Automated Functional Testing based on the Navigation of Web Applications

Boni García - Juan Carlos Dueñas

Universidad Politécnica de Madrid (Spain)

Table of Content

1. Introduction

- Web has become in one of the most influential instrument in the history of mankind
- Software testing is the main technique to ensure quality and find bugs, but it is time-consuming
- Testing is often poorly performed or skipped by practitioners
- **Test automation** can help to avoid this situation

1.1. Research Statement

- Problem:
  - How to achieve automated functional testing in web applications?
- Proposal:
  - Automated evaluation of the correct navigation of web applications (assessment of the specified functional requirements) by using real browsers

Table of Content

2. Background (I)

- Automated Software Testing (AST): *"Application and implementation of software technology throughout the entire Software Testing Lifecycle (STL) with the goal to improve efficiencies and effectiveness"*
- AST Frameworks (http://www.automatedtestinginstitute.com/):
  - 1st generation: Record & Playback (R&P)
  - 2nd generation: Data-driven
  - 3rd generation: Model-based

2. Background (II)

- Web Modeling:

	Data	UI	Person alization	Transactional	Non-functional
UWE	✓	✓	✓		
W2000	✓	✓	✓		✓
WebML			✓	✓	
NDT	✓	✓	✓	✓	✓

- Graph Theory:
  - Graph = Set of vertices (nodes) connected by edges (links)
  - Digraph = Graph in which edges have orientation
  - Multidigraph = Digraph in which multiples edges and loops are allowed
  - Path = Sequence of vertices such from each node there is an edge to the next vertex

Table of Content

3. Approach

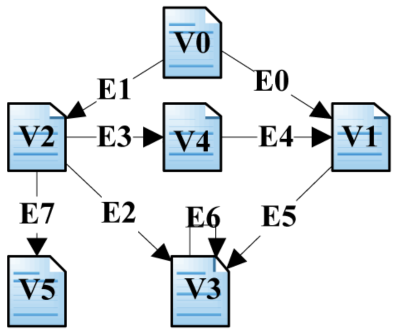
- High-Level Description:
  1. To use graph theory to represent and work with the navigation of web applications
  2. To model correct navigation with different models (*pre-automation*)
  3. To develop framework to perform automated test generation, execution and reporting
  4. To provide data-driven testing by compiling test data and oracles in tabular files, i.e. Excel spread-sheets (*post-automation*)

3.1. Finding Paths in a Digraph (I)

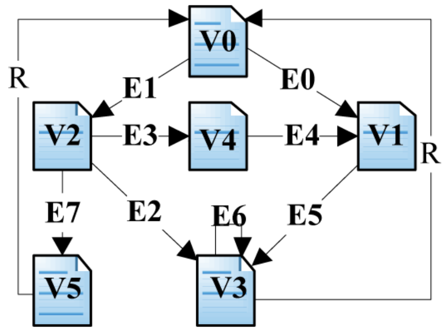
- Possible methods/algorithms:
  - Graph Traversal: Breadth-First Search (BFS) and Depth-First Search (DFS)
  - Traveling Salesman Problem (TSP)
  - The Shortest Path Problem (SPP): Dijkstra, Bellman-Ford, A*, and Floyd-Warshall
  - The Chinese Postman Problem (CPP)
  - The Node Reduction (NR) algorithm

3.1. Finding Paths in a Digraph (II)

- Alternatives: CPP or NR



i) Original



ii) Strongly connected

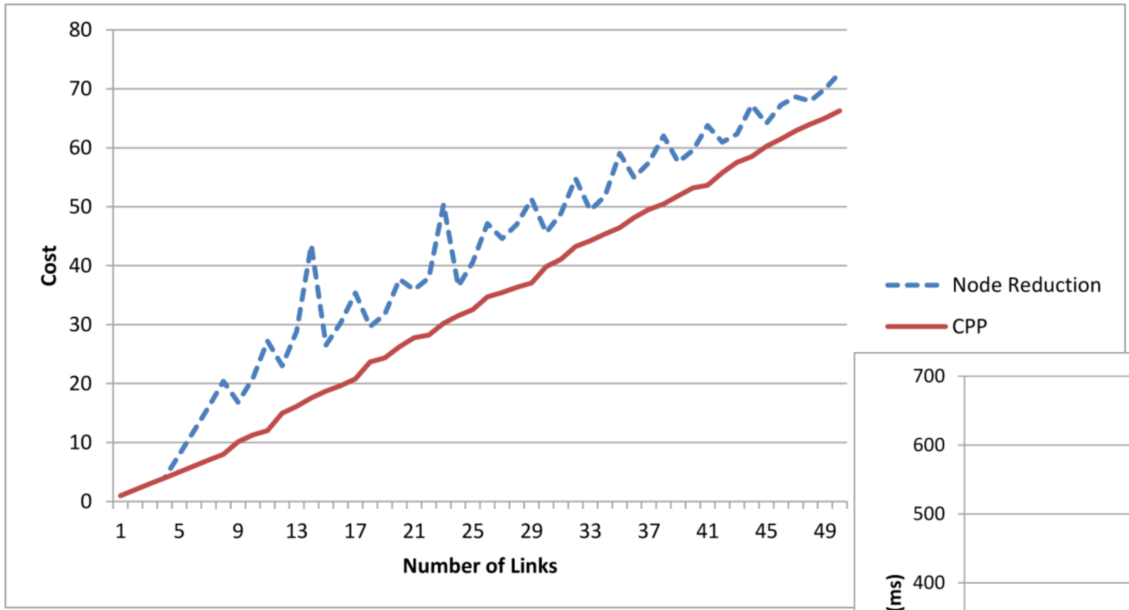
- CPP: E1·E3·E4·E5 + E1·E7 + E1·E2·E6 + E0
- NR: E1·E7 + E0·E5·E6 + E1·E3·E4·E5·E6 + E1·E2·E6

