universal.adapter

Lückenlos zwischen digitaler und realer Welt

- Home
- Impressum
- Über universal.adapter

Capture-Replay Regression Testing in Eclipse RCP


Dieser Artikel wurde von **Christoph Schmidt** geschrieben. Christoph Schmidt ist Softwareentwickler bei ubigrate. Er entwickelt dort unter anderem Werkzeuge auf Basis von Eclipse RCP und GEF.

19 Okt, 2009 Java, Programmierung

When it comes to testing the GUI of Java applications, you're typically faced with a tremendous amount of readily available testing tools for AWT and Swing, in all states of development, and all kinds of packages, be it standalone tools or some kind of plug-in to your favourite IDE. Now there's more than just AWT/Swing. Once you start developing your first Eclipse plug-in, you're bound to come across **SWT**, the GUI framework of Eclipse itself. Looking for tools capable of testing SWT-based applications is quite some task, the primary source for hints being some secluded mailing lists' archives. The task gets even tougher when you're setting more constraints: support for the Graphical Editing Framework (GEF) and robust Capture-Replay functionality, for example.

The result of your research will probably yield only a handful of tools which support SWT natively, and only one or two of them will explicitly satisfy your additional constraints. Here's the result of my own research, with AWT/Swing-only tools omitted:

tool	license	SUT-GUI-Toolkit support			
		SWT	AWT	Web	other
Abbot 1.0.2 [abb]	OSS CPL [lic06b]	☒[1]	☒	☐	
Bredex GUIdancer 3.1 [bre]	commercial (> 3900 EUR)	☒[2]	☒	☒	
Froglogic Squish for Java 3.2.3 [squ]	commercial (> 2400 EUR)	☒	☒	☐	
Instantiations WindowTester 3.9.1 [win]	commerical (> 629 USD)	☒	☒	☐	
Jemmy 2.3 [jem]	OSS CDDL1 [lic06a]	☒	☒	☐	Scenegraph, Java FX
QFS QF-Test 3.1.1 [qfs]	commercial (> 1595 EUR) and academic licenses	☒[2]	☒	☒	
SWTBot 2.0.0.371-dev-e34 [swtb]	OSS EPL [lic09]	☒	☐	☐	
TPTP 4.1 Automated GUI Recorder [tpt]	OSS EPL [lic09]	(☒)	☐	☐	Testframework for Eclipse-Internals

(July/August 2009; 1: „Undersupported“ [Roc06]. 2: With explicit GEF support.)

The situation among the GUI testing tools with SWT support is obvious: the majority of them is commercial, proprietary software, and those with an open source license are either not suitable for production yet or have shortcomings on the additional constraints part. SWTBot, for example, the most promising open source tool for SWT testing, lacks Capture-Replay support.

What remains is the commercial solutions. For evaluation purposes, **QFS' QF-Test** is chosen and presented in this article. This Capture-Replay enabled testing tool comes in three flavours (SWT, Swing, Web) and features a very detailed manual, an active mailing list and quick e-mail support.

Requirements

The evaluation is done to see if the chosen tool can meet the requirements set for the development of a GEF-based editor plug-in to Eclipse. The central requirement is functionality: the tool should actually be contributing to the quality of the developed product and mustn't be demanding much time to be tamed in the first place. Based on that, the tool primarily has to enable the developer to build up test suites with low effort and a linear workflow. Plus, it has to support GEF in a direct or at least indirect way.

The main problem met in any GUI test is recognizing the elements and controls of the graphical interface again in subsequent runs. Even under identical algorithmic circumstances, there is no guarantee that the structure of the object tree representing the user interface is going to be identical to the structures seen in subsequent runs. Thus, there must be a mechanism for user interface element recognition which on the one hand can abstract the objects found (e.g. truncate the standard identifiers to get rid of the ever-changing hashes) and on the other hand can pin down the properties which would allow for accurate discrimination between two objects of the same kind (e.g. two buttons of the same class, but with different captions). This mechanism should work in an efficient way and has to provide an easy way of correcting flawed behaviour.

