von Testentwicklern ohne tiefere Eclipse- oder SWT-Kenntnisse umgesetzt werden könnte. Durch die Operationen des *SWTBotNatTable* lassen sich typische Benutzerinteraktionen mit der Tabelle direkt aufrufen. Wie diese Operationen intern umgesetzt sind und was bei der Entwicklung von Bot-Klassen für SWT Controls zu beachten ist, wird im Folgenden näher beleuchtet.

Erweiterungsmöglichkeiten von SWTBot

Im Standardumfang von SWTBot ist bereits eine Vielzahl von Bot-Klassen für die gängigen SWT-Widgets enthalten, wie zum Beispiel *SWTBotButton*, *SWTBotLabel* oder *SWTBotText*. Diese Klassen leiten überwiegend von der abstrakten generischen Klasse *AbstractSWTBot* bzw. deren Spezialisierung *AbstractSWTBotCon-*

Allgemeine Herausforderungen von Eclipse-RCP-UI-Tests

Automatisierte User-Interface-Tests sind allgemein umstritten, vor allem was das Kosten-Nutzen-Verhältnis angeht. Typische Gegenargumente sind ein hoher Entwicklungs- und vor allem Pflegeaufwand, da Testcode meist auch nach nur kleinen Änderungen in Layout oder Verhalten der Benutzeroberfläche angepasst oder komplett neu erstellt werden muss. Ein Testframework mit Aufzeichnungsfunktion und daraus generiertem Testcode kann hier Zeit sparen, jedoch müssen typischerweise dynamische Testdaten und Prüfungen manuell ergänzt werden. In SWTBot gibt es ein experimentelles *Recorder and Test Generator*-Feature, das allerdings leider nur rudimentäre Hilfestellungen und ein grobes Gerüst für eigenen Testcode liefert. In der bisherigen Projektpraxis wurden SWTBot-Tests deshalb ausschließlich komplett manuell implementiert. Eine weitere Schwierigkeit sind Timing-Probleme und Race Conditions, die oft nur durch die extrem hohe Klickfrequenz der automatischen Ausführung verursacht werden. Dies kann teilweise durch bedingte Warteoperationen oder zusätzliche Operationen im Test, die z. B. ein bestimmtes Widget vor einem Klick fokussieren, gelöst werden. Durch die asynchrone Ausführung und Zeitverhalten von UI-Operationen gibt es aber generell bei SWTBot-Tests ein hohes Risiko, dass manche Fehler oder bestimmte Verhaltensweisen nicht deterministisch reproduzierbar sind. Man sollte sich deshalb vor allem beim Testen von komplexen Benutzerschnittstellen auf diverse Iterationen in der Entwicklung der Testfälle einstellen, bis diese zuverlässig und stabil ausgeführt werden können.

trol für Control-Elemente ab. Die abstrakten Klassen sind auch der erste Ansatzpunkt für eine Unterstützung weiterer, nicht im Standardumfang enthaltener Controls oder Widgets in eigenen Bot-Klassen. Die Klasse *AbstractSWTBot* lässt sich per Typparameter auf einen bestimmten SWT-Widget-Typ einschränken, nimmt Instanzen von diesem Widgettyp im Konstruktor entgegen und stellt eine Reihe nützlicher Hilfsmethoden zur Interaktion mit dem Widget bereit. Auf dieser Basis wurde die vorgestellte *SWTBotNatTable*-Klasse implementiert und es haben sich vor allem folgende Funktionalitäten als nützlich erwiesen:

- Ausführung von Logik im UI-Thread: jegliche Interaktion mit dem durch die Bot-Instanz angesprochenen SWT-Widgets muss im Kontext des UI-Threads stattfinden. Hierfür stellt die abstrakte Klasse diverse Methoden *syncExec* und *asyncExec* für die synchrone oder asynchrone Ausführung von Logik im UI-Thread bereit. Listing 2 zeigt ein Beispiel, wie dies auch mit Rückgabetypen funktioniert.
- Maus- oder Tastatursteuerung: Hilfsmethoden, die sich um das SWT Event Handling von Mausklicks auf das Widget oder die Eingabe von Tastatur-Shortcuts kümmern.
- Zugriff auf allgemeine Eigenschaften des Widgets wie globale Beschriftung, SWT Styles, Kontextmenüs oder Tooltip-Texte.

