WS 2004/05

KV Testen von Softwaresystemen

Ausarbeitung zum Thema:
„JUnit Web-Tools“

LVA-Leiter:
Dr. Christoph Steindl
LVA 246.221

Gruppe:

Christian Bleichenbach (Matr.Nr.: 0056578)
Andreas Hackl (Matr.Nr.: 0056490)
Daniel Pötschko (Matr.Nr.: 0155708)
Christian Schwarzbauer (Matr.Nr.: 0056391)

Inhaltsverzeichnis

1. HttpUnit

1.1. Entwicklung

HttpUnit wurde von Russel Gold 2003 entwickelt. Die aktuelle Version 1.6 released wurde am 3.Oktober 2004 veröffentlicht. HttpUnit ist kostenlos und kann unter Einhaltung der Lizenzbedingungen (MIT Lizenz) verwendet werden. HttpUnit basiert genau wie JUnit auf Java.

1.2. Ziel

HttpUnit ist eine Erweiterung des Testframeworks JUnit. Es stellt Werkzeuge zum Zugreifen verschiedenster Elemente auf einer Weboberfläche zur Verfügung. Es können damit Sessions, Cookies, Tables, Frames etc. behandelt werden. Im Grunde ist HttpUnit einen System- und Akzeptanztest für Webanwendungen. Es basiert auf dem Request- Response (Frage-Antwort) Verfahren.
HttpUnit bietet die Möglichkeit, Webseiten automatisiert aufzurufen. Mittels einer zentralen Klasse *WebConversation* wird ein Browser simuliert, der es ermöglicht, den Inhalt einer angeforderten Webseite zu erhalten. Handelt es sich beim Inhalt um XML, kann dynamisch ein DOM-Tree erzeugt werden.
Liegt HTML vor, stehen zusätzlich Navigationsmöglichkeiten zur Verfügung. Typische HTML-Elemente werden durch Klassen repräsentiert, wie z.B. *WebTable, WebWindow, WebLink* etc. und können bezüglich ihres Inhalts evaluiert werden.

1.3. Funktionalität

HttpUnit ist also ein ausgezeichnetes Werkzeug zum Überprüfen der Funktionalität von Webanwendungen.
Um die Methodik des Testens mit HttpUnit besser aufzeigen zu können, eine kurze Beschreibung einiger Möglichkeiten mit Beispielcode:

1.3.1. Reaktion einer Website

Man erstellt eine WebConversation, stellt ihr eine Anfrage und wartet auf eine Reaktion.

```
WebConversation wc = new WebConversation();
WebRequest     req = new GetMethodWebRequest( "http://www.meterware.com/testpage.html" );
WebResponse   resp = wc.getResponse( req );
```

1.3.2. Einem Link folgen

Die einfachste Weise zur Navigation durch Websites ist das Folgen von Links, die wiederum Reaktionen anderer Websites hervorrufen.

```
WebConversation wc = new WebConversation();
WebResponse   resp = wc.getResponse( "http://www.httpunit.org/doc/cookbook.html" );
WebLink       link = resp.getLinkWith( "response" );
link.click();
WebResponse   jdoc = wc.getCurrentPage();
```

1.3.3. Tabellen prüfen

Viele Webdesigner verwenden Tabellen, um die Struktur einer Website kontrollieren zu können.

```
WebTable table = resp.getTables()[0];
assertEquals( "rows", 4, table.getRowCount() );
assertEquals( "columns", 3, table.getColumnCount() );
assertEquals( "links", 1, table.getTableCell( 0, 2 ).getLinks().length );
```

Weitere Möglichkeiten bietet HttpUnit im Testen von Formularen (mit allen ihren Buttons, Boxen und Menüs) und Frames (siehe http://www.httpunit.org/doc/cookbook.html).

1.4. JavaScript-Unterstützung

Die JavaScript-Unterstützung ist in diesem Stadion noch nicht sehr ausgereift. In Zukunft soll aber die vollständige JavaScript 1.1 unterstützt werden. Browserspezifische Unterstützung ist nicht geplant.

1.5. Beispiel

Die Abbildung 1 zeigt ein einfaches Beispiel für ein HttpUnit Testskript und die Zusammenhänge mit anderen Komponenten.

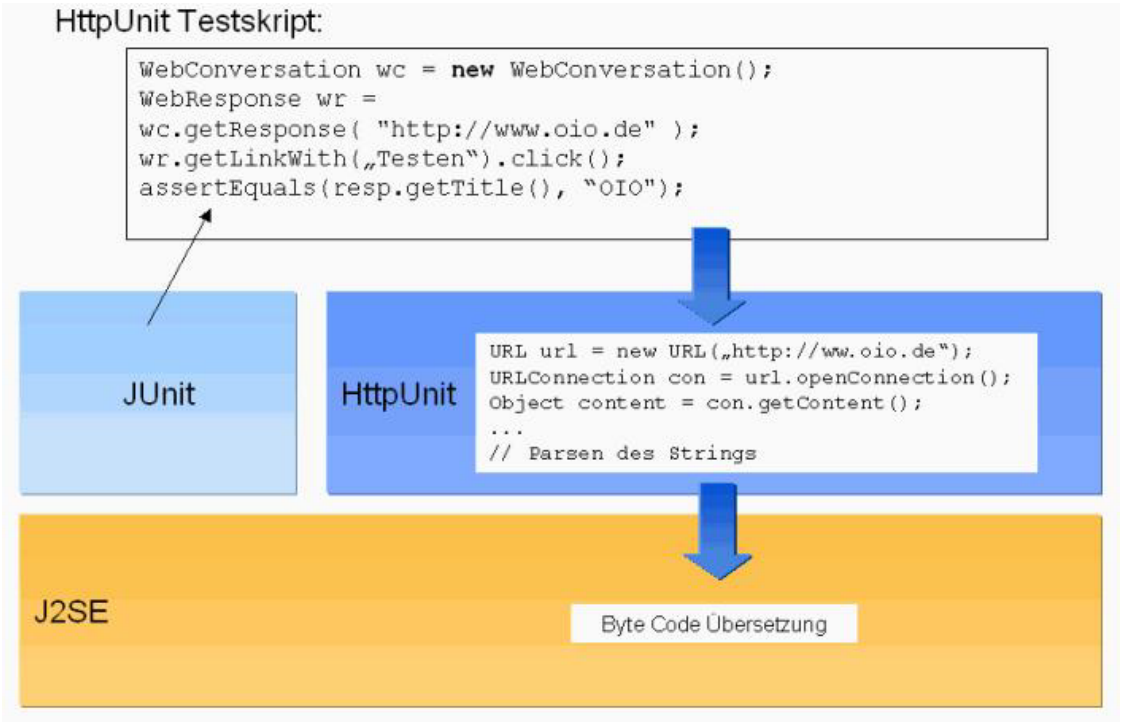


Abbildung 1: Beispiel Testskript HttpUnit

1.6. Quellen

http://www.junit.org/news/extension/web/index.htm
http://www.httpunit.org/
http://www.javamagazin.de/itr/online_artikel/psecom,id,365,nodeid,11.html
http://www.oio.de/m/jax2004-test-driven-development-with-open-source/jax2004-test-driven-development-with-open-source.pdf

2. HtmlUnit

2.1. Entwicklung

HtmlUnit wurde ursprünglich von Mike Bowler von Gargoyle Software entwickelt. Die aktuelle Version 1.3 wurde am 12.November 2004 veröffentlicht. HtmlUnit ist kostenlos und kann unter Einhaltung der Lizenzbedingungen (Gargoyle Lizenz) verwendet werden. HtmlUnit basiert genau wie HttpUnit und JUnit auf Java.

2.2. Ziel

Ähnlich wie HttpUnit kann auch HtmlUnit Webseiten automatisiert aufrufen und deren Inhalt prüfen. HtmlUnit unterscheidet sich jedoch dadurch, dass der Benutzer nicht mit Request-Response Objekten arbeitet, sondern seitenorientiert. Das heißt, der Benutzer beschäftigt sich mit einzelnen Seiten, Formularen und Tabellen.
HtmlUnit bietet sich für das gezielte Testen von statischen Inhalten von Websites, auch im Zusammenspiel mit JavaScript. Dabei bietet es noch speziellere Klassen an, als bei HttpUnit zur Verfügung stehen.

HttpUnit bietet folgende Möglichkeiten:

- Unterstützt Http und Https, wobei zu Https die Java Secure Socket Extension (JSSE: http://java.sun.com/products/jsse/) im Klassenpfad eingebunden sein muss.
- Unterstützung von Cookies
- Unterstützung von Get und Post Methoden bei Submits
- Unterstützt für html-Antworten:
  - Wrapper für HTML-Seiten, um einfachen Zugang zu allen Informationen im inneren zu erhalten
  - Submitting forms
  - Clicking links
  - Walking the DOM model
- Unterstützt Proxy Server
  etc.

2.3. Funktionalität

HtmlUnit ist ebenfalls ein ausgezeichnetes Werkzeug zum Überprüfen der Funktionalität von Webanwendungen.
Um die Methodik des Testens mit HtmlUnit besser aufzeigen zu können, eine kurze Beschreibung einiger Möglichkeiten mit Beispielcode:

2.3.1. Seitentitel prüfen

Die wichtigste Klasse bei HtmlUnit ist die Klasse com.gargoylesoftware.htmlunit.WebClient, welche auch den Startpunkt zum Testen darstellt.
Es wird im Beispiel zunächst ein Webclient angelegt und die HtmlUnit-Website geladen. Es wird kontrolliert, ob der Seitenname auch korrekt ist.

```
public void testHomePage() throws Exception {
    final WebClient webClient = new WebClient();
    final URL url = new URL("http://htmlunit.sourceforge.net");
```

```
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );
}
```

2.3.2. Spezielle Browser imitieren

Um einen speziellen Browser imitieren zu können gibt es die Klasse com.gargoylesoftware.htmlunit.BrowserVersion, wo Parameter von gängigen Browsern abgerufen werden können. Im Beispiel wird der Browser Mozilla mit der Version 1.0 angelegt.

```
public void testHomePage() throws Exception {
    final WebClient webClient = new WebClient(BrowserVersion.MOZILLA_1_0);
    final URL url = new URL("http://htmlunit.sourceforge.net");
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );
}
```

2.3.3. Proxy Server verwenden

Ein Konstruktor ermöglicht die Angabe eines Proxy Servers beim Anlegen eines WebClients.

```
public void testHomePage() throws Exception {
    final WebClient webClient = new WebClient(
          BrowserVersion.MOZILLA_1_0, "http://myproxyserver", 8000);
    final URL url = new URL("http://htmlunit.sourceforge.net");
    final HtmlPage page = (HtmlPage)webClient.getPage(url);
    assertEquals( "htmlunit - Welcome to HtmlUnit", page.getTitleText() );
}
```

Weitere Möglichkeiten von HtmlUnit siehe http://htmlunit.sourceforge.net/getting_Started.html.