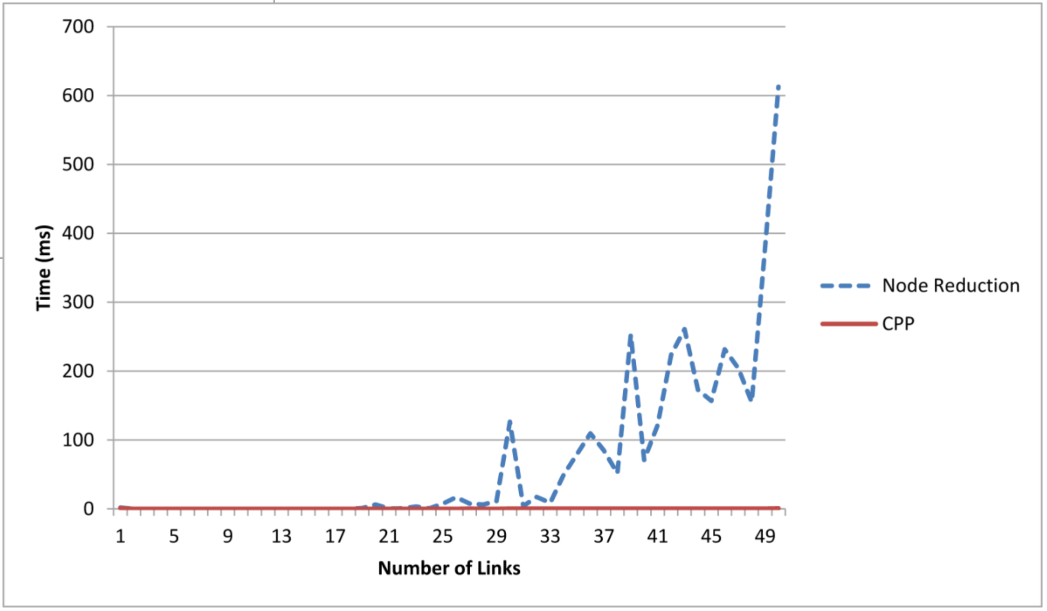
Is really better CPP than CPP?

3.1. Finding Paths in a Digraph (III)

- CPP vs. NR: Laboratory Experiment:
  - Comparison between these algorithms, using random multidigraphs with incremental number of links (from 1 to 50).
  - For each digraph NR and CPP will be executed, comparing its cost (number of links employed in the resulting set of paths), and the computation time (milliseconds to achieve the solution).
  - Repeated 100 times, and the mean of the values (cost and time) will be displayed by charts.

3.1. Finding Paths in a Digraph (IV)



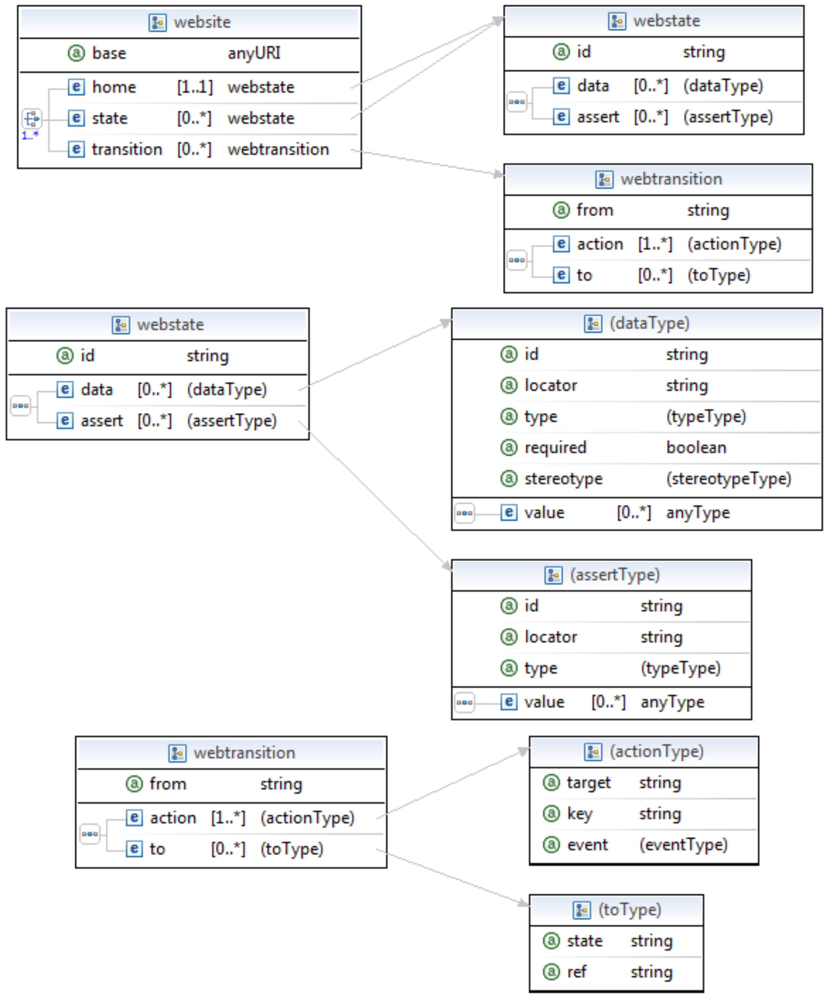


CPP has better behaviour in cost and time resolution!