Listing 1

```
@Test
public void testSampleTable() throws Exception {
  // Load SWTBotNatTable instance
  SWTBotNatTable botNatTable = new SWTBotNatTable(
    bot.widget(widgetOfType(NatTable.class)));
  // Verify number of rows and columns
  assertEquals(2, botNatTable.getBodyRowCount());
  assertEquals(5, botNatTable.getBodyColumnCount());

  // Add new element, wait for number of rows to increase
  bot.button("Add new table element").click();
  bot.waitUntil(botNatTable.getBodyRowCountCondition(3));

  // Set values of new entry
  botNatTable.setBodyValue(2, 1, "Vogel");
  botNatTable.setBodyValue(2, 2, "Lars");

  // Verify values
  assertEquals("Lars", botNatTable.getBodyValue(2, 2));
  assertEquals("beginner", botNatTable.getBodyValue(2, 3));
  assertEquals(false, botNatTable.getBodyValue(2, 4));
  // no label set for level column
  assertFalse(botNatTable.hasBodyCellLabel(2, 3, View.EXPERT_LABEL));
  // verify tooltip of 3rd column in second row:
  // hover for 100ms more than the default delay of 0,5 sec
  botNatTable.hoverBodyCell(1, 2, 600);
  assertEquals("Dirk", botNatTable.getDefaultTooltipText(bot));

  // Select 3rd value (expert) in combo box
  botNatTable.selectBodyComboBoxValue(2, 3, 2);

  // assert new value
  assertEquals("expert", botNatTable.getBodyValue(2, 3));
  // now label is set for level column
  assertTrue(botNatTable.hasBodyCellLabel(2, 3, View.EXPERT_LABEL));

  // Click (and thus check) the checkbox in the 5th column
  botNatTable.clickBodyCell(2, 4);
  assertEquals(true, botNatTable.getBodyValue(2, 4));

  // Perform right click on first line row header
  // (necessary to show context menu)
  botNatTable.clickRowHeader(0, true);
  // Delete second entry via context menu
  SWTBotMenu menuItem = botNatTable.contextMenu("Delete entry");
  assertNotNull(menuItem);
  menuItem.click();

  // Verify number of rows has decreased
  bot.waitUntil(botNatTable.getBodyRowCountCondition(2));
}
```


WIR SCHREIBEN GESCHICHTE(N)
webmagazin
Weiterlesen auf
www.webmagazin.de
SHAPING YOUR DIGITAL EXPERIENCES