2.4. JavaScript-Unterstützung

Seit dem Release 1.1 unterstützt HtmlUnit die Kernfunktionen der JavaScript-Sprache mit Hilfe der *Rhino JavaScript Engine*. Auch Browserspezifische Hostobjekte werden zur Zeit implementiert.
Es wir das Augenmerk nur auf die allgemein gebräuchlichen, viel verwendeten JavaScript-Funktionen gelegt.
Um die JavaScript-Unterstützung für einen Webclient auszuschalten, braucht man nur: WebClient.setJavaScriptEnabled(false) aufrufen. Bei kompletter Deaktivierung einfach das js.jar aus dem Klassenpfad wieder rauslöschen.

2.5. Beispiel

Die Abbildung 2 zeigt ein einfaches Beispiel für ein HtmlUnit Testskript und die Zusammenhänge mit anderen Komponenten.

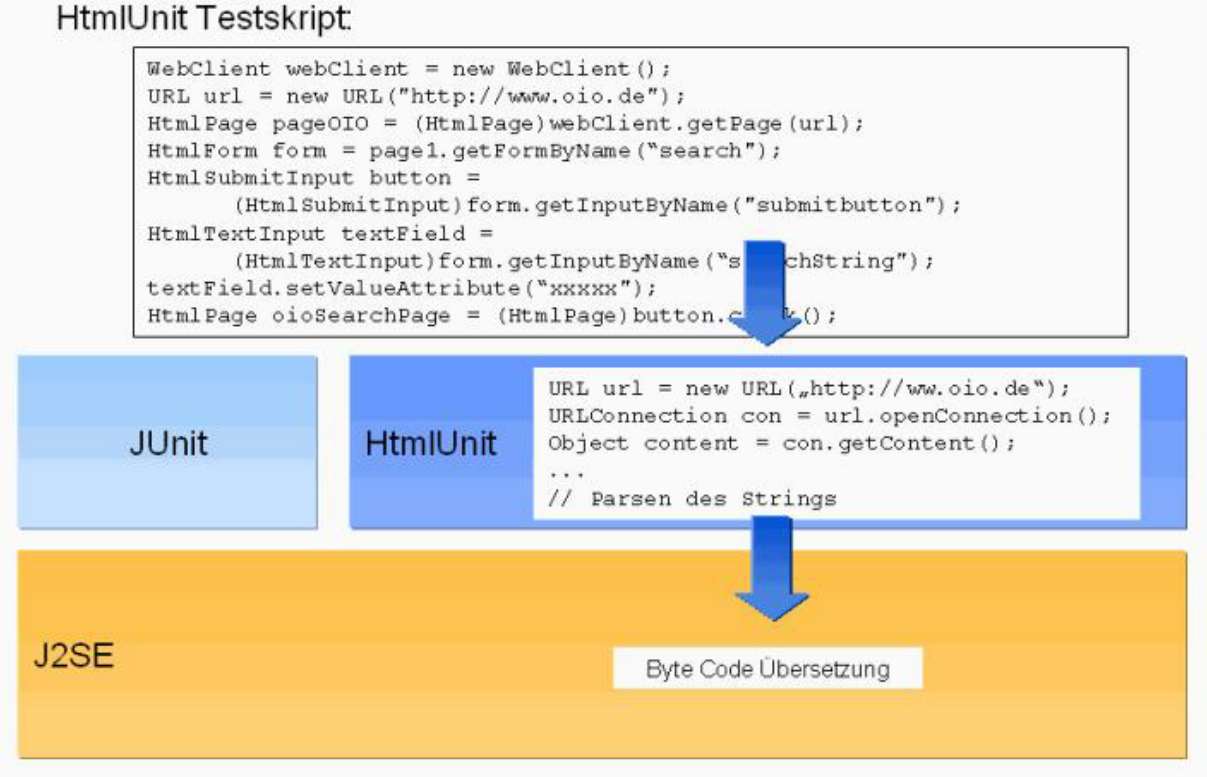


Abbildung 2: Beispiel Testskript HtmlUnit

2.6. Quellen:

http://www.junit.org/news/extension/web/index.htm
http://htmlunit.sourceforge.net/
http://www.javamagazin.de/itr/online_artikel/psecom,id,365,nodeid,11.html
http://www.oio.de/m/jax2004-test-driven-development-with-open-source/jax2004-test-driven-development-with-open-source.pdf

3. jWebUnit

3.1. Entwicklung

jWebUnit wurde ursprünglich von Martijn Dashorst entwickelt. Die aktuelle Version 1.2 wurde am 12.Juni 2004 veröffentlicht. jWebUnit ist kostenlos und kann unter Einhaltung der Lizenzbedingungen verwendet werden. jWebUnit basiert genau wie HttpUnit, HtmlUnit und JUnit auf Java.

3.2. Ziel

jWebUnit ist ein Framework, das die Erstellung von Akzeptanztests für Webanwendungen unterstützen soll. jWebUnit entwickelte sich während der Benutzung von HttpUnit und Junit. Es ist ein Ergebnis langer Refactoringsessions, um Codeverdopplungen und ähnliches zu entfernen.
jWebUnit bietet eine ausgereifte API für die Navigation von Websites kombiniert mit der Möglichkeit, die Korrektheit einer Anwendung zu verifizieren. Dies beinhaltet die Navigation zwischen Links, Formularen, Submissions etc. und andere typische Webanwendungen. jWebUnit macht sich auch HttpUnit im Hintergrund zu nutze (siehe Abbildung 4).
jWebUnit kann auch mit Ant und Maven verwendet werden (siehe http://jwebunit.sourceforge.net/building-ant.html und http://jwebunit.sourceforge.net/ building-maven.html)
Die folgende Abbildung 3 zeigt die Codeersparnis und Kürze von jWebUnit im Gegensatz zu HttpUnit und JUnit allein. Der Test zeigt eine Google-Suche zum Finden der HttpUnit Website.

HttpUnit und JUnit

```
package net.sourceforge.jwebunit.sample;

import junit.framework.TestCase;
import com.meterware.httpunit.WebResponse;
import com.meterware.httpunit.WebConversation;
import com.meterware.httpunit.WebForm;
import com.meterware.httpunit.WebRequest;

public class SearchExample extends TestCase {

   public void testSearch() throws Exception {
      WebConversation wc = new WebConversation();
      WebResponse resp = wc.getResponse( "http://www.google.com");
      WebForm form = resp.getForms()[0];
      form.setParameter("q", "HttpUnit");
      WebRequest req = form.getRequest("btnG");
      resp = wc.getResponse(req);
      assertNotNull(resp.getLinkWith("HttpUnit"));
      resp = resp.getLinkWith("HttpUnit").click();
      assertEquals(resp.getTitle(), "HttpUnit");
      assertNotNull(resp.getLinkWith("User's Manual"));
   }
}
```

jWebUnit

```
package net.sourceforge.jwebunit.sample;

import net.sourceforge.jwebunit.WebTestCase;

public class JWebUnitSearchExample extends WebTestCase {

   public JWebUnitSearchExample(String name) {
```

```
        super(name);
    }

    public void setUp() {
        getTestContext().setBaseUrl("http://www.google.com");
    }

    public void testSearch() {
        beginAt("/");
        setFormElement("q", "httpunit");
        submit("btnG");
        clickLinkWithText("HttpUnit");
        assertTitleEquals("HttpUnit");
        assertLinkPresentWithText("User's Manual");
    }
}
```

Abbildung 3: Unterschied HttpUnit und jWebUnit

3.3. Funktionalität

jWebUnit ist ebenfalls ein ausgezeichnetes Werkzeug zum Überprüfen der Funktionalität von Webanwendungen.
Um die Methodik des Testens mit jWebUnit besser aufzeigen zu können, eine kurze Beschreibung einiger Möglichkeiten mit Beispielcode:

3.3.1. Erstellen eines Testfalles

Es gibt zwei verschiedene Möglichkeiten einen WebTestCase zu erstellen.

```
import net.sourceforge.jwebunit.WebTestCase;

public class ExampleWebTestCase extends WebTestCase {
    public ExampleWebTestCase(String name) {
        super(name);
    }
}
```

```
import junit.framework.TestCase;
import net.sourceforge.jwebunit.WebTester;

public class ExampleWebTestCase extends TestCase {
    private WebTester tester;

    public ExampleWebTestCase(String name) {
        super(name);
        tester = new WebTester();
    }
}
```

3.3.2. Navigation von Webanwendungen

Zuerst muss der Server angegeben werden, auf dem die Webanwendung läuft.

```
public ExampleWebTestCase(String name) {
        super(name);
        getTestContext().setBaseUrl("http://myserver:8080/myapp");
    }
```

oder:

```
public void setUp() throws Exception {
        getTestContext().setBaseUrl("http://myserver:8080/myapp");
    }
```

Dann kann durch die Links hindurchnavigiert werden.

```
<a id="addLink" href="/addPage">Add Widget</a>
<a id="editlink" href="/editPage">Edit Widget</a>
```

```
    public void testMainPageLinks() {
        beginAt("/mainPage");
        assertLinkPresent("addLink");
        clickLink("addLink");
        assertTitleEquals("Widget Add Page");
        beginAt("/mainPage");
        assertLinkPresentWithText("Edit Widget");
        clickLinkWithText("Edit Widget");
        assertTitleEquals("Widget Edit Page");
    }
```

Weitere Möglichkeiten von jWebUnit siehe http://jwebunit.sourceforge.net/quickstart.html.