Most of the tools that I looked at, including QF-Test, support the *setData*-Interface of SWT (see o*rg.eclipse.swt.widgets.Widget#setData(java.lang.String, java.lang.Object)* in [swta]), which can

support recognizing the GUI elements by means of evaluated key-value pairs. For example, QF-Test recommends the *name* property to be set in order to be able to use it as an additional hint in its component finding mechanism.

splitCanvas.setData("name", "MDMLEdit.Mainview.Sashform");

Evaluating QF-Test

One of the tools evaluated for said GEF-based editor is QFS' *QF-Test*. QFS provides any interested developer with a fully featured evaluation license, good for 30 days of Capture-Replay fun. The tool is released under a proprietary, commercial license [QFS07]. Academic licenses are offered besides the free, fully featured 30-days evaluation license, and the limited demo version. The developer licenses start at 1595 EUR (excl. VAT), with prices scaling with the number of supported APIs (SWT, Swing, Web) and the number of requested licenses. Finally, there are runtime licenses, starting at 1036,75 EUR (excl. VAT), which can only be used to run existing test suites, but not for editing or creating them. (Prices as of August 10th, 2009.)

The version evaluated for this article is *QF-Test 3.1.0-p1 (build 3363)*. The installation of the binary distribution is done by a setup program which configures the JDK installation and the available memory to be used by QF-Test. The chosen Java installation is instrumented with a special jar file by means of the *accessibility.properties* property of the JDK. This instrumentation is what enables QF-Test to log all user-induced events taking place within the graphical interface during the recording of the test suite.

The supplied manual [Sch09a] is very detailed and, containing 667 (German version) and 630 pages (English version) respectively, quite extensive in comparison. The provided "learning by doing" tutorial [KK09] covers the creation and execution of test suites. Besides tutorial and manual, QFS provides quite a fast mail support and quite an active mailing list, which is also covered by the support team of the software producer.

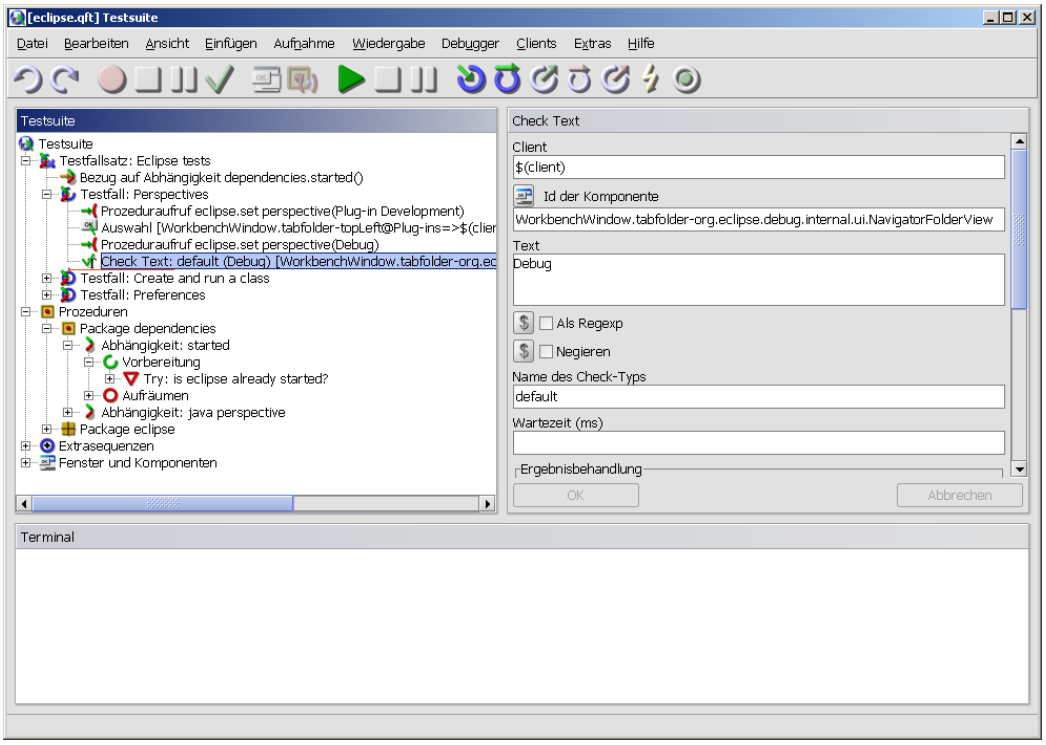
QF-Test is using Jython and Groovy as scripting languages. With those, user defined scripts may be integrated into the test suites by means of a special interface which the tool provides.

