Intrusive unit testing for Web applications

Philippe Poulard

April 20-24, WWW 2009 MADRID!SPAIN Madrid, Spain

Intrusive unit testing for Web applications

INRIA SOPHIA ANTIPOLIS

Philippe.Poulard@sophia.inria.fr

Testing a Web application:

User operations:

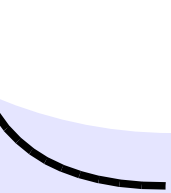
- Get the welcome page
- Click on a link
- Fill a form
- Check the pages
- etc

Features expected:

- Support of Javascript/AJAX
- Multiples windows/frames
- Forms, prompts, alerts
- etc

Operations to perform by a robot:

- A robot can replay the tests
- A robot won't complain
- A robot doesn't make mistakes (except those of the test designer)
- So far, a robot can't check what really happened server-side, except by checking the response sent to the client



WUnit can !
(yes we can !)

- ✔ XML/XPath
  - ➔ Require basic XSLT knowledge
  - ➔ Lots of XML libraries already available
- ✘ Not designed for low-level tests
  (does my Javascript/CSS hack works in IE and Firefox ?)
  - ➔ Use Selenium instead
- ✔ Tests-driven development
  - ➔ Write your tests BEFORE coding the Web app
- ✔ **Intrusive**: can act on server-side components
  (user session, request, response, Web application, servlet)
  - ➔ AFAIK, no other tool can do that
  - ➔ Examine what happens in the server
  - ➔ Update server components, GET the page that render them
    (REAL UNIT TEST)

How does it work ?

Non-intrusive mode

Start a conversation with a real server (Apache, Tomcat, PHP, etc):

```
<wunit:conversation>
    <!--your tests here-->
</wunit:conversation>
```

Wunit gives you the control of the Web client

Intrusive mode

Start a conversation with an emulator (servlets only):

```
<wunit:conversation
    application="file:///path/to/webapps/hello/WEB-INF/web.xml"
    uri="http://www.example.com/">
        <!--your tests here-->
</wunit:conversation>
```

WUnit gives you the control of:

- the Web client
- the Web server (for the host specified in `@uri`)

- GET a page:

```
<wunit:GET url="http://www.example.com/index.html?who=John Doe"/>
```

- Click a link:

```
<wunit:click target="{$wunit:document//A[@href][2]}"/>
```

- Fill a form:

```
<wunit:fill-form form="{$wunit:document//FORM[@name='login']}">
    <xcl:param name="user" value="Bill"/>
    <xcl:param name="password" value="Sesame"/>
</wunit:fill-form>
```

- POST datas:

```
<wunit:POST url="http://www.example.com/login.html">
    <xcl:param name="user" value="Bill"/>
    <xcl:param name="password" value="Sesame"/>
</wunit:POST>
```

- Check HTTP headers (and cookies):

```
<xunit:assert-number-equals expected="200"
        result="{value($wunit:frame/@wunit:response-code)}"/>
<xunit:assert-string-equals expected="text/html"
        result="{string($wunit:frame/@wunit:mime-type)}"/>
```

Modules in Active Tags and RefleX

- **WUnit** is just a module (a library) from a larger framework called **Active Tags** that allows to design XML-based applications.
- **RefleX** is an implementation of **Active Tags** in **Java**.

Each module can define active tags `<my:elem>`, XPath functions `my:funct()`, XPath variables `$my:var`, foreign attributes `@my:attr`, data types `#my:data-types`

Tag libraries available (from Active Tags and RefleX):

- WUnit: `<wunit:GET>`, `<wunit:click>`, `$wunit:document`, etc
- XUnit: `<xunit:test-case>`, `<xunit:assert-node-equals>`, etc
- XCL (the XML Control Language): `<xcl:if>`, `<xcl:then>`, `<xcl:else>`, `<xcl:parse>`, `<xcl:transform>` (XSLT), etc
- I/O: `io:file()`, `#io:input`, etc
- Web: `<web:mapping>`, `$web:session`, `#web:x-session`, `web:mime-type()`
- etc (SQL, XQuery...)

Hundreds of active materials are available

Screenshots of our Web application: an image gallery

AJAX autocompleter

Test of the Welcome page


```
<!--set the boundary of the test case-->
<xunit:test-case name="report/1-gallery-welcome" label="[Web] Welcome page">
    <!--start to discuss with the servlet emulator-->
    <wunit:conversation application="../webapp/WEB-INF/web.xml"
                        uri="http://www.example.com/">
        <!--get the welcome page hosted in our server emulator-->
        <wunit:GET url="http://www.example.com/"/>
        <!--do we have it ?
              we ask to the $wunit:frame predefined property
              what is the HTTP response code-->
        <xunit:assert-number-equals
            result="{ value( $wunit:frame/@wunit:response-code ) }"
            expected="200"/>
        <!--is it an HTML page ?-->
        <xunit:assert-string-equals
            result="{ string( $wunit:frame/@wunit:mime-type ) }"
            expected="text/html"/>
        <!--parse the static welcome page from the file system...-->
        <xcl:parse-html source="../webapp/index.html" name="expected"/>
        <!--...and compare it with those returned by the server.
              $wunit:document is a predefined property that refers to
              the DOM of the HTML document of the current page-->
        <xunit:assert-node-equals
            result="{ $wunit:document }"
            expected="{ $expected }"/>
    </wunit:conversation>
</xunit:test-case>
```

Test of the gallery (1)

But, it is empty !?!