3.4. Beispiel

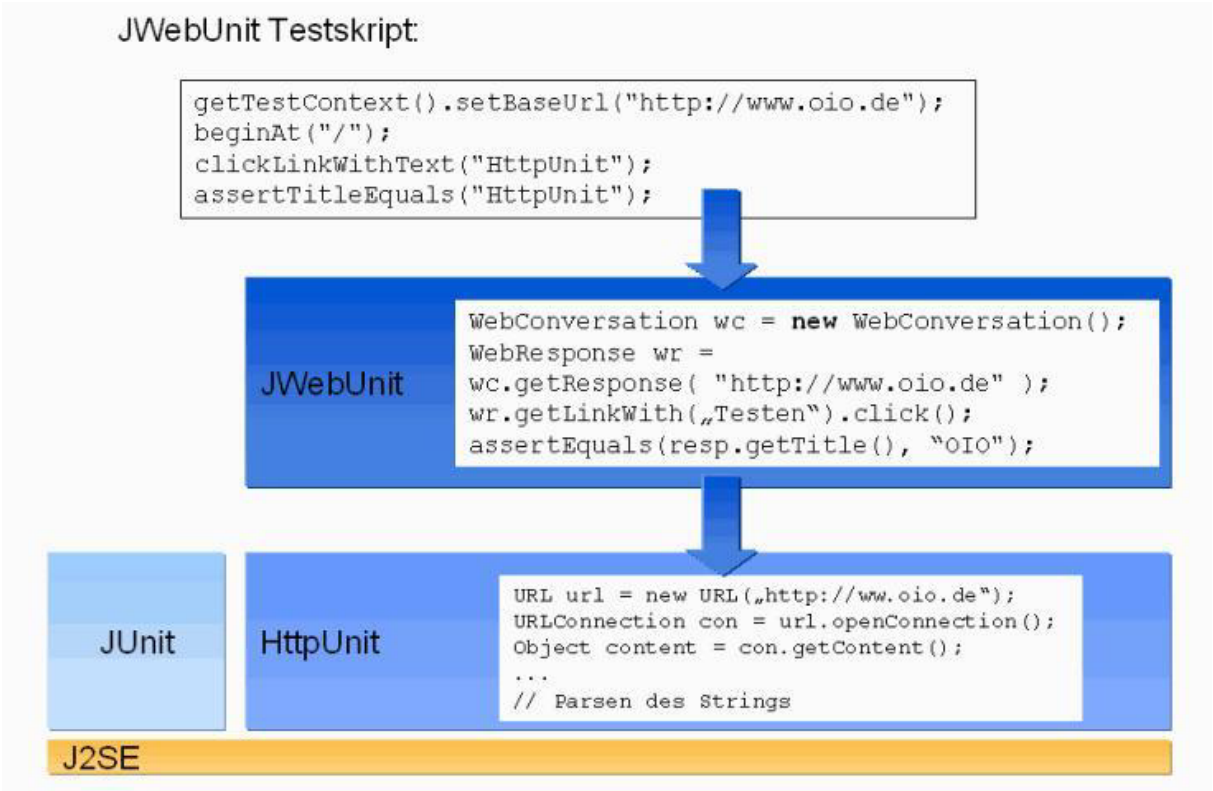


Abbildung 4: Beispiel Testskript JWebUnit

3.5. Quellen:

http://www.junit.org/news/extension/web/index.htm
http://jwebunit.sourceforge.net/
http://www.oio.de/m/jax2004-test-driven-development-with-open-source/jax2004-test-driven-development-with-open-source.pdf

4. TagUnit

4.1. Entwicklung

TagUnit wurde ursprünglich von Sam Dalton und Simon Brown entwickelt. Die aktuelle Version 1.0.1 wurde am 11.März 2004 veröffentlicht. TagUnit ist kostenlos und kann unter Einhaltung der Lizenzbedingungen verwendet werden.

4.2. Ziel

Das Testen von Servlets und Java Server Pages gestaltet sich trotz Tools wie Cactus, JunitEE und HttpUnit als schwierig, wenn Custom Taglibs verwendet werden. Es macht wenig Sinn, Custom Tags durch den Aufruf ihrer Methoden zu testen, da sie als Komponente in einem speziellen Umfeld verwendet werden.
TagUnit ermöglicht es, Taglibs in diesem Umfeld zu testen. Es werden Assertions zur Verfügung gestellt, die den Inhalt und die Seiteneffekte (Attribute, Cookies, etc.) prüfen, die durch ein Custom Tag erzeugt werden.

"In the same way that JUnit allows us to write unit tests for Java classes, TagUnit allows us to unit test JSP custom tags, inside the container. In essence, TagUnit is a tag library for testing custom tags within JSP pages" (http://www.tagunit.org/tagunit/faq.jsp).

TagUnit kann auch mit Ant verwendet werden (http://www.tagunit.org/tagunit/docs/tagunit-with-ant.pdf).

4.3. Beispiel

Man kann zum Beispiel die Beschreibungen von Tags testen.

```
the JSTL <c:set> tag can be tested.
  <tagunit:assertBodyContent name="JSP"/>

  <tagunit:assertAttribute name="value" required="false" rtexprvalue="false"/>
  <tagunit:assertAttribute name="var" required="false" rtexprvalue="false"/>
  <tagunit:assertAttribute name="scope" required="false" rtexprvalue="false"/>
  <tagunit:assertAttribute name="target" required="false" rtexprvalue="false"/>
  <tagunit:assertAttribute name="property" required="false" rtexprvalue="false"/>
```

Weitere Möglichkeiten von TagUnit siehe http://www.tagunit.org/tagunit/examples.jsp.

4.4. Quellen

http://www.junit.org/news/extension/web/index.htm
http://www.javamagazin.de/itr/online_artikel/psecom,id,365,nodeid,11.html
http://www.tagunit.org/tagunit/index.jsp

5. XML Test Suite

5.1. Entwicklung

Die „XML Test Suite“ wurde unter David Rutter entwickelt und am 4. November 2002 auf Junit.org vorgestellt. Seit 6. Mai 2003 ist die „XML Test Suite“ in der Version 1.2 unter der GNU Library bzw. Lesser General Public License (LGPL) frei verfügbar. Die Applikation basiert auf Java und ist daher auf diversen Windows und Unix-basierten Systemen nutzbar.

5.2. Ziel

Die „XML Test Suite“ dient dem Testen von Web-Applikationen. Die Tests des Tools sind vor allem auf das Prüfen der Struktur, des Inhaltes und der Funktionalität von HTML-Dateien gerichtet. Hierbei ist es möglich Test-Szenarien in XML zu schreiben. Dabei kann auf JavaScript-Variablen, XPath-Ausdrücke und Datenbankabfragen eingegangen werden.
Da die Tests in XML geschrieben werden sind keine Java-Kenntnisse nötig. Es reicht sich mit HTML und XML auszukennen. Durch die Verwendung von XPath kann man sich auf den wichtigen Teil des HTML-Dokumentes konzentrieren und das Layout im Hintergrund lassen.
Die „XML Test Suite“ ist in Kombination mit JUnit verwendbar, es ermöglicht eine testbasierte Entwicklung sowie ein funktionales Testen.
Die „XML Test Suite“ ist vor allem für Tester und Web-Entwickler mit geringem Hintergrundwissen in Java geeignet.

5.3. Beispiel

Um nun Tests mit der „XML Test Suite“ durchführen zu können müssen nun XML-Dateien, die den Prüfungscode enthalten, erstellt werden. Das folgende Beispiel zeigt eine XML-Datei die alle Links einer Website überprüft:

```
<?xml version="1.0" ?>
<!DOCTYPE testSpec SYSTEM "test.dtd">
<testSpec>
      <steps>
            <fetch url="http://server/..."/>
             <verifylinks/>
      </steps>
</testSpec>
```

In dieser XML- Datei wird auf eine dtd-Datei mit dem Namen „test.dtd“ verwiesen. In dieser sind alle Möglichkeiten, mit welcher ein Testfall geschrieben werden kann definiert. Auf der angegebenen Website werden dann alle Links rekursiv überprüft. Weitere Möglichkeiten der „XML Test Suite“ zeigt die folgende Datei. Hierbei wird eine Website auf Fehler im HTML-Code überprüft. Weiters überprüft man ob die Website ein Frameset ist, welches die Frames „top“ und „main“ beinhaltet.

```
<?xml version="1.0" ?>
<!DOCTYPE testSpec SYSTEM "test.dtd">
<testSpec>
      <steps>
```

```
            <fetch url="http://server/..."/
              htmlwarnings="true">
            <verify>
                <frameset name="mainframeset">
                  <frame name="top" url="..." />
                  <frame name="main" url="..." />
                </frameset>
             </verify>
      </steps>
</testSpec>
```

Die Ausführung der beiden definierten XML-Dateien kann direkt von der Kommandozeile aus erfolgen. Weiters ist es, wie oben erwähnt, möglich die XML-Dateien in eine JUnit Test Suite einzubinden. Die Beispiele zeigen nur einen kleinen Teil der Möglichkeiten mit der „XML Test Suite“. Komplexer wäre es z.B. diverse Verbindungen von Web-Applikationen mit Datenbanken zu überprüfen wie es in der folgenden Datei dargestellt ist:

```
<?xml version="1.0" ?>
   <!DOCTYPE testSpec SYSTEM "test.dtd">
   <testSpec  application="..." baseURL="http://server/">
         <database dbdriver = "..." dbconnection = "...">
              <dbTable name="Contacts" identity="ContactId"/>
              <dbTable name="Suppliers" identity="SupplierId"/>
              <dbTable name="log" identity="LogId"/>
         </database>
          <steps stepid="testAll">
                 <!-- Call a stored procedure to enter data into database
                       specify that an insertion is expected in Suppliers and Contacts tables-->
                 <exec query="exec insertNewSupplier 'SupplierName', 'ContactName' ">
                    <dbInsert id="newSupplierId" dbTable="Suppliers"/>
                    <dbInsert id="newContactId" dbTable= "Contacts"/>
                 </exec>

                <!-- Verify that a search by supplier gives the expected contact record-->
                <set name="supplierId" query="
                       select contactid fromSuppliers sjoin Contacts con s.contactid= c.idwhere
name= 'SupplierName' "/>  < BR >                    <eval  expr="${contactid} = ${newContac-
tId}"
         </steps>
   <testSpec/>
```

Eine vollständige Liste der Möglichkeiten zeigen die Java-Docs zur API und der DTD-Datei unter:
http://xmltestsuite.sourceforge.net/docs/api/index.html und
http://xmltestsuite.sourceforge.net/docs/dtd.html

5.4. Quellen

http://www.junit.org/news/extension/web/index.htm
http://xmltestsuite.sourceforge.net/
http://sourceforge.net/project/showfiles.php?group_id=66414

6. KaCoMa

6.1. Entwicklung

KaCoMa steht für Kaarle's ContentManagement und wurde unter Kaarle Kaila entwickelt. KaCoMa ist seit dem 1. Mai 2002 auf JUnit.org veröffentlicht. Die Applikation ist in Java implementiert und untersteht der GNU General Public License (GPL). Die aktuelle Version ist 1.3.

