Property-based testing for Web Services

Simon Thompson
University of Kent

Thomas Arts
Quviq
Introduction
EU PROWESS project

Aims to improve testing, particularly for web services, through uptake and use of property-based testing (PBT).

The QuickCheck tool for PBT can be used to test web services as well as systems built in Erlang, Java, C, …

… but system models and properties are written in Erlang.
University of Sheffield  UK
University of Kent  UK
Chalmers University of Technology  Sweden
Universidad Politécnica de Madrid  Spain
University of A Coruña  Spain
Quviq AB  Sweden
Erlang Solutions Ltd  UK
Interoud Innovation S.L.  Spain
SP Technical Research Institute of Sweden  Sweden
Erlang ecosystem

- QuickCheck
- Megaload
- Wrangler
Overview
test results

implementation
tests

test results
QuickCheck

tests

implementation

test results
test results

properties

QuickCheck

GoodExamples

tests

implementation

test results
specification

WStoolkit

properties

QuickCheck

tests

implementation

test results
QuickCheck

properties

tests

implementation

test results
properties

QuickCheck

FaultCheck

tests
implementation

test results
properties

QuickCheck

tests
implementation

Mu2

test results
properties

QuickCheck

tests

implementation

Smother

test results
properties

QuickCheck

tests
implementation

ComplexityCheck
test results
QuickCheck
QuickCheck

properties

tests

implementation

test results
properties
  QuickCheck
  tests
  implementation
  test results
  Ranker
Case study
VoDKATV

Internet-Protocol TV (IPTV) / “Over the top” content (OTT) Cloud Middleware Architecture.

Interactive services for IPTV/OTT environments, eg, hotels.

Runs on a set-top-box (STB), connected to a TV + remote.

Component-based; on client side: STB, tablet, PC, phone, …
Set-top box

The STB includes

- a portable middleware layer implemented in Erlang,
- a UI layer developed in HTML, JavaScript and CSS (Webkit browser);
- communication between the UI layer and the middleware via a WebSocket-based protocol.
Web services for interactions

Some APIs respond in XML, others in JSON

Different kinds of authentication for access to the APIs:
  ● none required,
  ● authentication with cookies
  ● authentication with tokens, e.g. expiration time, max # logins per user, …
Property-based testing for VoDKATV

This is where the demo by Thomas fits …
The toolset
Evolution in PBT with WStoolkit

Using Wrangler, Kent’s tool for refactoring Erlang systems.

Infer of changes between WSDL descriptions …

… from these generate refactoring scripts …

… which automate model evolution as much as possible.
test results

properties

specification

Megaload

QuickCheck

tests

implementation

test results
Megaload – Load testing VoDKA

Cloud-based load testing of systems.

Megaload: loads, monitors and presents results.

Generating load profiles …

… and shrinking to minimal (counter-) examples in the most load-effective way.
Inference and PBT

How to develop properties for a system. Two tools:

- *James* – infer models for web services from unit tests written in Java, using JUnit.

- *Synapse* - infer FSM from systems, and visualise the difference between models / systems.
New JUnit tests from existing tests, by model inference.

Track a combination of data- / control-flow information … … extracted from running the test suite on the SUT … run the tests on the Java VM … track information using C++ agent and JVM-TI API
Track and send to an Erlang server:
● the execution order of the calls in the JUnit tests, and
● how objects are reused.

Server generates a model … visualised through GraphViz.

Translate model into QuickCheck … then generate new tests, that can be added to the original test suite.
Synapse

An Erlang interface to grammar inference tools.

Synapse interfaces to the StateChum tool for passive and active inference of FSM models, as well as:

- active and passive learning,
- model differencing, and
- FSM and difference visualisation.
Understanding properties and models

*ReadSpec* to render QuickCheck models in (semi-)natural language.

*Synapse* tool allows users to visualise differences between variants of models / systems as FSMs.

*GoodExamples* tool to make the meaning of a property more concrete by viewing it as a set of unit tests.
properties

QuickCheck

ReadSpec

tests

implementation

test results
ReadSpec

ReadSpec uses QuickCheck to automatically generate semi-natural language descriptions of QuickCheck properties and QuickCheck state machine models.

Example: `simple_eqc.erl` contains a property to test the delete operation of the lists module:
?FORALL({I,L}, {int(), list(int())},
    not lists:member(I, lists:delete(I, L)))

**FEATURE**: Simple QuickCheck properties
**SCENARIO**: Deleting an integer from a list should result in a list that does not contain that integer.
**GIVEN** I have the integer 19
**AND** I have the list [7, -24, -18, 17, -8, -9, -8]
**THEN** lists:member(19, lists:delete(19, [7, -24, -18, 17, -8, -9, -8]))
**IS FALSE.**
Good Examples tool

It can be hard to tell what a property tests…

properties - powerful and general;
unit tests - easy to understand but specific.