Yes, but let's store authoritatively some images in the user session (because this application works like this), and check how is rendered the page

```
<wunit:conversation application="../webapp/WEB-INF/web.xml"
                    uri="http://www.example.com/">
    <!--store directly some objects in the current session server-side
        (objects are stored as attributes) ;
        the objects stored are images from the file system-->
    <xcl:attribute referent="{ $wunit:session }"
                   name="item1"
                   value="{ io:file( '../img/xml-10.png' ) }"/>
    <xcl:attribute referent="{ $wunit:session }"
                   name="item2"
                   value="{ io:file( '../img/xml2006-logo.png' ) }"/>
    <!--get the gallery page-->
    <wunit:GET url="http://www.example.com/gallery.html"/>
    <!--check the result:
                    .../...
    -->
```

Test of the gallery (2)


```
    <!--
                    .../...
    -->
    <!--check what we GET:-->
    <xunit:assert-number-equals
        result="{ value( $wunit:frame/@wunit:response-code ) }"
        expected="200"/>
    <xunit:assert-string-equals
        result="{ string( $wunit:frame/@wunit:mime-type ) }"
        expected="text/html"/>
    <!--we should have 2 rows-->
    <xunit:assert-number-equals
        result="{ count( $wunit:document//TR ) }"
        expected="2"/>
    <!--first row-->
    <xunit:assert-string-equals
        result="{ $wunit:document//TR[1]/TD[1] }"
        expected="item1"/>
    <xunit:assert-string-equals
        result="{ $wunit:document//TR[1]/TD[2]/IMG/@src }"
        expected="images/item1"/>
    <!--second row-->
    <xunit:assert-string-equals
        result="{ $wunit:document//TR[2]/TD[1] }"
        expected="item2"/>
    <xunit:assert-string-equals
        result="{ $wunit:document//TR[2]/TD[2]/IMG/@src }"
        expected="images/item2"/>
</wunit:conversation>
```

Upload a file and test if it's in the Web server

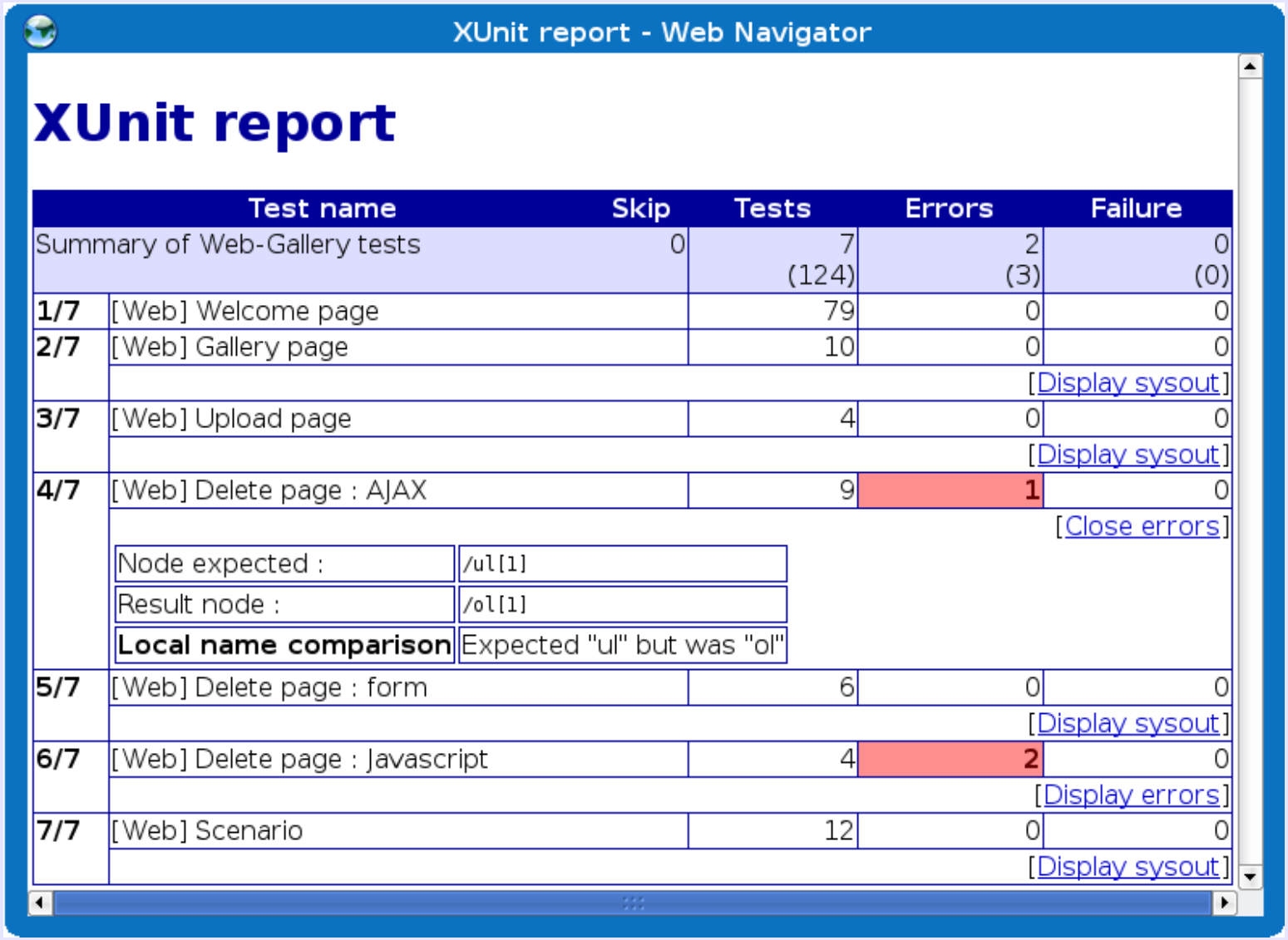
```
<!--fill the HTML form and POST it to the server-->
<wunit:fill-form form="{ $wunit:document//FORM[@name='upload'] }">
    <!--the text input is a string-->
    <xcl:param name="imageName" value="test1"/>
    <!--the file to upload, just give it a file-->
    <xcl:param name="imageFile" value="{ io:file( '../img/xml-10.png' ) }"/>
</wunit:fill-form>
<!--check the message in the
    response page sent
    by the server-->
<xunit:assert-string-equals
    result="{ normalize-space( $wunit:document//BODY/DIV[1] ) }"
    expected="xml-10.png uploaded."/>
<!--check that the session object was created ;
    it contains a reference to a file stored in the server ;
    we just check the name of the file-->
<xunit:assert-string-equals
    result="{ name( value( $wunit:session/@test1 ) ) }"
    expected="xml-10.png"/>
```