6.2. Ziel

Die Idee hinter KaCoMa ist ein Content Management System für Websites mit Datenhaltung am Applikationsserver zu bieten. Diese Idee stellt aber nur eine Zukunftsvorstellung dar. KaCoMa wird im derzeitigen Entwicklungsstand nicht als CMS verwendet.

KaCoMa verwendet diverse andere Technologien im Tutorial, diese sind:

- XML – für das Speichern von Dateninhalten
- JDOM – um Daten aus XML-Dateien zu lesen
- ANT – für das Ausführen der Tests im Tutorial
- JUNIT – zum Testen von KaCoMa-Methoden
- JUNITEE – für Serverseitige Tests
- Custom tag library – innerhalb von JSPs um auf Daten der KaCoMa-library zuzugreifen
- CACTUS – um eine Verbindung zwischen JUnit und Web-Anwendungen zu schaffen

6.3. Funktionalität

Bei der Installation von KaCoMa ist ein Ant build Script auszuführen. Weiters ist für die Benutzung ein Applikationsserver (z.B. Tomcat) nötig. Die Software soll in Zukunft aus einer eigenen Datenbank bestehen, zurzeit werden aber noch persistente XML-Dateien verwendet. Ist die Software eingerichtet soll sie zwei Aspekten dienen:

- Die Verwendung von Software für Unit Tests zu erlernen.
- Um „custom tag libraries“ zu erstellen.

Die Software besteht aus mehreren Teilen. Einer Datenbank (noch XML-Dateien) um Datenbestände an dynamische Websites zu liefern. Der Applikation an sich, um Datenbankinhalte auszulesen, je nach Anfragen der Website. Aus JUnit Testfällen, um Methoden der KaCoMa-Applikation zu testen. Aus einer tag-library, die von JSPs verwendet wird um Daten von der KaCoMa Applikation zu erhalten. Aus Cactus Testfällen, welche nötig sind tag-libraries zu testen, da JUnit Webressourcen nicht erreichen kann. Und aus diversen Webinhalten, dargestellt durch JSPs. Alle diese Teile sollten zusammen das CMS KaCoMa bilden.
KaCoMa bietet aber nur eine Einführung in die genannten Technologien mit Hinweisen auf deren Wichtigkeit. Mit KaCoMA können die Technologien praxisnah, anhand mehrerer Beispiele gelernt werden. Die ursprüngliche Geschäftslogik die in der KaCoMa Applikation steckt sowie das „tag library model“ von KaCoMa zeigt das folgende UML-Diagramm:

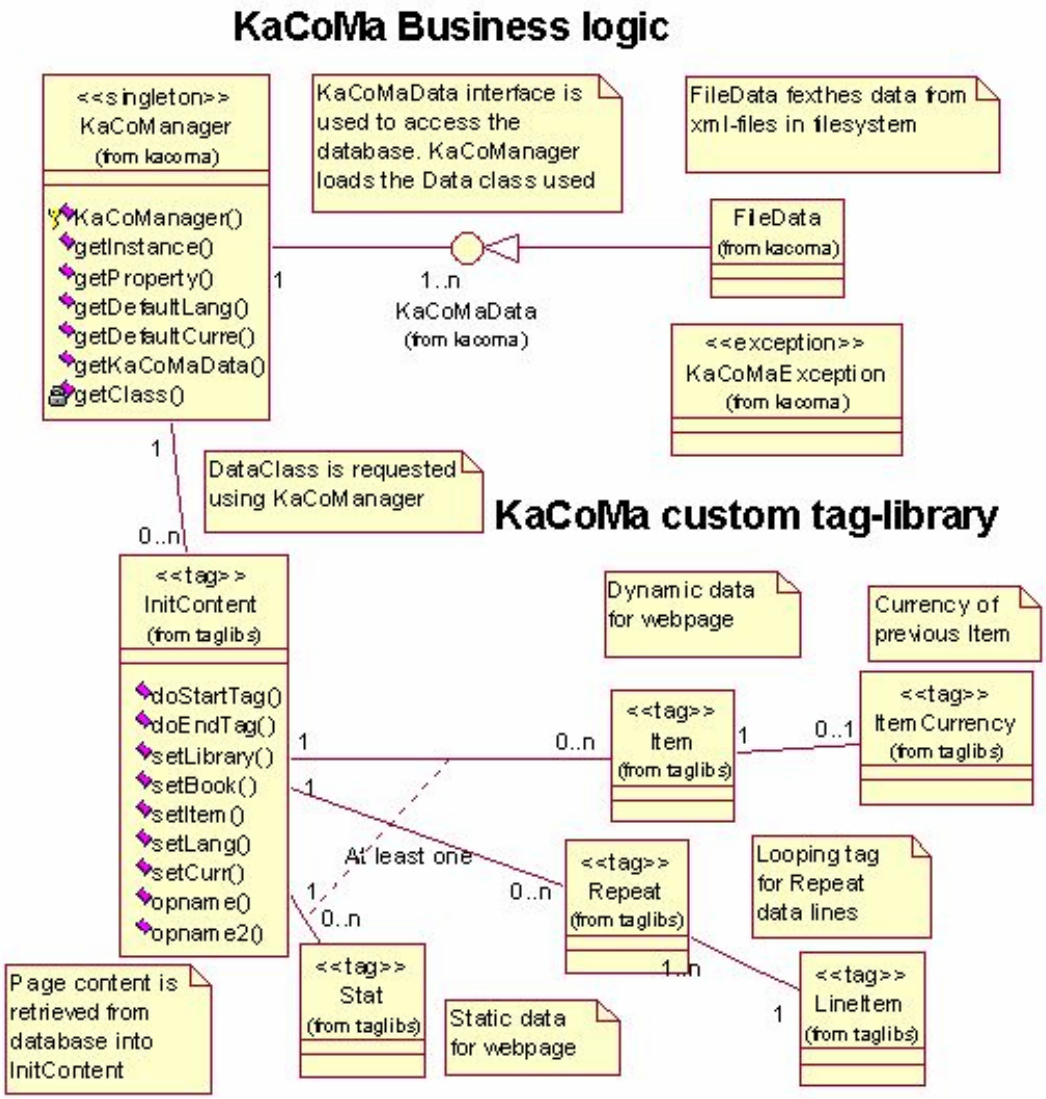


Abbildung 5: KaCoMa Geschäftslogik und custom tag-library

Der KaCoMaManager ladet die korrekten Klassen für die Anwendung. Weiters ist er für die Verwaltung der Applikationseinstellungen verantwortlich.

Exceptions werden in der Anwendung als KaCoMaExceptions geworfen. Diese Fehler werden von einer einfachen JSPs gefangen, welche den Fehler in einer benutzerfreundlichen Art darstellen soll.

Dateninhalte können nur von Klassen erreicht werden die die Schnittstelle KaCoMaData implementieren.

KaCoMaCommands ist eine Helferklasse um benutzerspezifische Parameter zu speichern und weiterzuleiten.

FileData wird verwendet um auf XML-Dateien auf der Festplatte zugreifen zu können.

Weiters existiert ein UML-Diagramm über die KaCoMa Testfälle, welches in der Abbildung 6 dargestellt wird:

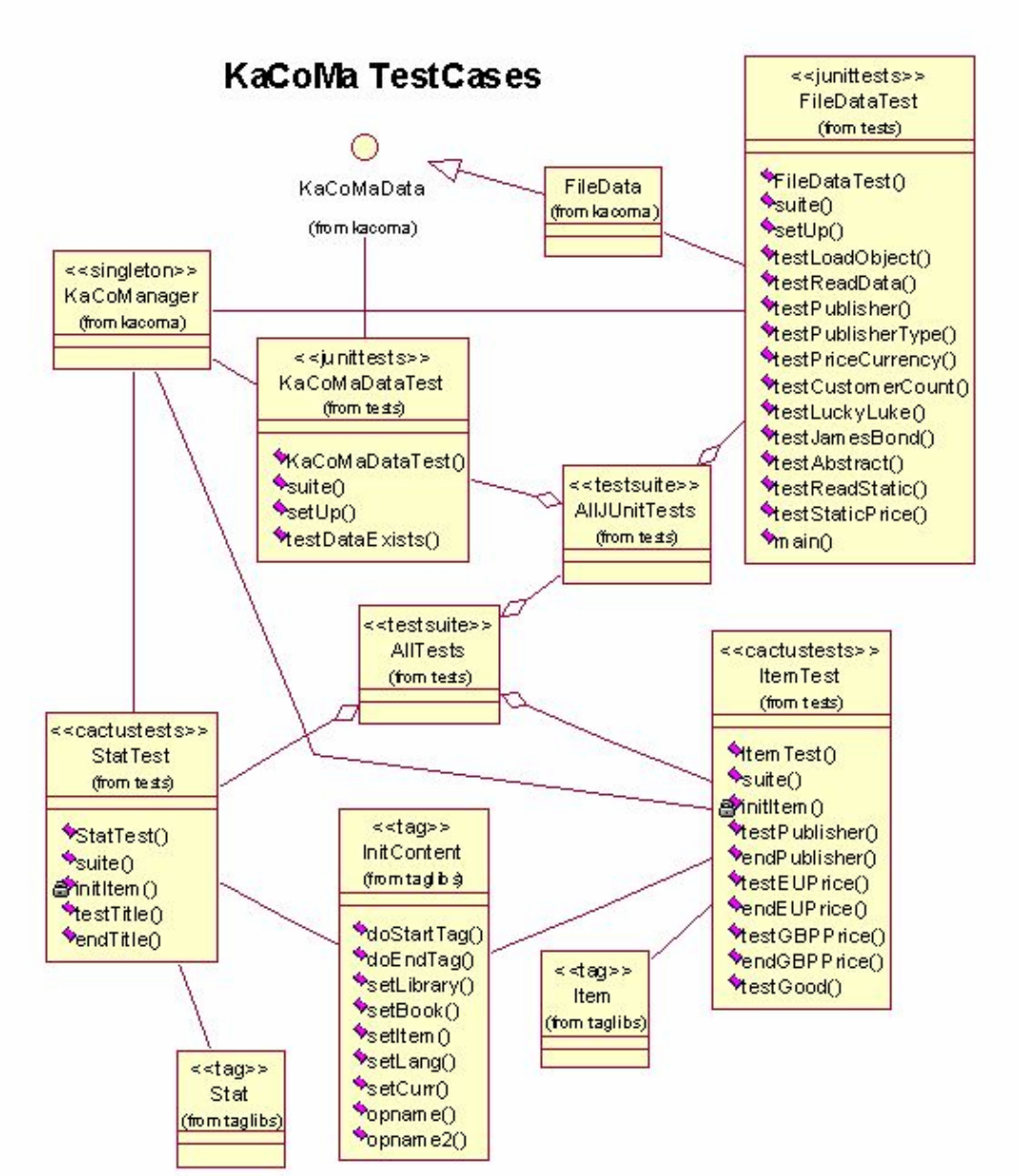


Abbildung 6: KaCoMa Testfälle

KaCoMaDataTest sind Tests für Klassen die die Schnittstelle KaCoMaData implementieren.

FileDataTest sind Tests für die FileData Klassen.

KaCoMa ist in der aktuellen Version noch nicht ausgereift, und erfüllt daher nur einen Teil der eigentlichen Vision des Projektes.

6.4. Quellen

http://www.junit.org/news/extension/web/index.htm
http://kacoma.sourceforge.net/
http://sourceforge.net/projects/kacoma

7. Epicentric JUnit JSP test harness

7.1. Entwicklung

Epicentric JUnit JSP test harness wurde von David McCue designed. Es ist in der Version 1.0 verfügbar und wurde am 23. Dezember 2001 auf JUnit.org veröffentlicht.

7.2. Ziel

Epicentric stellt eine intuitive Schnittstelle zum Ausführen von Web basierten JUnit Tests und zum Darstellen von Ergebnissen zur Verfügung. Es erlaubt Testklassen auf einen Server zu laden und diese dort als Plattformtests auszuführen. Es bietet einen effektiven Weg zum Testen von Java Code innerhalb bestimmter JSP-Frameworks. Epicentric ist sozusagen eine Web-Schnittstelle zum Testen.
Die Installation erfolgt auf einem Server, auf diesem wird dann die Web-Schnittstelle eingerichtet. Ist die Software installiert können Testklassen automatisch über die Web-Schnittstelle oder manuell durch kopieren hinzugefügt werden.

7.3. Funktionalität

Nun können Tests ausgeführt werden. Hierzu werden die folgenden vier Ausführungsmöglichkeiten unterschieden:

- Ausführen einzelner Tests
- Ausführen einzelner Testmethoden
- Ausführen aller Testklassen eines Packages
- Rekursives ausführen aller Testklassen eines Packages und derer Sub-Packages

Sind die Tests durchgeführt können die Ergebnisse betrachtet werden. Weiters werden diese in einer Ergebnisdatei gespeichert.

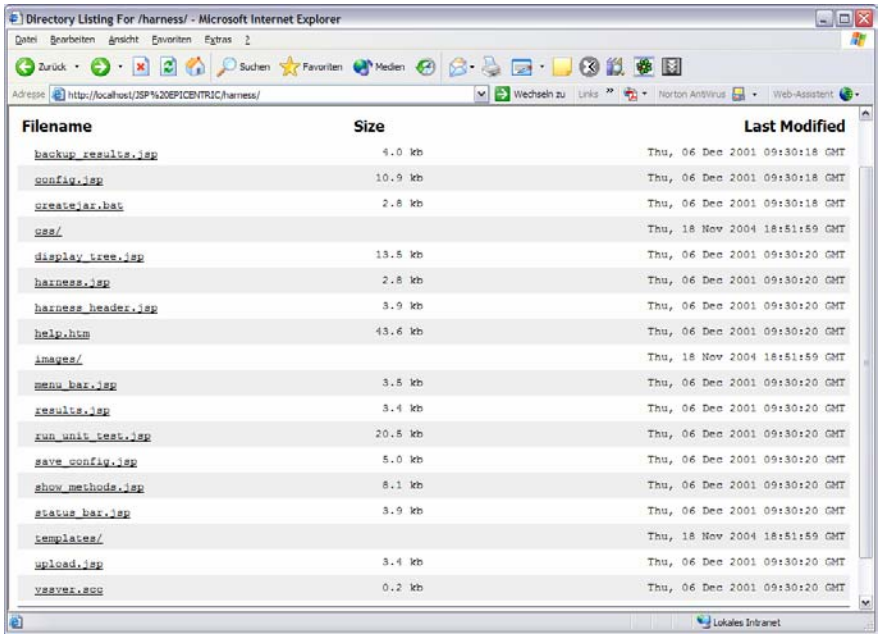


Abbildung 7: Epicentric Web-Interface

Die Abbildung 7 zeigt das Epicentric JUnit Web-Interface, mit einer Auflistung der verwendbaren JSP-Dateien. Für Epicentric gibt es leider keine veröffentlichte Dokumentation, sondern nur eine Installationsanleitung.

7.4. Quellen

http://www.junit.org/news/extension/web/index.htm

8. Simple classes to help test JSPs

8.1. Entwicklung

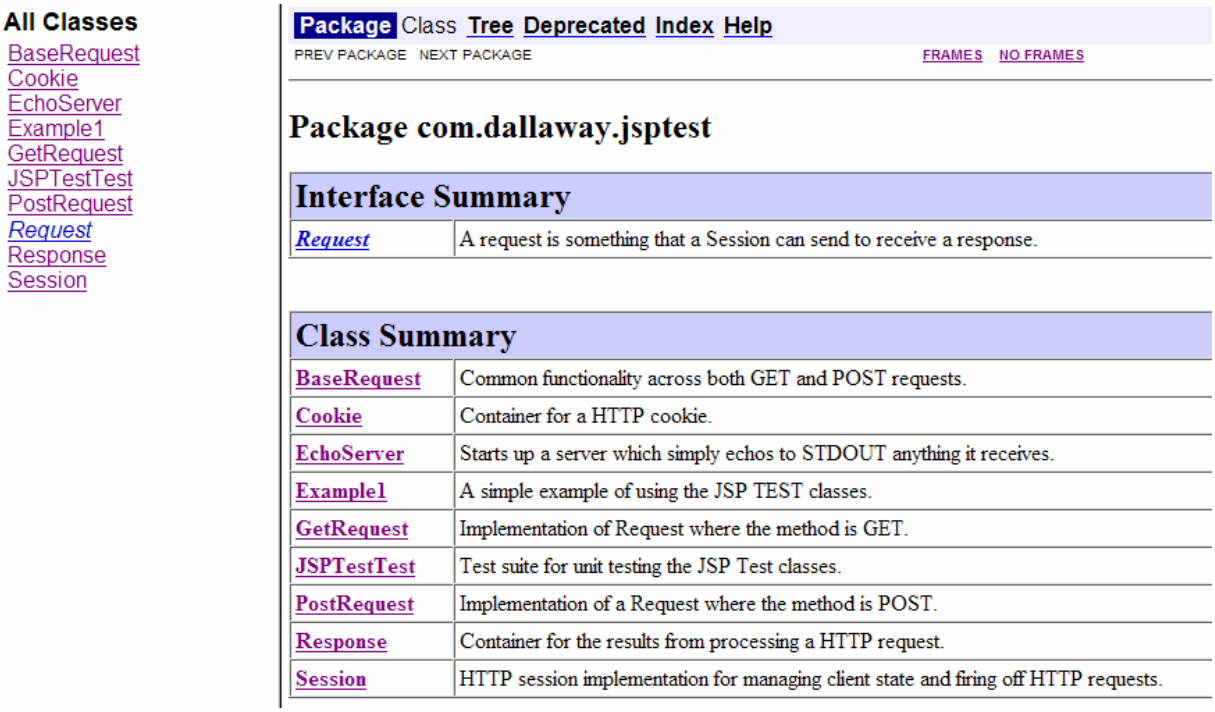
Dies ist eine Sammlung von Java Klassen die ein automatisches Testen von JSPs ermöglichen soll. Die Sammlung ist aktuell unter der Version 6 verfügbar, und wurde erstmals am 19. Juni 2001 auf JUnit.org veröffentlicht.

8.2. Ziel

Die Klassen laufen unabhängig von einem Web-Server oder einer JSP Servlet Engine. Sie sollten jedoch mit einem Testwerkzeug wie JUnit oder HTTPUnit verwendet werden.

8.3. Funktionalität

Die Idee hinter der Sammlung war, diverse Website-Tests automatisch ablaufen zu lassen, nachdem man Änderungen am Code durchgeführt hat. Somit können diverse Tests mittels JUnit auf Websites ausgerichtet werden. Abbildung 8 zeigt die Klassen der Sammlung und welche Möglichkeiten mit diesen zum Testen entstehen:



All Classes
BaseRequest
Cookie
EchoServer
Example1
GetRequest
JSPTestTest
PostRequest
Request
Response
Session

Package Class Tree Deprecated Index Help
PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package com.dallaway.jsptest

Interface Summary	
Request	A request is something that a Session can send to receive a response.

Class Summary	
BaseRequest	Common functionality across both GET and POST requests.
Cookie	Container for a HTTP cookie.
EchoServer	Starts up a server which simply echos to STDOUT anything it receives.
Example1	A simple example of using the JSP TEST classes.
GetRequest	Implementation of Request where the method is GET.
JSPTestTest	Test suite for unit testing the JSP Test classes.
PostRequest	Implementation of a Request where the method is POST.
Response	Container for the results from processing a HTTP request.
Session	HTTP session implementation for managing client state and firing off HTTP requests.