Creating a test suite always starts by linking up with the System Under Test (SUT). This can be done via the Quickstart Assistant which instruments the test subject with a modified SWT version since SWT isn't part of the JDK but part of the Eclipse distribution instead. Hence, the instrumentation during QF-Test's setup is not sufficient to capture events in SWT-based GUIs.

The basic workflow of creating a test suite consists of the recording of user-induced event sequences (ie. clicks and keystrokes) directly in the SUT, and enriching these event sequences with so called *Checks*. These Checks impose validations on elements of the captured interface interactions, similar to *assert()* of JUnit, and are used to test the behaviour of the SUT's interface during the replay of the event sequences. This is the foundation of the regression test the developer is heading for.

The big question looked upon during evaluation was – besides testing the GEF support – whether non-deployed Eclipse-plug-ins could be tested. That is, could plug-ins be tested which run within an Eclipse started by another Eclipse? It would be optimal if the tool would recognize the child instance and use it transparently for the testing process. It would be suboptimal, effort-wise, if the plug-in had to be deployed prior to every test run.

The tutorial starts off with an example suite, *Options.qft*, which is used to introduce the numerous features of QF-Test, concentrating on test execution and result inspection.



The user interface of the testing tool, as seen above, centers around a tree structure representing the structure of the suite (e.g. the test cases). This tree consists of a couple nodes of different types, which form the single steps of the test suite. On the right-hand side, a dynamic properties sheet allows for setting up the currently selected node. The bottom of the application window is taken up by a console.

The nodes contained in the example suite are as follows:

- Setup “Preparation”: Starts the SUT and waits for the connection to the JVM and for the main component of the SUT to show (i.e. the mainscreen).
- Test case “Clickstream”: Contains a sequence of mouse clicks and keystrokes. Test cases usually consist of several of these sequences.
- Sequences “Table” and “Tab”: Sequences are containers for several single actions or events (keystroke, text entered, mouse click, focus, window events, etc.). The first sequence contains two clicks, relative to a target component. A very nice feature is the context-sensitive help system, which links the current situation or view of the tool to a HTML version of the manual.
- Test case “Text Check”: Introduces the *Check* concept. In a Check, SUT-GUI elements are tested in the true spirit of regression tests, for example whether a certain label shows the correct text caption. The tutorial introduces the *Protocol* feature of QF-Test by willingly checking for a wrong value. This Protocol can be accessed after each test run and lists exactly what error was hit in which component, which value didn’t match, the stack trace of the erroneous situation, a screenshot of the visible screen content and a screenshot of the SUT window at the time of the error. This is what makes the Protocol a prime helper for debugging as well.
- Test case “Selected Test”: This test case extends the Check concept to non-textual components like radio buttons and tests whether the correct button is selected in the course of the test run.
- Tear down “Clean up”: Intended to shut down the SUT gracefully. At first, the tool waits for a certain amount of time for the SUT to close down normally. In case this doesn’t work out, a *try-catch* structure induces the hard termination of the SUT. This try-catch structure is used to introduce nodes which can steer the control flow of the test run. There’s a total of eleven of these structures: loops, while, break, if, else, elseif, try, catch, finally, throw and rethrow, complemented by the possibility to

execute your own scripts.

Setup and tear down embrace every single test case. During execution of the entire suite, a Protocol is created which highlights any errors and failed attempts, and can be used to generate an HTML report, based on XML and XSLT.

Creating test suites

The second half of the tutorial enables the user to create its own test suite, based on the Swing demo "FileChooserDemo". It is explained in detail how the SUT-start is being scripted and how some first, simple tests can be recorded and organised. Starting from here, tests for causal dependencies (aka. "Business Logic") are introduced, which do not solely validate the correct feedback behaviour of UI-primitives. Supernode "Extra sequences" is used as playground, under which recently recorded sequences can be organised.