xml-10.png uploaded.

Other tests to consider:

- The AJAX autocompleter (please refer to the RefleX web site)
- A complete scenario (please refer to the RefleX web site)
- Checking database updates: this could be perform with a direct SQL or XQuery query to the database.

Tools for running all the test cases, aggregating the results in a single XML report, and an XSLT styleshet are supplied



Feature not (yet) covered

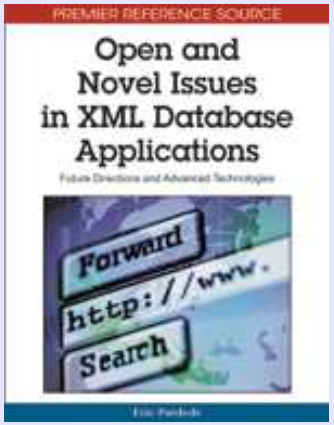
- SSL and certificates:
  Could use a dummy certificate server-side

- JNDI support:
  Would allow to have a testing environment that includes test databases (SQL, XQuery, etc) that could be initialized by the test suite
  Right now: use the production database or duplicate the Web application

- WUnit/XUnit as an output language for Selenium ?

- A better error report (with HTML rendered fragments in the output)

About the engine



- Chap 8: "Native XML Programming: Make Your Tags Active" in "Open and Novel Issues in XML Database Applications: Future Directions and Advanced Technologies" (April 2009)

ISBN: 978-1-60566-308-1
http://www.igi-global.com/reference/details.asp?ID=33277

- Properties of schema mashups: dynamicity, semantic, mixins, hyperschemas

Balisage 2008 The Markup Conference

http://www.balisage.net/Proceedings/html/2008/Poulard01/Balisage2008-Poulard01.html
http://hal.inria.fr/docs/00/32/26/61/ANNEX/Bal2008poul061003.pdf

- Active Tags: Mastering XML with XML

2007 Extreme Markup Languages® a registered trademark of IDEAlliance

http://www.idealliance.org/papers/extreme/proceedings/html/2007/Poulard01/EML2007Poulard01.html
http://hal.inria.fr/docs/00/17/37/16/ANNEX/eml2007-active-tags.pdf

- Active Tags: an XML system for Native XML Programming

<XML2006>

http://2006.xmlconference.org/programme/presentations/156.html

Free, open source

RefleX

The Active Tags engine, in Java

- 110,000 lines of code (stripped from comments and blank lines)
- Jar size: 1.3MB

- Have the RefleX !
  http://reflex.gforge.inria.fr
- XUnit:
  http://reflex.gforge.inria.fr/xunit.html
- WUnit:
  http://reflex.gforge.inria.fr/wunit.html
  http://reflex.gforge.inria.fr/wunit-quick-start.html
- Active Tags:
  http://ns.inria.org/active-tags/
  http://ns.inria.org/active-tags/references/references.html

Questions ?