Abbildung 8: JSP-Test Klassen

Tests können nun in die Klasse JSPTestTest eingefügt werden. Die Ausführung dieser erfolgt mittels JUnit. Die JUnit Ausgabe zeigt nun ob die definierten Tests erfolgreich waren oder nicht, wie in Abbildung 9 dargestellt:

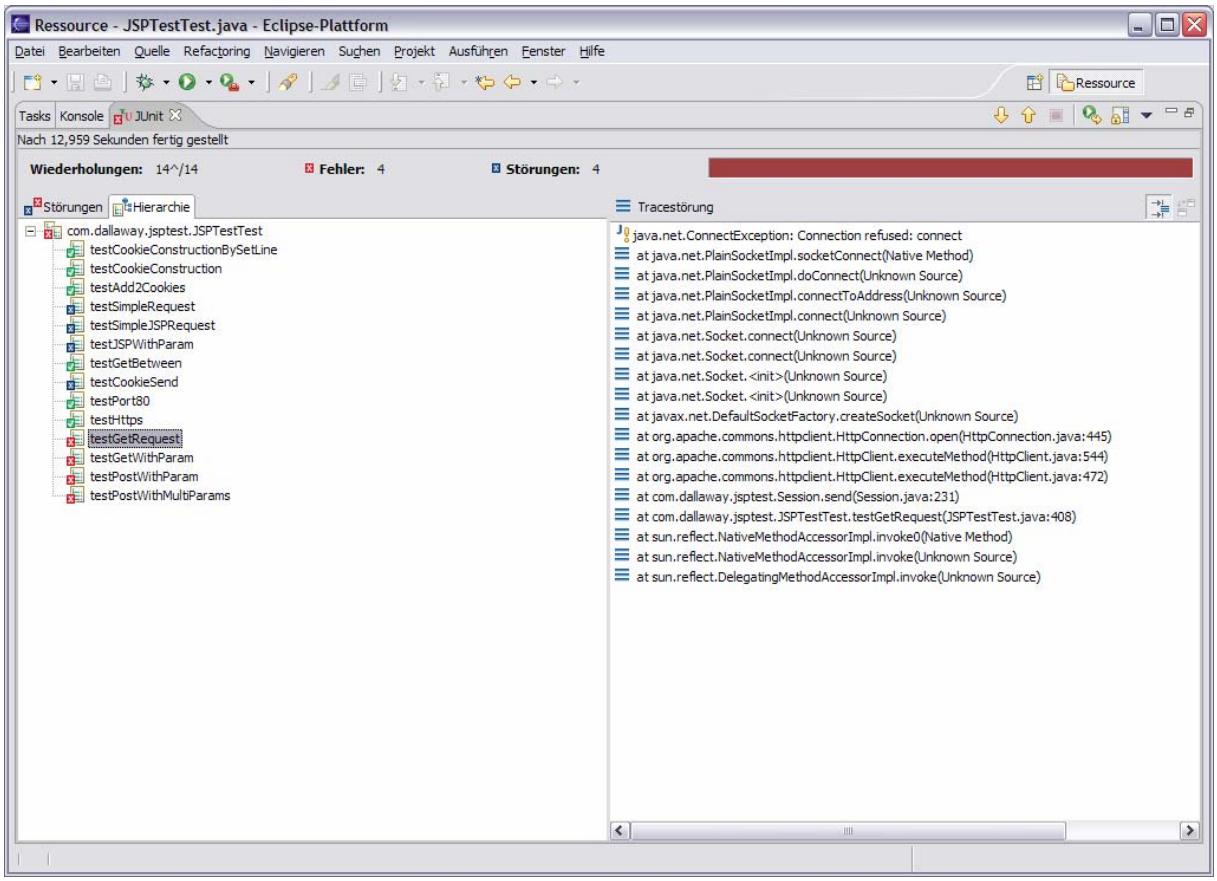


Abbildung 9: JUnit Ausgabe JSPTestTest.java

Die Klassensammlung bietet also eine Unterstützung zum Testen von Websites mit Hilfe eines Testwerkzeuges wie JUnit. Eine einfache Handhabung ist gewährleistet, da keinerlei aufwendige Installationsvorgänge durchgeführt werden müssen. Für die Erstellung der Tests bedarf es jedoch an Java-Erfahrung.
Für weitere Informationen und Fragen zu „Simple classes to help test JSPs" gibt es auch eine Mailingliste auf: http://www.dallaway.com/jsptest/

8.4. Quellen

http://www.junit.org/news/extension/web/index.htm
http://www.dallaway.com/jsptest/

9. Canoo WebTest

9.1. Entwicklung

Canoo WebTest ist ein Open-Source Projekt, das für das automatisierte Testen von Web-Applikationen entwickelt wurde. Es wurde von der Canoo Engineering AG, die ihren Standort in Basel hat, entwickelt. Es wurde am 14. Februar 2002 auf junit.org hinzugefügt und ist im Moment in der stabilen Version 1.6 erhältlich. Die Software untersteht der Lizenz auf folgender Seite: http://webtest.canoo.com/manual/license.html.

9.2. Ziel

Testen ist ein wichtiger Bestandteil jedes Programmier-Projekts und im speziellen für Web-Anwendungen ist es unbedingt erforderlich. Canoo WebTest, das auf *HttpUnit* aufbaut, ermöglicht es, ohne Programmiererfahrung einfach Testfälle für solche Web-Anwendungen zu erstellen.

Das Tool ermöglicht im speziellen folgende Aufgabestellungen:
- reduzieren der Defekt-Rate der Web-Applikation
- messen der extern beobachtbaren Qualität der Applikation
- darstellen des Entwicklungs-Fortschritts anhand der funktionierenden Use-Cases
- abschätzen, ob die Applikation sicher und ohne Komplikationen arbeitet
- einfaches und schnelles erstellen von Tests und unbeaufsichtigtes automatisches ausführen dieser Tests

9.3. Funktionalität

Canoo WebTest ist in der Lage, Testschritte, wie zum Beispiel folgende durchzuführen:
- gehe zur Login-Seite
- überprüfe, ob der Titel der Seite „Login Page“ lautet
- setze den Namen „scott“ in das Feld für den Usernamen
- setze als Passwort „tiger“ in das Passwort-Feld
- drücke den OK Button
- überprüfe, ob der Titel der Seite „Home Page“ lautet

Dieses einfache Beispiel zeigt bereits, dass es unbedingt notwendig ist, dass diese Schritte in genau dieser Reihenfolge ausgeführt werden und während einer Sitzung des Benutzers. Ein solches Szenario wird meist als Use Case bezeichnet.

Dieses Beispiel kann nun einfach in eine passende Darstellung für Canoo WebTest gebracht werden:

```
<target name="login">
      <testSpec name="normal">
            &config;
            <steps>
            <invoke stepid="get Login Page"
                  url="login.jsp" />
            <verifytitle stepid="we should see the login title"
                  text="Login Page"/>
```

```
            <setinputfield stepid="set user name"
                  name="username"
                  value="scott"/>
            <setinputfield stepid="set password"
                  name="password"
                  value="tiger"/>
            <clickbutton stepid="Click the submit button"
                  label="let me in" />
            <verifytitle stepid="Home Page follows if login ok"
                  text="Home Page"/>
            </steps>
      </testSpec>
</target>
```

Wie man sofort sieht, werden WebTests in XML spezifiziert. Die Erstellung mithilfe eines XML-Editors ist somit sehr einfach und aufgrund der dazu erhältlichen *WebTest.dtd* ist auch Syntax-Highlighting und Code-Completion möglich, sofern der XML-Editor dies unterstützt.
Dem aufmerksamen Beobachter fällt sicher sofort der Befehl *&config;* auf. Dieses XML-Entity verweist auf den Inhalt einer anderen Datei. Während der Ausführung des Tests wird diese Datei vom XML-Parser eingefügt. Auf diese Weise kann man einfach die selben generelle Einstellungen für alle Test-Schritte verwenden. In diesem Beispiel werden die Einstellungen für Protokoll, Host, Port und Name der Web-Applikation geteilt.
Für die automatische Erstellung wird Ant benutzt, dadurch wird ermöglicht, dass Tests zu Modulen strukturiert werden können. Diese können einzeln oder als Ganzes ausgeführt werden. Somit kann jeder WebTest in Isolation ausgeführt werden und Tests können in Gruppen eingeteilt werden.
Die Ausführung der einzelnen Tests ist momentan durch Nutzung der API von *HttpUnit* implementiert. Die Resultate der Tests werden entweder in normalem Text oder in XML dargestellt, was einem die Möglichkeit der späteren Transformation durch XSLT ermöglicht. Für die XSLT-Transformation stellt Canoo WebTest Standard XSLT-Stylesheets zur Verfügung, die einfach für die eigene Verwendung abgeändert werden können.

9.4. Beispiel

Das folgende einfache Beispiel zeigt, wie man eine Verbindung zu einer Web-Applikation unter der `URL http://www.myserver.com:8080/myApp/login` aufbaut.

```
<?xml version="1.0"?>
<!DOCTYPE project SYSTEM "WebTest.dtd"[
      <!ENTITY definition SYSTEM "definition.xml">
      <!ENTITY config     SYSTEM "config.xml">
]>

<project name="SimpleDemo" basedir="." default="main">

      &definition;

      <target name="main">
            <testSpec name="myTest">
                  &config;
                  <steps>
                        <invoke
                              stepid="get Login Page"
                              url="login" />
                        <verifytitle
                              stepid="we should see the login title"
                              text="Login Page" />
```

```
                </steps>
            </testSpec>
      </target>
</project>
```

Die Task-Definition wird voraussichtlich in mehreren Dateien verwendet und wird daher aus Gründen der Redundanzvermeidung als XML-Entity definiert und ausgelagert (`definition.xml`):

```
<taskdef file="${webtest.home}/webtestTaskdefs.properties">
      <classpath>
            <fileset dir="${webtest.home}" includes="**/lib/*.jar"/>
      </classpath>
</taskdef>
```

Dasselbe geschieht mit der Konfiguration (`config.xml`):

```
<config
      host="www.myserver.com"
      port="8080"
      protocol="http"
      basepath="myApp" />
```

9.5. Quellen

http://www.junit.org/news/extension/web/index.htm
http://webtest.canoo.com/manual/WebTestHome.html

10. StrutsTestCase for JUnit

10.1. Entwicklung

StrutsTestCase for JUnit ist ein Open-Source Projekt, das auf SourceForge gehostet und von Christopher Pickslay, Deryl Seale, Sean Pritchard und Ted Husted entwickelt wurde. Seit 30. November 2001 ist das Projekt auf junit.org vertreten und die aktuelle Version 2.1.3 wurde am 31. Oktober 2004 veröffentlicht und ist mit Struts 1.2 kompatibel. Das Tool wird unter der Apache Software License vertrieben.