3.2. Navigation Modelling (I)

- Modelling navigation by testers/developers (pre-automation):
  - UML: Based on NDT, created with Enterprise Architect (in XMI format):
    - Use cases
    - Activity diagrams
    - Presentation diagrams
  - R&P: HTML Scripts, recorded with Selenium IDE
  - XML: Self-defined XSD Schema

3.2. Navigation Modelling (II)



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<website xmlns="http://www.dit.upm.es/atp"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://www.dit.upm.es/atp
http://atestingp.sourceforge.net/atp.xsd"
base="http://localhost:8080/WebAdmin/">

   <home id="login">
      <data locator="username">
         <value>Administrador</value>
      </data>
      <data locator="password">
         <value>admin</value>
      </data>
   </home>

   <transition from="login">
      <action target="frmDatos_0" event="click" />

      <to state="init" />
      <to state="login" />
   </transition>

   <state id="init">
      <assert locator="texto-entrada" type="text">
        <value>Welcome</value>
      </assert>
   </state>

</website>
```

3.3. Data-Driven View

- Data-driven testing (post-automation): Test data and oracles in tabular files (Excel):

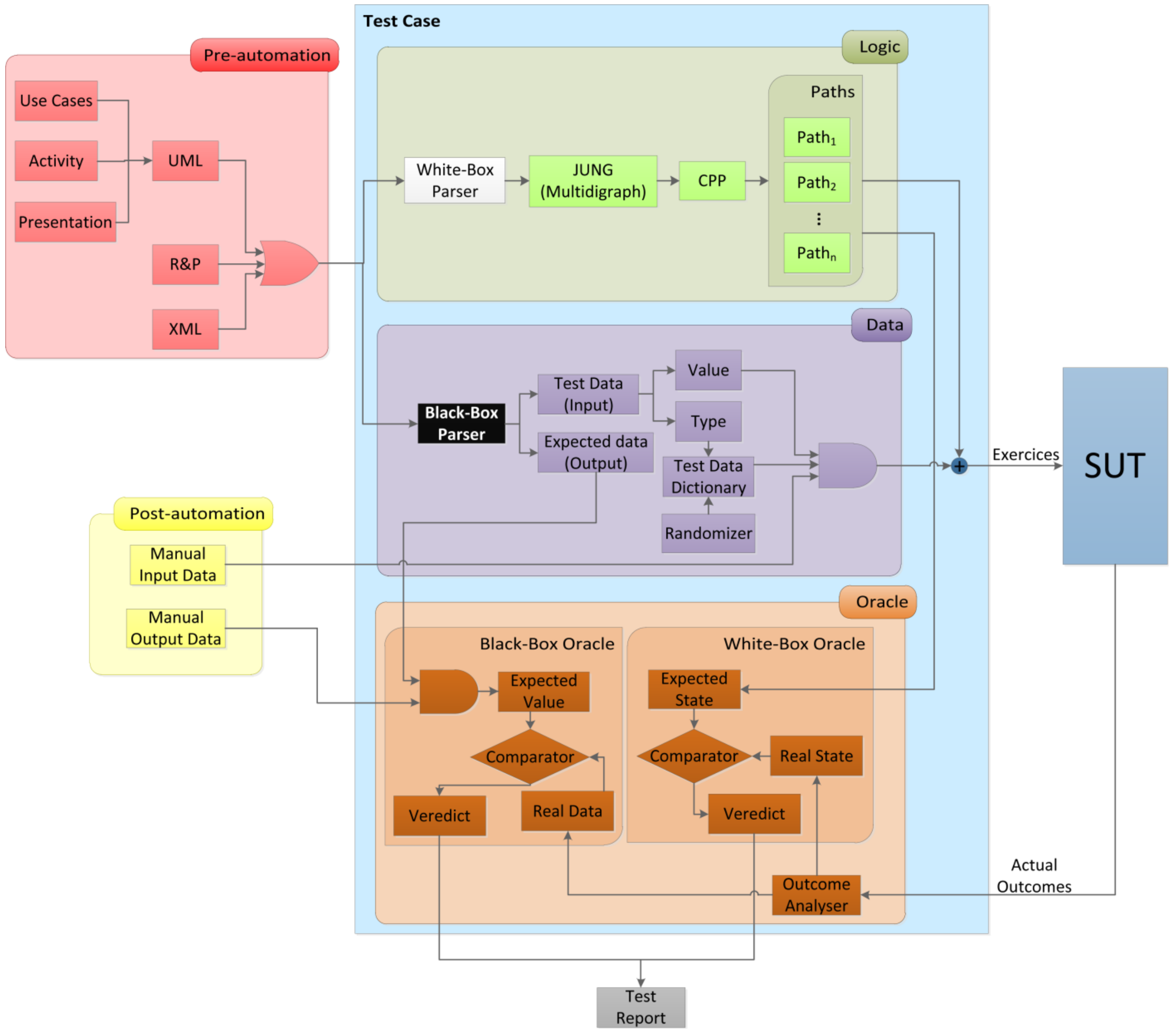
Test data (input)			Oracles (expected output)		
$in\_element_1$	…	$in\_element_n$	$out\_element_1$	…	$out\_element_m$
$data_{1\_1}$	…	$data_{1\_n}$	$outcome_{1\_1}$	…	$outcome_{1\_m}$
$data_{2\_1}$	…	$data_{2\_n}$	$outcome_{2\_1}$	…	$outcome_{2\_m}$
…	…	…	…	…	…

- The elements are located in the DOM document using looking for:
  - `id`, `name`, `value`, `text` attributes
  - XPath expression

3.4. Automated Verification

- While automated browsing is performed, the following assessment is done:
  - State verification:
    - The existence of each defined data field is ensured
    - Each test oracle is assessed
    - Each is state component in the DOM is checked (e.g. frames, images, …)
    - Each state is validated as the aggregation of the defined locators (data fields and oracles)
    - JavaScript notifications are captured
  - Transition verification:
    - Each transition locator is assessed and executed

3.5. Summary



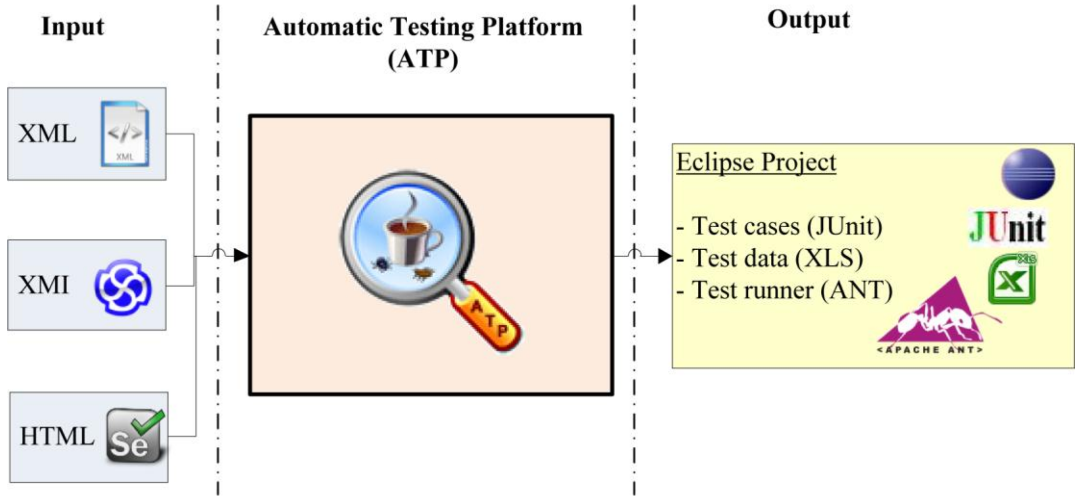
Pre-automation
Use Cases
Activity
Presentation
UML
R&P
XML
Test Case
Logic
White-Box Parser
JUNG (Multidigraph)
CPP
Paths
$Path_1$
$Path_2$
$Path_n$
Data
Black-Box Parser
Test Data (Input)
Expected data (Output)
Value
Type
Test Data Dictionary
Randomizer
Post-automation
Manual Input Data
Manual Output Data
Oracle
Black-Box Oracle
Expected Value
Comparator
Veredict
Real Data
White-Box Oracle
Expected State
Comparator
Real State
Veredict
Outcome Analyser
Exercices
SUT
Actual Outcomes
Test Report

Table of Content

4. Implementation: ATP

- The proposed approach has been implemented in a open-source testing framework: Automatic Testing Platform (ATP):

http://atestingp.sourceforge.net/



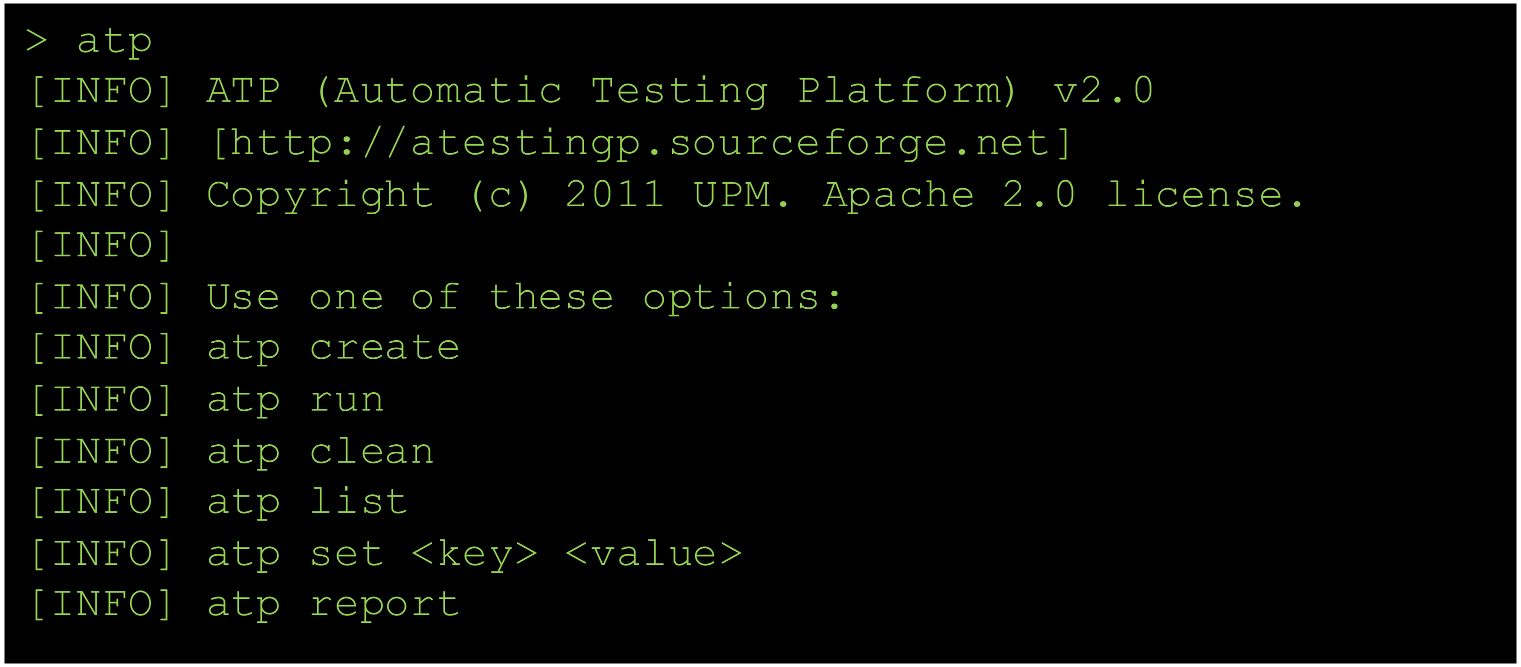
4.1. ATP Architecture

- Open-source components:

Function	Library	URL
Unit Framework	JUnit	http://www.junit.org/
Web browsing	Selenium	http://seleniumhq.org/
Test case generation	Freemarker	http://freemarker.sourceforge.net/
Data generation	dgMaster	http://dgmaster.sourceforge.net/
Test case execution	Ant	http://ant.apache.org/
Graph manipulation	JUNG	http://jung.sourceforge.net/
XML parsing	JDOM	http://www.jdom.org/
Spread-sheet access	JExcelAPI	http://jexcelapi.sourceforge.net/

4.2. ATP Commands

- ATP has been implemented as a command-line (shell) tool:



```
> atp
[INFO] ATP (Automatic Testing Platform) v2.0
[INFO] [http://atestingp.sourceforge.net]
[INFO] Copyright (c) 2011 UPM. Apache 2.0 license.
[INFO]
[INFO] Use one of these options:
[INFO] atp create
[INFO] atp run
[INFO] atp clean
[INFO] atp list
[INFO] atp set <key> <value>
[INFO] atp report
```

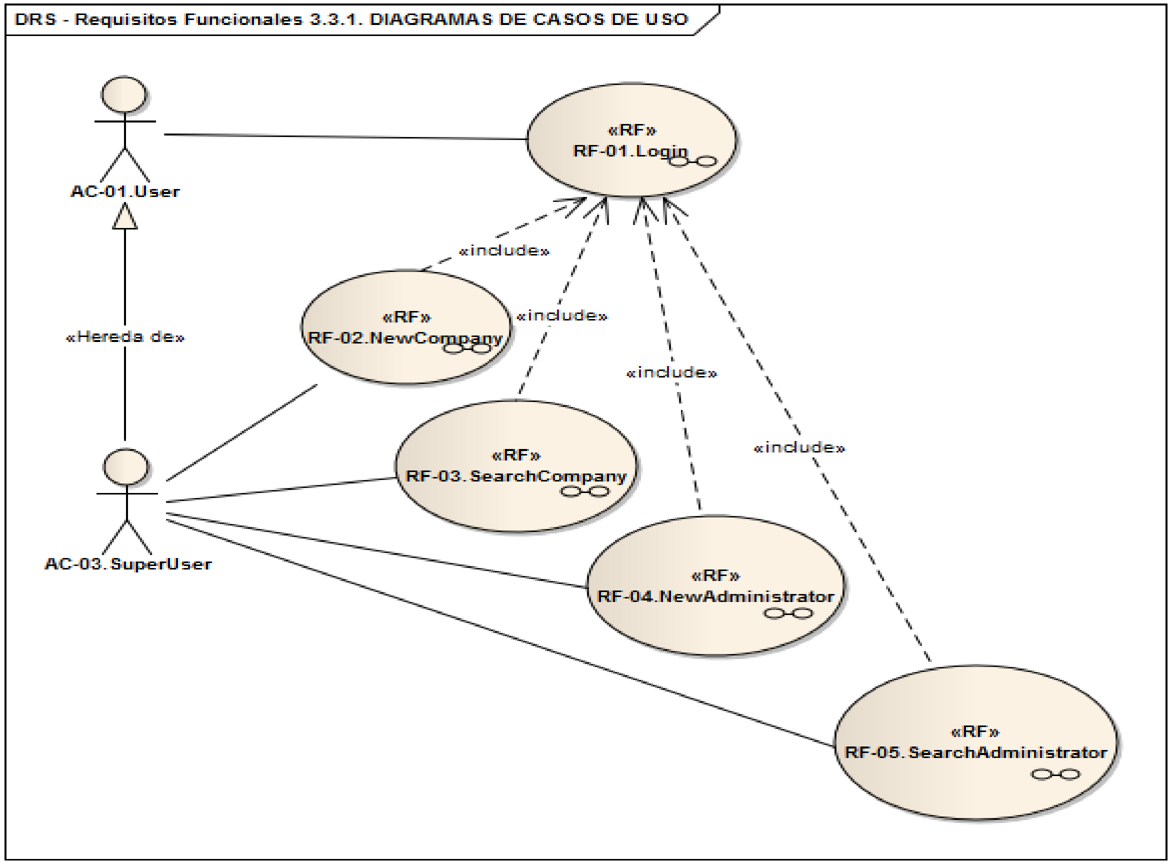
Table of Content

5. Case Study

- System Under Test: “Factur@”, which is an electronic invoice web management system which has been developed using a Model Driven Engineering (MDE) approach
- Research Questions (RQ):
  - RQ1: Does the Factur@ application accomplish its functional requirements?
  - RQ2: Is ATP capable of finding defects in a finished web application?
  - RQ3: What are the advantages and disadvantages of different types of input (UML, XML, and R&P) to ATP?

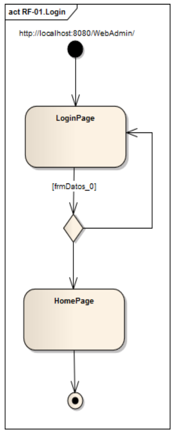
5.1. Pre-Automation (I)

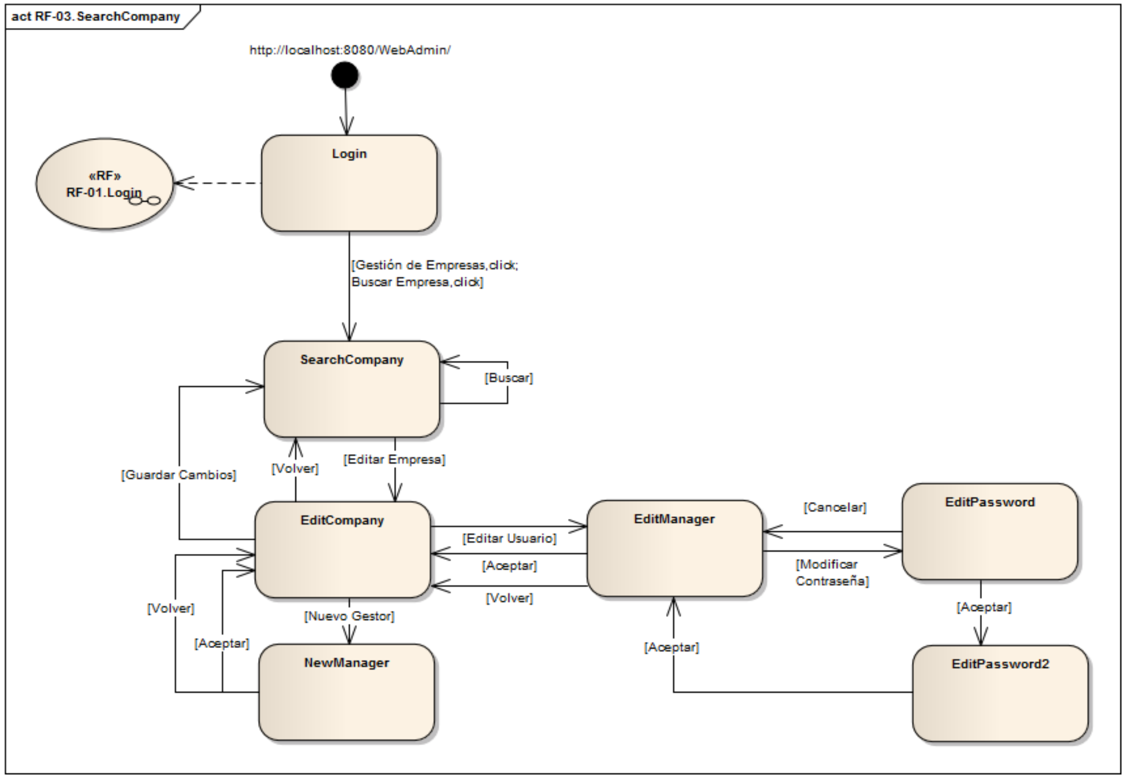
- Use cases:

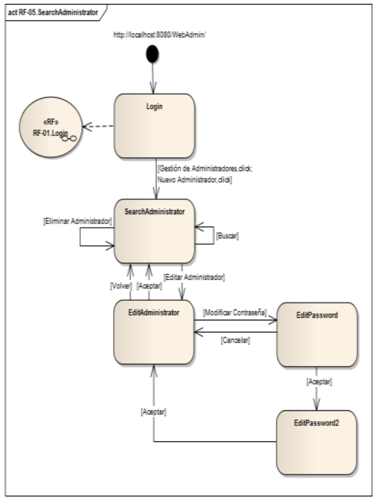


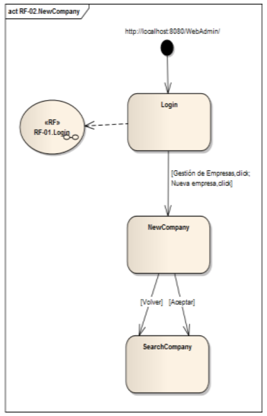
5.1. Pre-Automation (II)

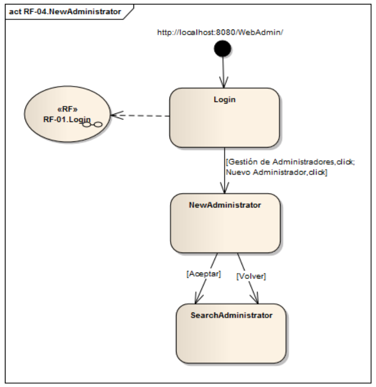
- Activity diagrams:





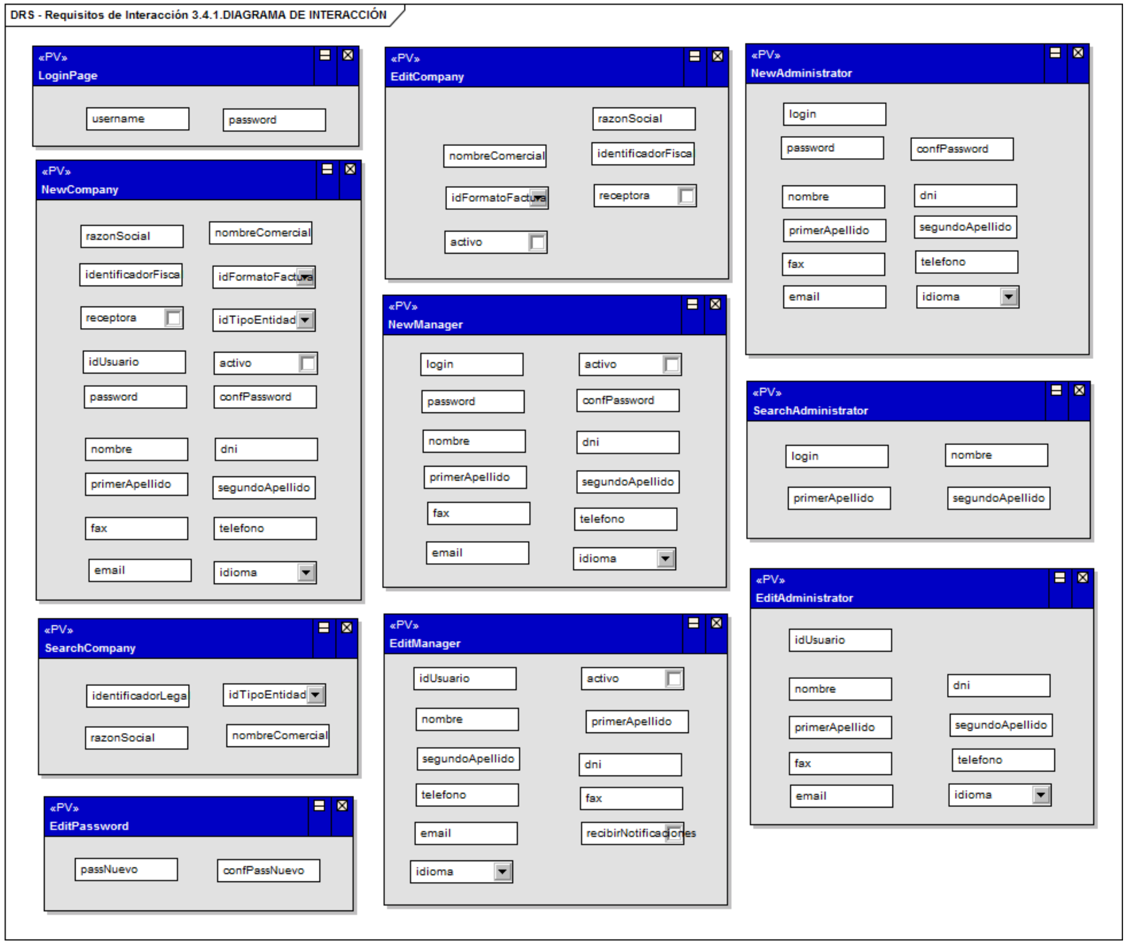





act RF-04.NewAdministrator
http://localhost:8080/WebAdmin/
«RF»