Instead of the Quickstart Assistant, the start node "Start Java SUT client" is filled manually this time – all along to the call to the JVM. The Quickstart Assistant itself supports five types of SUT contexts:

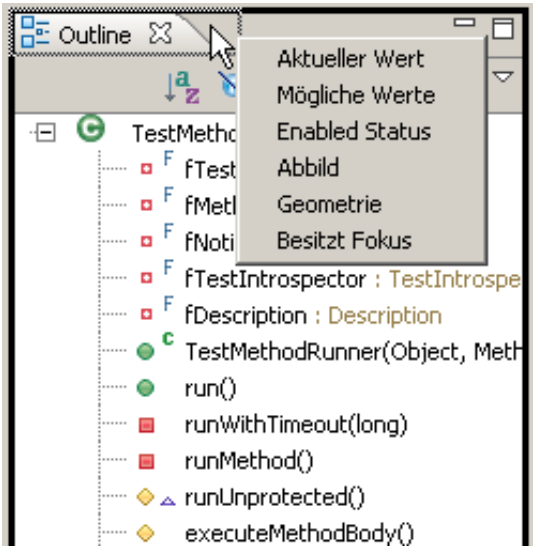
- Starting script/executable (bat/exe)
- Java Webstart/applet in a web browser
- Jars
- Class files
- Websites in a web browser

In order to record simple event sequences, the SUT is started through the testing tool and the recording mode is activated by a button in QF-Test. Afterwards, the tester can move freely inside the SUT. When done, the recording is stopped in QF-Test, which results in the captured sequence being dropped to the "Extra sequences" supernode. This sequence is already sufficient to be run as a test of the GUI primitives . It can also be used as fully automated live program demo. These recorded sequences are then used to assemble the actual suite. Multiple test cases can be grouped in a so called *test case set*, consisting of several event sequences. The test cases in turn consist of several event sequences. Then there are Procedures, which contain reusable sequences which can be accessed by means of special calling nodes. Another category, besides the supernode "Extra sequences" described above, is the "Windows and components" supernode which lists all seen GUI components with their respective properties.

For cleaning up the SUT in-between tests, prepare and clean-up nodes can be created and reused as dependencies between test cases. In them the SUT should be (re-)started or cleaned up to provide for an intact test bed. After starting the SUT, the tool waits for the client, which can then be referenced by a variable. Next, a wait-on-client or wait-on-component node is inserted to hold the test until the connection to the SUT is established and the GUI is displayed.

At this point, expressively named test cases can be filled with sequences from "Extra sequences" and inserted after the "prepare" references. An alternative to a full-blown clean-up node would be a simple sequence to the close-button of the SUT window.

Inserting Checks



Testing the client behaviour by means of Checks can be accomplished similarly to recording the sequences in the first place: while in record mode, a Check-Recording mode can be activated in QF-Test via F12 or a dedicated button. This enables a special context menu in the SUT, where the user can choose the kind of requested Check on the selected component. Some of the available Checks (depending on the component) are:

- Text (e.g. Caption equals …)
- Enabled status
- Image
- Elements (structure)
- Geometry, …

The test suite created through these steps is finally executed and indeed succeeds. The tutorial concludes this part by showing the error case again: the expected test values are modified on purpose and the resulting errors are explained in the Protocol.

The last part of the tutorial centers around a presentation of tests for the business logic of a SUT: in a sequence, a radio button is activated which enables a text field. At this point, a check on the text field is inserted which in turn tests the text field's "Enabled"-status. In the next step, the recorded click on another radio button disables the text field again. The consequences of these clicks are fixed through the recorded Checks and can then be regression-tested quite easily.

It shows that the work flow for creating test suites with QF-Test is downright linear, which is a highly valuable property for efficient test creation. After starting the recording, actions can simply be recorded live on the SUT, and tested by Checks in the same step. This enables the developer to work without interruptions or much backtracking, as is the case with purely scripting-based solutions like SWTBot.