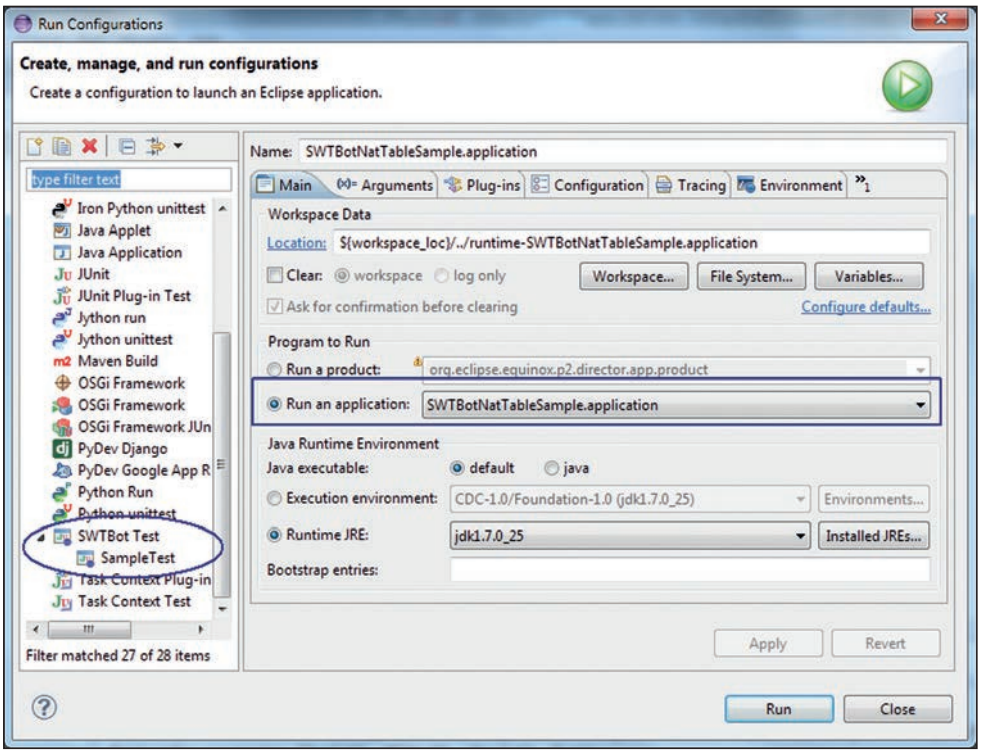

Abb. 2: Anlegen der Run Configuration für den SWTBot-Test in der Eclipse-IDE

Listing 2

```
/**
 * Returns the current row count of the body region of the table.
 */
public int getBodyRowCount() {
  return syncExec(new IntResult() {
      @Override
      public Integer run() {
        return getBodyLayer().getPreferredRowCount();
      }
  });
}
```

Testen von Tooltips

Der Beispieltestfall in diesem Artikel enthält die Prüfung der Anzeige eines Tooltip-Texts, der durch den standardmäßig in der NatTable enthaltenen *NatTableContentTooltip* erzeugt wird. Da das Verhalten und der Aufbau von Tooltips sehr stark implementierungsabhängig ist, wurde dies im *SWTBotNatTable* in zwei separate Schritte aufgespalten: Eine erste Methode *hoverBodyCell()* ermöglicht es, ein Hovern mit der Maus über eine bestimmte Tabellenzelle für einen bestimmten Zeitraum zu simulieren. Eine zweite Methode *getDefaultTooltipText()* sucht dann über verschiedene Kriterien den geöffneten Tooltip und liefert dessen Text zurück. Die Implementierung geht dabei von den Details des *org.eclipse.jface.window.DefaultToolTip* aus, in dem der Tooltip aus einer separaten Shell mit einem *CLabel*-Element besteht. Diese Shell kann leider nur auf Basis sehr spezieller und stark implementierungsabhängiger Details identifiziert werden. Bei Umsetzung individueller Tooltip-Klassen für die NatTable kann diese Struktur abweichen und die Überprüfung muss entsprechend der eigenen Implementierung angepasst werden.

Details zur Implementierung des SWTBotNatTable

Für das Testen von NatTable Controls muss darüber hinaus noch auf Widget-spezifische Funktionalität zugegriffen werden. Die Klasse *SWTBotNatTable* kapselt dabei vor allem folgende Aspekte:

- Zugriff auf einzelne Zellen der Tabelle unter Berücksichtigung der Layer-Struktur der NatTable. Das NatTable Control ist intern aus einer Komposition und Verschachtelung diverser Layer aufgebaut, die entweder die Struktur (Zeilen- oder Spaltenheader, Body) oder Funktionalität (z. B. Filterung, Selektion) der Tabelle realisieren. Dies wird intern im *SWTBotNatTable* berücksichtigt, indem sich die meisten Methoden auf einen bestimmten Struktur-Layer beziehen. Dieser wird für zusammengesetzte Tabellen mit einem übergreifenden *CompositeLayer* anhand des Region Labels ermittelt. Für einfache Tabellen wird davon ausgegangen, dass nur ein Body-Layer-Stack existiert. Die Methoden des *SWTBotNatTable* bieten dann die Abfrage von Werten einer Zelle (Objektwert, Labels), Ausführung von Interaktionen (Klick, Kontextmenü, Setzen von Werten inklusive Berücksichtigung spezieller Zelleneditoren wie *ComboBoxCellEditor*) sowie auf den kompletten Layer bezogene Abfragen wie Anzahl der Zeilen oder Spalten. Auf die Layer für bestimmte Funktionalität wird nur bei Bedarf zurückgegriffen, z. B. bei der Ermittlung der aktuellen Selektion.
- Ausführung von Commands: Die NatTable bringt intern eine Schnittstelle für die Ausführung von Commands auf den Layern (*ILayerCommand*) mit. Diese wird vom *SWTBotNatTable* zum einen genutzt, um mit vorhandenen Commands verschiedene Operationen auf der Tabelle anzustoßen (Setzen von Werten, Auswahl von Zeilen oder Zellen). Zum anderen wird auch eine allgemeine Methode zur asynchronen Ausführung von Commands bereitgestellt, die zum Testen von selbst implementierten Commands verwendet werden kann.
- Ermittlung der Koordinaten für Mausklicks: Das Rechteck jeder Tabellenzelle der NatTable mit Informationen zu Startkoordinaten und Breite sowie Höhe kann über das Layer-Objekt direkt abgefragt werden. Hier ist allerdings zu beachten, dass die Position immer relativ zum aktuellen Layer ist, deshalb müssen zur kompletten Positionsbestimmung noch die entsprechenden Dimensionen von eventuell vorhandenen Spalten- oder Zeilenheader-Layern addiert werden. Die im *SWTBot-*