Good Examples - makes the meaning of a property more concrete by viewing it as a set of unit tests.
Scenario 1: From a test suite, which of our test cases the property captures?
Solution: Our technique can say with high probability whether a property captures a given test case.

Scenario 2: what does a property test?
Solution: Our technique to generates representative examples of what a QuickCheck property tests.
Support for Web Services

Tools to support data generation for web services models:

JSONgen is a library for generating QuickCheck generators from descriptions of JSON data using JSON schemas, and for automatically exploring and testing JSON web services.

wsdl_dsl is a QuickCheck library that implements a domain specific language which re-uses the WSDL syntax to allow users to express WSDL types as QuickCheck generators.
specification

existing tests

WStoolkit

properties

QuickCheck

tests

implementation

test results
JSONgen – convert and explore

Convert JSON schema to mochijson2 Erlang term.
Convert JSON schema into a QuickCheck generator.
Convert JSON data value in mochijson2 format to text
Explore and test a JSON based web service using the web links / data types embedded in the JSON schema args.
Can tailor the actions with a QuickCheck state machine.
QuickCheck

tests

properties

implementation

---

test results
Scaling PBT

Model using *components* instead of a single model.

Library for *mocking* the behaviour of callout components.

*Clustered* system resulting from the component models.
MoreBugs

QuickCheck “by hand”: run QC, fix bug, repeat …

With MoreBugs, can find “all” bugs at once, through

- find bug,
- generalise
- modify generator to avoid it

and repeat …
Graphical editing
properties

QuickCheck

tests
implementation

Smother

test results
Validating quality of test suites

*Smother* used to assess the MC/DC coverage of a test suite.

*Mu2* supports mutation testing.

<table>
<thead>
<tr>
<th>Condition</th>
<th>Matched</th>
<th>Non-Matched</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A = 0$</td>
<td>1 times</td>
<td>2 times</td>
</tr>
<tr>
<td>$B &gt; 4$</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

When non-matched: 75.0% sub-component coverage

matched non-matched

$\text{dv}(A, B) \Rightarrow$

\[
\begin{align*}
\text{if } (A == 0) \text{ and } (B > 4) \rightarrow \\
B / 1;
\end{align*}
\]

\[
\text{end.}
\]

$(A == 0) \text{ and } (B > 4)$

- Matched: 1 times
- Non-Matched: 2 times
properties

QuickCheck

FaultCheck
tests
implementation

test results
Testing non-functional requirements

FaultCheck …

… a fault-injection tool for C code that combines fault-injection and property based testing using QuickCheck.
WebDriver
Continuous integration
-true(test).
#include_lib("eunit/include/eunit.hrl").
-compile_export_all.
-endif.

-record(mstore, [name, file, offset, size, index=gb_trees:empty(), next-@]).
-record(mset, [size, files-[], dir, metrics=gb_sets:new()]).

-define(OPTS, [raw, binary]).
-export([put/4, get/4, new/2, delete/1, close/1, open/1, metrics/1,
  fold/3]).

%% @doc Opens an existing mstore.

delete(MSet = #mset(dir=Dir)) ->
  close(MSet),
  {ok, Files} = file:list_dir(Dir),
  Files1 = [[Dir, 47 | File] || File <- Files],
  [file:delete(F) || F <- Files1],
  file:del_dir(Dir).

-spec open( Dir :: string() ) -> {ok, #mset[]} | {error, not_found}.
open(Dir) ->
  case open_mstore([Dir | "/mstore"] ) of
    {ok, FileSize, Metrics} ->
      {ok, #mset(size=FileSize, dir=Dir, metrics=Metrics)};
      {error, not_found}.
    end.

new(FileSize, Dir) when is_binary(Dir) ->
  quickcheck-ci.com/cover/build/408/mstore.erl.html
Comparing different implementations
Results
Scalable PBT: components and mocking
**Accessible PBT:** ReadSpec, GoodExamples
PBT for web services: WStoolkit, JSONgen

QuickCheck

tests

implementation

specification

existing tests

properties

test results
Discovering properties: James, Synapse

QuickCheck

tests  implementation

test results
Improved testing: Smother, Mu2, FaultCheck
Acknowledgement

The Universities of Sheffield, Kent, A Coruña, Chalmers Technical University and the Polytechnic University of Madrid; Quviq AB, Interoud, Erlang Solutions Ltd and SP gratefully acknowledge the support of the European Commission for the PROWESS project, funded under Framework Programme 7.
Results

**Scalable PBT**: components, mocking

**Accessible PBT**: ReadSpec, GoodExamples

**PBT for web services**: WStoolkit, JSONgen

**Discovering properties**: James, Synapse

**Improved testing**: Smother, Mu2, FaultCheck

**Evolution and PBT**: QC CI, WStoolkit, Ranker

www.prowess-project.eu