10.2. Ziel

StrutsTestCase ist eine Erweiterung der Standard Junit TestCase Klasse, die Möglichkeiten bietet, Code zu testen, der auf dem Struts Framework basiert. Hierbei stellt StrutsTestCase sowohl einen Ansatz mit Mock Objekten als auch einen Cactus-Ansatz zur Verfügung, um das ActionServlet von Struts auszuführen und den Code zu testen, egal ob eine laufende Servlet Engine vorhanden ist oder nicht.

10.3. Funktionalität

Es gibt 2 populäre Ansätze, um serverseitige Klassen zu testen: Mock Objekte, die Klassen testen, indem sie den Server-Container simulieren, und In-Container Tests, die Klassen Testen, die im aktuellen Server-Container ausgeführt werden. Mit StrutsTestCase kann jede dieser beiden Methoden benutzt werden ohne maßgeblichen Einfluss auf den aktuellen Unit-Test Code. Da die Setup- und Validierungs-Methoden von StrutsTestCase genau dieselben für beide Ansätze sind, liegt der einzige Unterschied, welchen Ansatz man wählt, in Wirklichkeit nur darin, welche Basisklasse man verwendet.
StrutsTestCase liefert 2 BasisKlassen, die beide vom Standard JUnit TestCase abgeleitet sind: *MockStrutsTestCase* benutzt HttpServlet Mock Objekte, um eine Container-Umgebung zu simulieren, ohne eine laufende Servlet-Engine vorauszusetzen. *CactusStrutsTestCase* hingegen benutzt das Cactus Test-Framework, um Struts Klassen im aktuellen Server-Container zu testen.

10.4. Beispiel

Das folgende Beispiel soll die Wirkungsweise von StrutsTestCase verdeutlichen:

```
public class LoginAction extends Action {

      public ActionForward perform(ActionMapping mapping,
                                   ActionForm form,
                                   HttpServletRequest request,
                                   HttpServletResponse response)
      {

            String username = ((LoginForm) form).getUsername();
            String password = ((LoginForm) form).getPassword();

            ActionErrors errors = new ActionErrors();

            if ((!username.equals("deryl")) || (!password.equals("radar")))
```

```
                errors.add("password",new
                            ActionError("error.password.mismatch"));

            if (!errors.empty()) {
                saveErrors(request,errors);
                return mapping.findForward("login");
            }

            // store authentication info on the session
            HttpSession session = request.getSession();
            session.setAttribute("authentication", username);

            // Forward control to the specified success URI
            return mapping.findForward("success");
      }

}
```

Hier erhalten wir eine *ActionForm* Bean, die Login-Informationen enthalten soll. Zunächst versuchen wir, Benutzername und Passwort auszulesen, und überprüfen diese danach auf Gültigkeit. Wenn diese nicht gültig sind, generieren wir eine ActionError Nachricht und versuchen, zur Login-Seite zu kommen, um erneut einzuloggen. Wenn Benutzername und Passwort gültig sind, speichern wir Authentifizierungs-Informationen in der Sitzung und versuchen, zur nächsten Seite zu gelangen.

Bei diesem Beispiel gibt es einige Sachen, die wir testen können:

- funktioniert das LoginForm Bean korrekt? Wird es korrekt instanziert?
- wird im Fehlerfall ein Fehler korrekt gespeichert? Werden wir wieder zur Login-Seite weitergeleitet?
- wird im Erfolgsfall die korrekte Seite angezeigt? Werden keine Fehler angezeigt? Wird die Authentifizierungs-Information korrekt gespeichert?

Mit StrutsTestCase können all diese Bedingungen im JUnit Framework getestet werden. Also erzeugen wir einen Testfall, abgeleitet von *MockStrutsTestCase*, für ein erfolgreiches Login:

```
public class TestLoginAction extends MockStrutsTestCase {

      public void setUp() { super.setUp(); }

      public void tearDown() { super.tearDown(); }

      public TestLoginAction(String testName) { super(testName); }

      public void testSuccessfulLogin() {
            setRequestPathInfo("/login");
            addRequestParameter("username","deryl");
            addRequestParameter("password","radar");
            actionPerform();
            verifyForward("success");
            assertEquals("deryl",(String)
                        getSession().getAttribute("authentication"));
            verifyNoActionErrors();
      }

      public void testFailedLogin() {
            addRequestParameter("username","deryl");
            addRequestParameter("password","express");
            setRequestPathInfo("/login");
            actionPerform();
```

```
            verifyForward("login");
            verifyActionErrors(new String[] {"error.password.mismatch"});
            assertNull((String)
                        getSession().getAttribute("authentication"));
      }

}
```

Die Methode *testSuccessfulLogin* testet, ob bei Eingabe von richtigen Benutzerdaten die richtige Seite angezeigt wird, die Authentizitäts-Informationen richtig gespeichert werden und keine Fehler angezeigt werden.
Die Methode *testFailedLogin* testet hingegen, ob bei Eingabe falscher Benutzerdaten (falsches Passwort) wieder zur Login-Seite weitergeleitet wird, ein Fehler erzeugt wird und keine Authentizitäts-Informationen gespeichert wurden.

10.5. Quellen

http://www.junit.org/news/extension/web/index.htm
http://strutstestcase.sourceforge.net/

11. TestCaseTool

11.1. Entwicklung

„TestCaseTool“ wurde von Masayuki Otoshi entwickelt und auf JUnit.org am 6. Dezember 2003 vorgestellt. Die aktuelle Version 1.2.10 wurde am 27.Mai 2004 veröffentlicht.
Das Tool kann ist kostenlos und kann unter Einhaltung der Lizenzbedingungen weitergegeben und verändert werden, es basiert auf Java.

11.2. Ziel

Bei den Standardtools, die zum Testen von Webanwendungen verwendet werden, ergibt sich, besonders bei sich ändernden Spezifaktionen, ein erheblicher Programmieraufwand um die Tests auf dem aktuellen Stand zu halten. „TestCaseTool“ arbeitet mit einem einfach zu erstellenden Test Case Dokument als Eingabe und versucht den Zusatzaufwand dadurch zu verhindern.

11.3. Funktionalität

Das Test Case Dokument lässt sich ohne Programmierkenntnisse erstellen und um zusätzliche Testfälle erweitern, zudem ist das Tool gratis. Somit kann „TestCaseTool“ in allen Entwicklungsstufen und von allen Stakeholdern eingesetzt werden.
Idealerweise werden Test Case Dokumente an den Benutzer mitgeliefert. Dieser kann die Funktionalität nun problemlos prüfen und bei etwaigen Fehlern oder fehlenden Funktionen eigene Test Cases hinzufügen. Auch von den Personen, die das Produkt warten, kann der Entwickler auf diese Art und Weise ein kurzes und prägnantes Feedback erhalten.

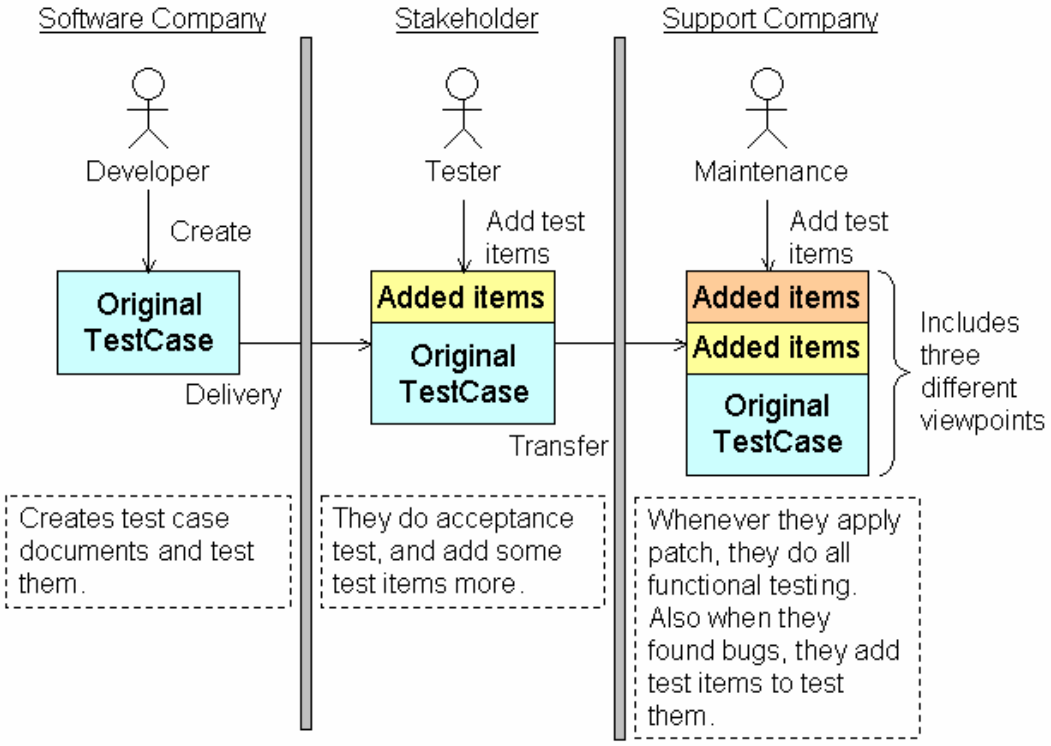


Abbildung 10: Einsatzgebiet

11.4. Beispiel

Die Eingabe für „TestCaseTool“ befindet sich im CSV Format und lässt sich mit z.B. mit Excel bearbeiten. Die Erstellung eines Testdokuments mit dem „TestCaseMaker“ wird im Folgenden anhand eines kleinen Beispiels erklärt.

Test Case:

1. Go to Google site and search 'Yahoo UK'.
2. Click the link named 'Yahoo! UK & Ireland' on the search result page.
3. Check to see if home page title of Yahoo UK is 'Yahoo! UK & Ireland'
4. Check to see if there is the link to News page.