Down to business: testing the prototype

Following up on the successful insight into the tool provided by the tutorial, the next item on the To-Do list is: will the tool work in my environment, with my project? How does the GEF support look like? Based on these questions, the evaluation version of QF-Test is applied to a prototype of the protocol model editor in question (*MDMLEdit* [Sch09b]).

First, the naive approach is tested: A sequence is recorded, using the "outer" Eclipse, where the Editor plug-in is being developed, as SUT and recording the process of starting an inner Eclipse, with the plug-in

deployed in a virtual way – it works! The inner Eclipse shows up as a child process of the actual SUT and can henceforth be used like a stand-alone SUT. Based on this environment, it is very easy to create and destroy an appropriate, modular testing context and put these into reusable Procedures. The context consists of a project within the inner Eclipse, which contains a file with the extension "*.mdml*", for which the plug-in registers.

In the next step, I test whether the tool can still recognize all the GUI elements, even within the inner context, and what happens when the GEF area is accessed. At first, the GEF-based main area of the editor presents itself as a single big, uniform, impenetrable control (a *FigureCanvas*). In order to enable GEF support, a Script node containing two simple lines of code taken from the mailing list archive of the tool needs to be inserted in the SUT-Setup.

import gef
reload(gef)

This script imports and enables the built-in GEF support of QF-Test. Now, the single *FigureCanvas* resolves into individual elements of the editor main view, like the palette and the parts of the displayed PDU.

There are, however, problems showing. The tool seems to have problems to recognize the palette of the editor in subsequent runs of the recorded sequence. It looks like the standard component resolver truncates too much of the corresponding object's ID. Part of this might be attributed to SWT: The editor main view is split by means of a *SashForm*. This is a container component which provides several *Sashes* for substructures. Now the palette and the editing area combine to the editor's main view in such a *SashForm*. Unfortunately, the *Sash* objects are simply numbered consecutively. This naming scheme is probably recognized as part of a standard Java object hash and truncated by the resolver. That way, palette and editing area effectively become superposed. At this point, however, there are already a couple obvious ways visible on how to resolve this in a reasonably effective way: for one, the Resolver can be refined by means of introducing customized *Item Resolvers* (compare [Sch09a], pages 546-580) into QF-Test, and for two, the naming of the single *Sashes* could be improved by either modifying their *toString()* method or setting the "*name*" property in their *setData*-interface. Using the *setData*-interface solved a problem with the resolver in another spot in no time. Changing *toString()* or using the *setData*-Interface would, however, require a customized *SashForm* class.

In summary, tests can easily be assembled after reaching the editor area and even in a quick-and-dirty evaluation test case, many Checks could be brought to good use already and worked as expected. There are some flaws in the component resolver, but there are at least two reasonable ways of fixing this, while the generally good GEF support makes up for this anyway. The work flow is extremely linear and the instrumentation approach used to log all events in the SUT during recording is fast and robust. Connecting with other testing tools like *XMLUnit* can likely be done by means of the scripting interface provided by QF-Test through the Script nodes. QF-Test will be subject to further testing here at ***ubigrate***. The next step is evaluating the possibilities of QF-Test to be used in *Model Driven Testing*, but that will have to wait for another article here on *universal.adapter*.

PS: You might want to check out another QF-Test walk-through in the article by Markus Stäuble [Stä09].

Sources and links

- Graphical Editing Framework (GEF) Homepage
- [abb] Abbot Homepage
  [bre] Bredex GUIdancer Homepage
- [jem] Jemmy Homepage
- [KK09] Karlheinz Kellerer, Michael Lehto Magnus Lindström, Martin Moser: QF-Test – Das