NatTable implementierten Klickmethoden führen Klicks immer auf die Mitte einer Tabellenzelle aus.

Wie im Testfall bereits ersichtlich, werden asynchrone Operationen in SWTBot über den Aufruf der Methode *bot.waitUntil(condition)* angestoßen und das Ergebnis abgeprüft. Hierfür enthält die Klasse *SWTBotNatTable* einige beispielhafte Conditions als Erweiterung der *org.eclipse.swtbot.swt.finder.waits.DefaultCondition* in Form von privaten Klassen, die für das Warten auf eine Änderung der Zeilenanzahl oder bestimmter Werte in der Tabelle verwendet werden können.

Die in der *AbstractSWTBot*-Klasse vorhandene Methode zum Rechtsklick auf einen Koordinatenpunkt funktionierte leider nicht, deshalb wurde hier eine korrigierte Methode direkt implementiert.

Die vollständige Implementierung des *SWTBotNatTable* ist im Quelltext zu diesem Artikel vorhanden und liefert mit den Codekommentaren detailliertere Hinweise und Begründungen zu den einzelnen Methoden.

Ausführen des Tests in Eclipse

Zur Nutzung von SWTBot und der NatTable in der Eclipse-IDE müssen zunächst einige Features installiert werden:

- *SWTBot for Eclipse Testing*, *SWTBot for SWT Testing*, *SWTBot IDE Features* von der SWTBot-Updatesite [4] (für den Beispielcode zusätzlich noch *SWTBot Junit Headless Launchers for Eclipse*)
- *NatTable Core Feature* 1.0.1 und *NatTable GlazedLists Extension Feature* von [5]
- *GlazedLists* über die Orbit-Updatesite [6]

Sind diese Voraussetzungen geschaffen, lässt sich das zum Artikel gehörende Beispielcodearchiv in Eclipse 4.2 oder 4.3 als vorhandenes Projekt importieren und nutzen. Den Beispielcode finden Sie auf der Seite des Eclipse Magazins bei den Infos zu dieser Ausgabe. Der SWTBot-Testfall kann am einfachsten über den JUnit 4 Test Runner aus der Eclipse-IDE heraus ausgeführt werden. Hierzu muss eine SWTBot Test Run bzw. Debug Configuration angelegt werden, in der im Main-Tab unter „Program to Run“ die zu testende Application ausgewählt wird (**Abb. 2**). Beim Ausführen dieser Run Configuration öffnet sich ein separates Fenster mit der zu testenden Anwendung. In diesem werden die Testschritte ausgeführt und abschließend das Fenster wieder geschlossen. Die Ergebnisse des Tests sind anschließend in der JUnit View in Eclipse sichtbar.

Das Beispiel kann als Referenz für eigene SWTBot-Implementierungen dienen.