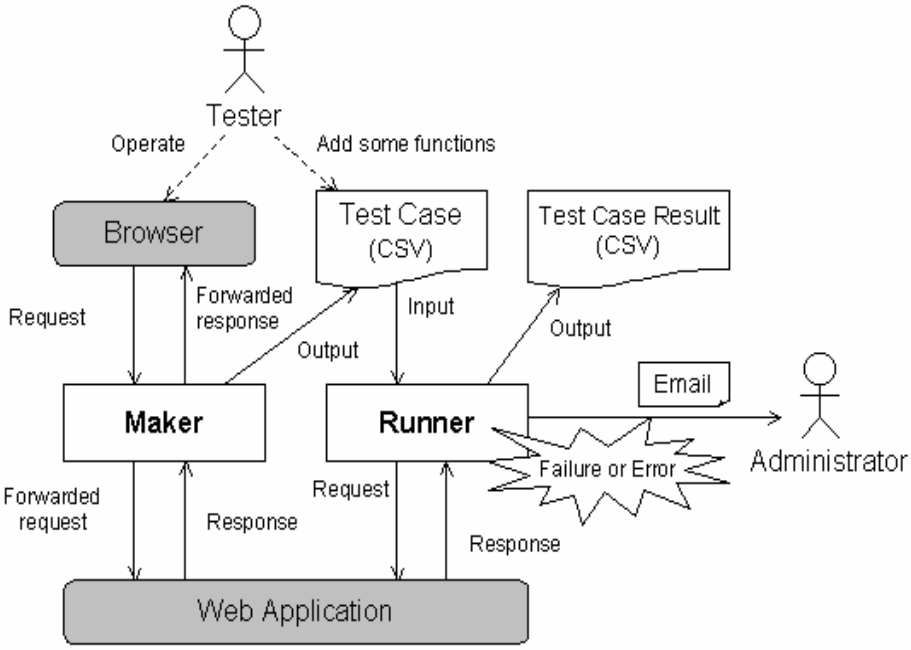


Abbildung 11: Anwendungsfälle

Der “TestCaseMaker” wird zwischen Browser und Web Applikation geschalten. Er protokolliert die ausgeführten Befehle und fügt sie in ein Test Dokument im CSV Format ein.

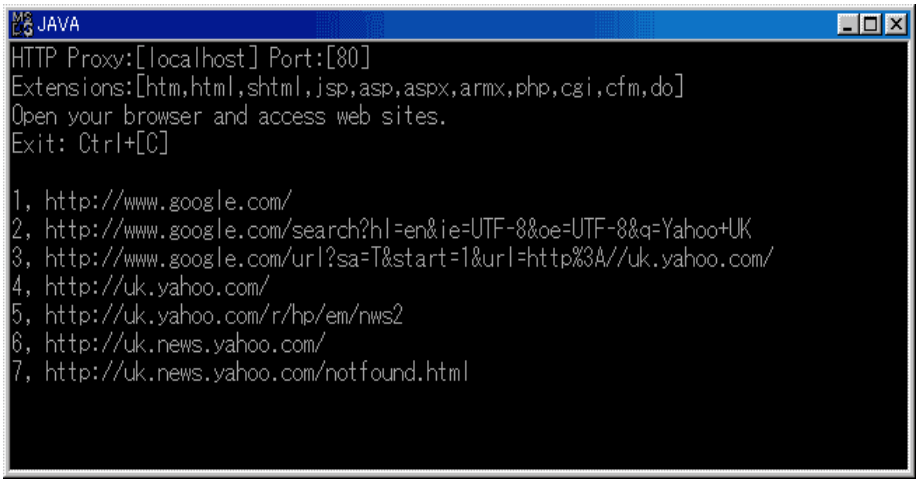


Abbildung 12: Aufzeichnung der besuchten Seiten

Um den anfangs beschriebenen Test durchzuführen muss die Ausgabe vom „TestCaseMaker“ allerdings noch erweitert werden (Der Maker protokolliert nur einfache Html-Befehle wie POST und GET). Abbildung 13 zeigt ein Dokument, welches alle Testfälle enthält.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	TESTID	METHOD	URL	POST	USERAGENT	RESULT	INFO	TIME	DATE	COMMENT	FUNCTION					
2	0									Initialize variable	Func SetGlobal Variable	baseUrl	http://www.google.com			
3	1	GET	${baseUrl}		Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)					Go to Google and search 'Yahoo UK'	Func Submit Form	f	btnG	${baseUrl}	q	Yahoo UK
4	2									Click the link which link name is Yahoo! UK & Ireland.	Func ClickLink	Yahoo! UK & Ireland				
5	3									Check to see if the title is Yahoo! UK & Ireland.	Func Exists TitleTag	Yahoo! UK & Ireland				
6	4									Check to see if there is a link to News.	Func Exists Tag	a	href	hp/em/nws2	![text]	News

Abbildung 13: Eingabe und Ausgabe in einem Dokument

Mit dem „Runner“ kann der Test ausgeführt werden, die Ergebnisse werden direkt in die Tabelle eingetragen. Der „TestCaseRunner“ unterstützt durch seinen Scheduler und der Email Benachrichtigung eine automatische Ausführung der Tests.

11.5. Quellen

http://www.junit.org/news/extension/web/index.htm
http://www5f.biglobe.ne.jp/~webtest/testcasetool/index.html

12. MaxQ

12.1. Entwicklung

MaxQ wird von der Open Source Community Tigris.org entwickelt, derzeit befindet sich das Tool in der Version 0.98, welche am 25. Oktober 2004 veröffentlicht wurde. Es ist frei erhältlich und Entwickler, die sich am Projekt beteiligen, werden gesucht.

12.2. Ziel

Mit MaxQ soll ein Tool bereitgestellt werden, das die einfache Erstellung von Testfällen ermöglicht. Die Funktionalität soll sich durch Einbindung von SQL oder JTidy leicht erweitern lassen.

12.3. Funktionalität

MaxQ ist ein Testing Tool, das einen HTTP Proxy benutzt um ein Test Script aufzunehmen. Der Proxy wird zwischen Web Browser und Web Server geschalten und kann dadurch die Eingaben des Benutzers aufzeichnen. Neben dem Verlauf der besuchten Seiten werden auch Einträge in Formulare unter zu Hilfenahme von Variablen aufgezeichnet.

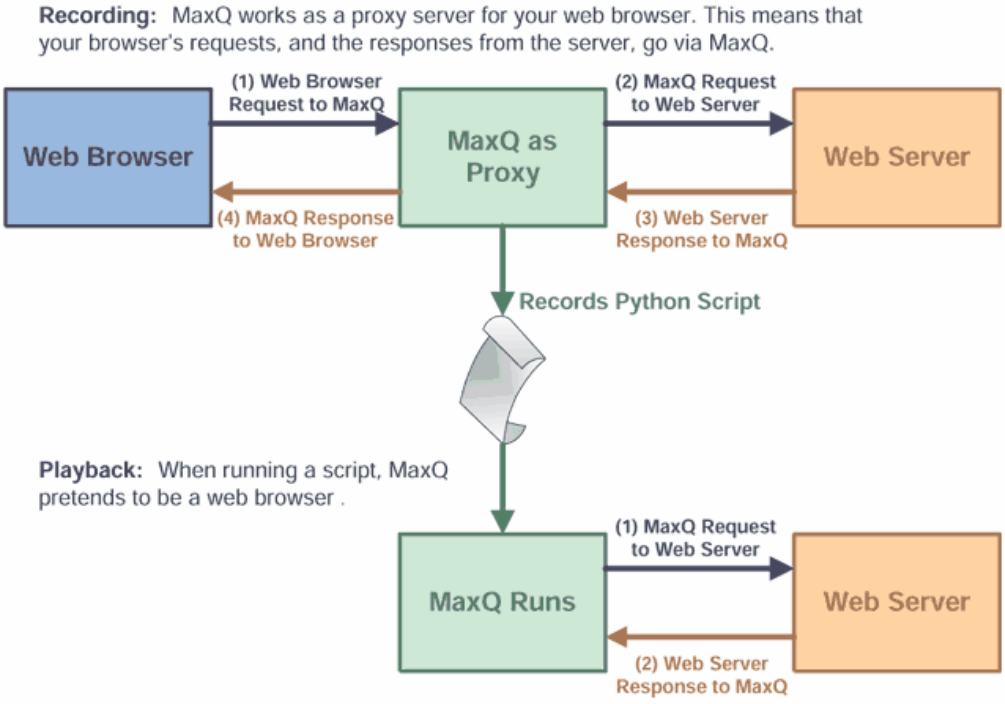


Abbildung 14: Funktionsweise con MaxQ

MaxQ zeichnet Jython (eine Implementierung der Sprache Python für Java Programme) Scripts auf. Folgendes Script würde zum Beispiel bei einem Besuch von www.google.com und anschließender Suche von „testing+tools“ entstehen.

```
1: # imports
2: from com.bitmechanic.maxq import HttpTestCase, EditorPane
3: from junit.textui import TestRunner
```

```
 4: from java.util import *
 5:
 6: # defintition of test class
 7: class MaxQTest(HttpTestCase):
 8:     def __init__(self):
 9:         HttpTestCase.__init__(self, "")
10:
11:     def runTest(self):
12:         self.get("http://www.google.com/")
13:         self.assertEquals(200, self.getResponse().getStatusCode())
14:
15:         self.get("http://www.google.com/images/logo.gif")
16:         self.assertEquals(304, self.getResponse().getStatusCode())
17:
18:
19:         list = [
20:             ("hl", "en"),
21:             ("q", "testing+tools"),]
22:         self.get("http://www.google.com/search", list)
23:         self.assertEquals(200, self.getResponse().getStatusCode())
24:
25: ########################################
26:
27: # Code to load and run the test
28: test = MaxQTest()
29: test.runTest()
```

Nach der Aufzeichnung können die Scripts über die Befehlszeile gestartet werden. Neben Funktionstests von Webseiten kann MaxQ mit Hilfe JTidy auch die Validierung von HTML Code übernehmen.

12.4. Quellen

http://maxq.tigris.org/
http://www.junit.org/news/extension/web/index.htm