Tutorial, Version 3.1.1, Quality First Software GmbH (2009)
- [lic06a] Common Development and Distribution License, Version 1.0 (2006)
- [lic06b] Common Public License, Version 1.0 (2006)
- [lic09] Eclipse Public License – v 1.0 (2009)
- [qfs] Quality First Software QF-Test Homepage
- [QFS07] QFS: Lizenzvertrag Quality First Software GmbH, version 1.6 (2007)
- [Roc06] Roche, Thomas L: Re: Abbot Vs TPTP ?, Newsgroup (2006)
- [Sch09a] Schmid, Gregor: QF-Test – Das Handbuch, Version 3.1.1, Quality First Software GmbH (2009)
- [Sch09b] Schmidt, Christoph: Vorstudie für einen Protokollmodelleditor, Großer Beleg, Technische Universität Dresden (2009)
- [squ] Froglogic Squish for Java Homepage
- [Stä09] Stäuble, Markus: Automatisierte Kontroll-Untersuchung für RCP – QFTest/swt: Kommerzielles Werkzeug für automatisierte SWT-Tests. Eclipse Magazin (2009), Bd. 1: S. 43–45
- [swta] SWT API-Reference as part of the Eclipse Platform API Specification (*org.eclipse.swt.**)
- [swtb] SWTBot Homepage
- [tpt] Eclipse Test & Performance Tools Platform Project Homepage
- [win] Instantiations WindowTester SWT Homepage

Schlagworte: Eclipse • GEF • GUI • Java • QF-Test • QFS • RCP • Regression • SUT • Swing • SWT • SWTBot • Test

Hinterlasse einen Kommentar

Name:

eMail:

Website:

Kommentar:

Sag es

Speichern bei

Letzte Artikel

- Was ist eigentlich Business Activity Monitoring?
- Parameterized Builds in Jenkins – choosing subversion folders
- Building a custom UI Object with GT Designer 2 for Mitsubishi GOT 1000
- Capture-Replay Regression Testing in Eclipse RCP
- ServiceBrowser: browsing your rails controllers

Schlagworte

AOP ASM AT Automatisierung BCEL Bluetooth Bytecode Bytecode-Engineering C-MUX CEP Demo DPWS Eclipse Energy Monitoring Escape Flex Gastbeitrag GPS HMI Instrumentierung Java Javassist JAXB JUG Saxony Logistik Mindstorms Performance Preis Produktion RCP RFID RS232 RS485 SAP SOA Sun SPOT Telnet ubigrate usability USB Web Service wireless XJC XML Schema Zukunftsforum

Kategorien

- Allgemein (21)
- Anwendungen (10)
- Forschung (1)
- GUI (1)
- JUG Saxony (5)
- Markt (1)
- Programmierung (15)
  - C (1)
  - Flex (4)
  - Java (10)
- Technik (11)
  - Automatisierung (1)
  - Protokolle (4)
- ubigrate (8)

Archiv

- Januar 2012 (1)
- Mai 2011 (1)
- Mai 2010 (1)
- Oktober 2009 (1)
- Juni 2009 (1)
- Mai 2009 (2)
- April 2009 (2)
- März 2009 (2)

- Februar 2009 (1)
- Januar 2009 (2)
- Dezember 2008 (1)
- November 2008 (3)
- Oktober 2008 (5)
- September 2008 (2)
- August 2008 (2)
- Juli 2008 (2)
- Juni 2008 (4)
- Mai 2008 (7)
- April 2008 (5)

Letzte Kommentare

- Marcus bei Was ist eigentlich Business Activity Monitoring?
- RalfLippold bei Was ist eigentlich Business Activity Monitoring?
- Md Sirajuddin bei Creating Flex UIs in Java – A Short Tutorial
- Matt Doar bei Parameterized Builds in Jenkins – choosing subversion folders
- RalfLippold bei Neues vom RFID-Markt: Welche Krise?

Sonstiges

- Anmelden
- Artikel-Feed (RSS)
- Kommentare als RSS
- WordPress.org

LetzteBeiträge

- Was ist eigentlich Business Activity Monitoring?
- Parameterized Builds in Jenkins – choosing subversion folders
- Building a custom UI Object with GT Designer 2 for Mitsubishi GOT 1000
- Capture-Replay Regression Testing in Eclipse RCP
- ServiceBrowser: browsing your rails controllers

MeineLinks

- ubigrate Website
- Java Usergroup Saxony

AndereBlogs

- The RTLS Blog
- M2M Blog