Integration in automatisierte Build-Prozesse

SWTBot lässt sich auch außerhalb der Eclipse-Workbench durch Werkzeuge wie Ant oder Maven von der Kommandozeile aus benutzen. Hierfür gibt es ein separates Feature *SWTBot JUnit Headless launchers for Eclipse*, wobei die Bezeichnung „Headless“ ein wenig missverständlich ist, da immer noch ein Display benötigt wird und auch aus der Kommandozeile heraus eine Eclipse-Workbench zur Durchführung der Tests geöffnet wird. Für Continuous Integration Server wie Hudson oder Jenkins gibt es aber Möglichkeiten, ein Display zu simulieren [7] und die Tests tatsächlich auf einem Server ohne grafische Anzeige auszuführen.

Die Beschreibung in diesem Artikel soll sich auf einen Kommandozeilen-Build mit vorhandenem Display im Rahmen von Maven Tycho [8] beschränken. Für die Ausführung der UI-Tests muss das SWTBot-Headless-Feature in der Target Platform vorhanden sein und die Plug-ins für Maven Surefire und Tycho Surefire müssen eingebunden werden. Für Letzteres müssen noch zwei spezielle Parameter und die ID der zu testenden Applikation gesetzt werden (siehe [9] und Listing 3). Damit werden die SWTBot-Tests im Rahmen des Maven Builds wie Unit Tests ausgeführt, entweder explizit über das Maven Target *integration-test*, oder im Rahmen des kompletten Builds über das Target *install*. Im Beispielcode finden sich die kompletten Maven-Definitionen für eine Eclipse Juno Target Platform. Die wesentlichen Maven-Definitionen wurden in ein separates Projekt *SWTBotNatTableSampleBuild* ausgelagert.

Fazit und Ausblick

Mit dem hier beschriebenen Ansatz lassen sich die wichtigsten Funktionen der Eclipse Nebula NatTable in einem automatisierten UI-Test abdecken. Gleichzeitig kann das Beispiel als Referenz oder Vorlage für eigene SWTBot-Implementierungen weiterer Controls dienen. Der Beispielcode ist verwendungsfähig mit aktuellen Versionen von Eclipse und NatTable, allerdings bei Weitem nicht vollständig und an diversen Stellen ausbaufähig, da nicht alle möglichen Funktionen oder Operationen enthalten sind. Eine Übernahme des *SWTBotNatTable* als Testfragment in das NatTable-Projekt selbst ist angedacht, um den Sourcecode der Eclipse-Community zur Verfügung zu stellen und die Testmöglichkeiten auch für Regression-Tests der NatTable selbst zu nutzen.

Jan-Paul Buchwald ist Senior Consultant im Java-Team der BeOne Stuttgart GmbH und verfügt über langjährige Erfahrung aus vielen Rollen und Lebenszyklusphasen der Softwareentwicklung mit Java, sowohl im Bereich portal- und webbasierte Lösungen als auch Rich-Client-Anwendungen.

Listing 3

```
<plugin>
  <groupId>org.eclipse.tycho</groupId>
  <artifactId>tycho-surefire-plugin</artifactId>
  <version>${tycho-version}</version>
  <configuration>
    <useUIHarness>true</useUIHarness>
    <useUIThread>false</useUIThread>
    <application>SWTBotNatTableSample.application</application>
  </configuration>
</plugin>
```

Struktur des Beispielcodes

Der referenzierte Beispielcode besteht aus drei Eclipse-Projekten: *SWTBotNatTableSample* enthält die Testanwendung, die im Wesentlichen durch die Klasse *com.beone.sample.View* implementiert ist. Das Projekt *SWTBotNatTableSampleTest* enthält die *SWTBotNatTable*-Klasse und die Testklasse *SampleTest* mit dem Beispieltestfall. *SWTBotNatTableSampleBuild* besteht nur aus einer Maven *pom.xml* zur Demonstration, wie UI-Tests über Maven angestoßen werden können.

Links & Literatur

[1] http://eclipse.org/swtbot/

[2] http://eclipse.org/nattable/

[3] Fauth, Dirk: „Nebula NatTable 1.0.0“, in Eclipse Magazin 5.2013, S. 78–83

[4] http://download.eclipse.org/technology/swtbot/releases/latest/

[5] http://download.eclipse.org/nattable/releases/1.0.1/repository/

[6] http://download.eclipse.org/tools/orbit/downloads/drops/R20130517111416/repository/

[7] http://wiki.eclipse.org/SWTBot/CI_Server

[8] http://eclipse.org/tycho/

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