RF-01.Login
Login
[Gestión de Administradores,click;
Nuevo Administrador,click]
NewAdministrator
[Aceptar]
[Volver]
SearchAdministrator

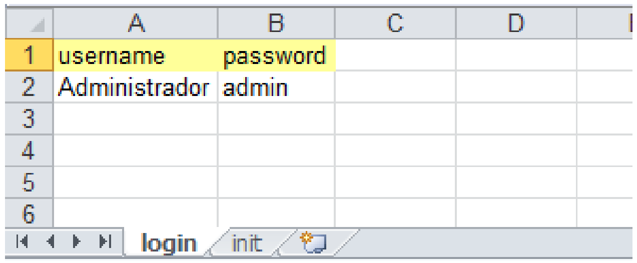
5.1. Pre-Automation (III)

- Presentation UML diagrams:

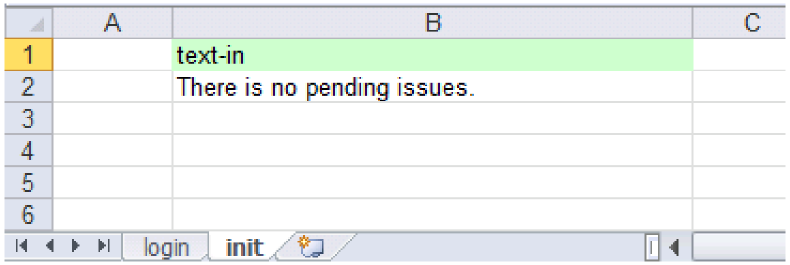


5.2. Post-Automation

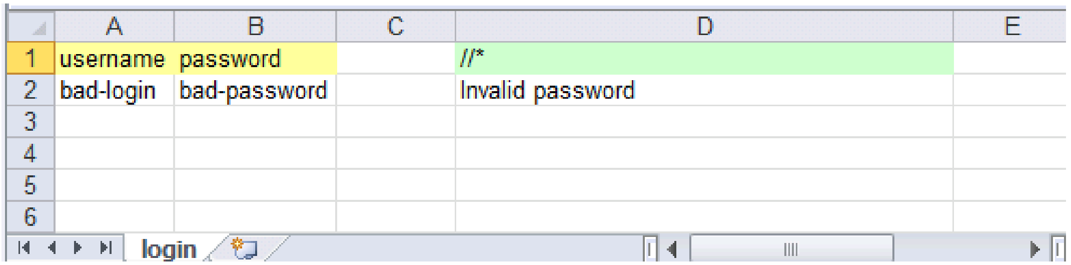
- Post-Automation: Test data and oracles in Excel files:



	A	B	C	D
1	username	password		
2	Administrador	admin		
3				
4				
5				
6				



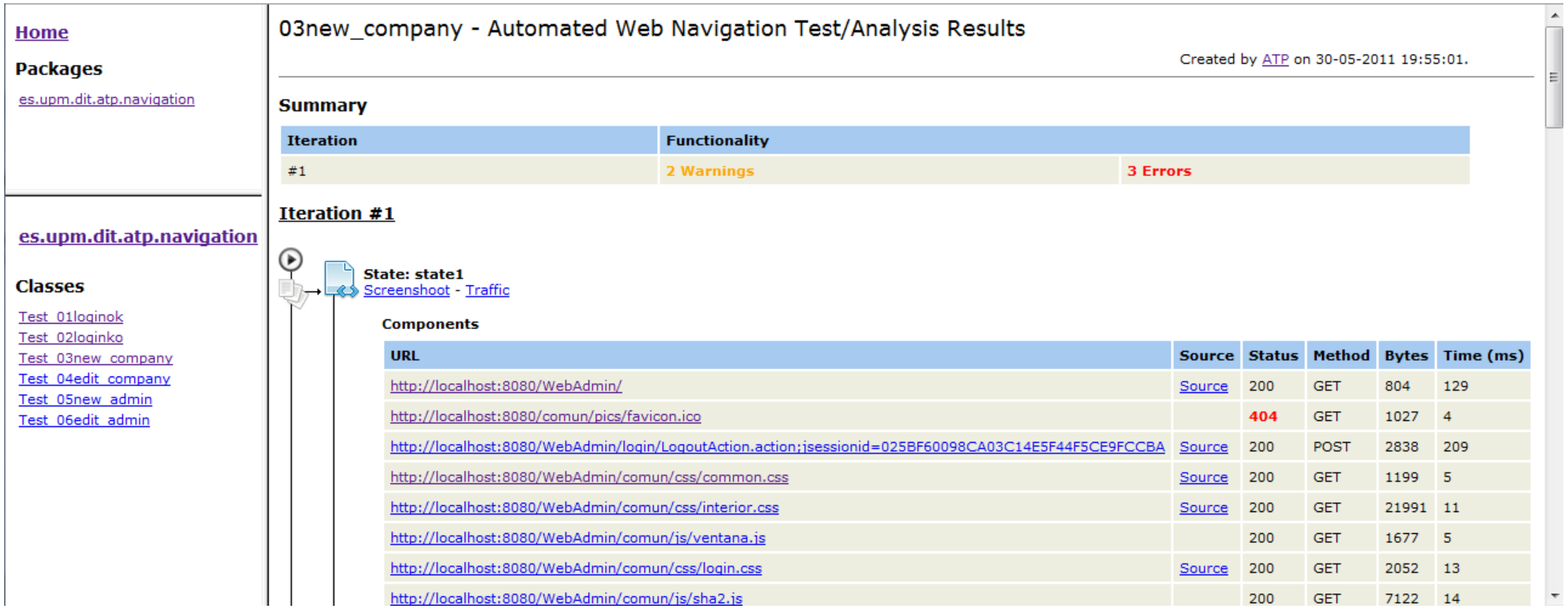
	A	B	C
1		text-in	
2		There is no pending issues.	
3			
4			
5			
6			



	A	B	C	D	E
1	username	password		//*	
2	bad-login	bad-password		Invalid password	
3					
4					
5					
6					

5.3. Results (I)

- Verdicts in the report:
  - 6 functional errors (due to broken links, not navigational issues)
  - 8 functional warning (due to JavaScript notifications)



Home

Packages

es.upm.dit.atp.navigation

es.upm.dit.atp.navigation

Classes

Test_01loginok
Test_02loginko
Test_03new_company
Test_04edit_company
Test_05new_admin
Test_06edit_admin

03new_company - Automated Web Navigation Test/Analysis Results

Created by ATP on 30-05-2011 19:55:01.

Summary

Iteration	Functionality	
#1	2 Warnings	3 Errors

Iteration #1

State: state1
Screenshoot - Traffic

Components

URL	Source	Status	Method	Bytes	Time (ms)
http://localhost:8080/WebAdmin/	Source	200	GET	804	129
http://localhost:8080/comun/pics/favicon.ico		404	GET	1027	4
http://localhost:8080/WebAdmin/login/LogoutAction.action;jsessionid=025BF60098CA03C14E5F44F5CE9FCCBA	Source	200	POST	2838	209
http://localhost:8080/WebAdmin/comun/css/common.css	Source	200	GET	1199	5
http://localhost:8080/WebAdmin/comun/css/interior.css	Source	200	GET	21991	11
http://localhost:8080/WebAdmin/comun/js/ventana.js		200	GET	1677	5
http://localhost:8080/WebAdmin/comun/css/login.css	Source	200	GET	2052	13
http://localhost:8080/WebAdmin/comun/js/sha2.js		200	GET	7122	14

5.3. Results (II)

- Answers to RQs:
  - RQ1: Factur@ has no navigation errors
  - RQ2: ATP can discover failures in web applications
  - RQ3: ATP input pros and cons:

Input	Pros	Cons
UML	• Reuse model for testing • Models describes every possible path	• Test data and oracles are not attached in the models. • Post-automation step is mandatory.
XML	• Every possible path can be depicted. • Data and oracles can be attached	• XML files must be coded and maintained by hand.
R&P	• Scripts creation using Selenium IDE against the real application. • Data and oracles can be attached to HTML scripts	• There is always a single path by HTML script (linear recording). • Error paths should be defined in different scripts.

Table of Content

6. Conclusions (I)

- This piece of research has presented a method to assess functional requirements of web applications based on its navigation
- Correct navigation structure must defined (pre-automation): UML (NDT), XML, R&P (HTML)
- Data-driven approach (post-automation) based on generated Excel files with the data extracted from the model and easily extensible
- Graph theory has been employed to model the navigation and CPP is the algorithm to find the paths

6. Conclusions (II)

- Implementation of this approach in a open-source tool: Automatic Testing Platform (ATP)
- ATP is a *mixed* testing framework:
  - 1st generation: It can use R&P linear scripts
  - 2nd generation: It is data-driven
  - 3rd generation: It can use UML models
- Future work:
  - Extension of the approach to non-functional requirements: Performance, Security, Compatibility, Usability, and Accessibility
  - Release of the implementation, ATP v2.0 (currently only available ATP v1.0 to download)

Thank you!

Boni García

bgarcia@dit.upm.es