TESTBEDS 2010 - Paris

Rich Internet Application Testing Using Execution Trace Data

Dipartimento di Informatica e Sistemistica
Università di Napoli, Federico II
Naples, Italy

Domenico Amalfitano
Anna Rita Fasolino
Porfirio Tramontana
{domenico.amalfitano, fasolino, ptramont}@unina.it

Context and Goal

- Context
  - Rich Internet Applications (RIAs)
- Goal
  - Proposing and investigating techniques, models and tools for effective testing of RIAs .

Rich Internet Applications (RIAs)

- RIAs represent the new generation of Web applications, providing richer, more interactive, dynamic and usable user interfaces than traditional ones.
- **AJAX** (Asynchronous JavaScript and XML) is a set of technologies (JavaScript, XML, XMLHttpRequest objects) providing one of the most diffused approach for implementing RIAs.
- Examples of popular RIAs: Google Maps, Flickr, Gmail.
- Hereafter we focus our attention on Ajax-based RIAs

User Interface of AJAX-based RIAs ...

- It is implemented by Web pages composed by individual components, which can be updated, deleted or added at run time independently.
- The manipulation of the page components is performed by an **Ajax engine** written in JavaScript that
  - is loaded by the **browser** at the start of the session,
  - accesses the page components by the **DOM** interface,
  - is responsible for communicating with the server to exchange few amounts of data. The communication between Client and Server may be **asynchronous.**

… User Interface of AJAX RIAs

- Its status changes due to Javascript **event handlers elaborations.**
- **Event** Handlers are triggered by **user events** or other **external events (**such as timeout events or asynchronous responses from the server).

- The User Interface of Ajax-based RIAs can be considered like an ***event-driven software*** system (EDS).

Open issues in RIA testing

- The traditional web testing approaches are based on the assumption that the **business logic** is entirely implemented by the server side of the application.
- Vice-versa, in RIAs the business logic is implemented by the client side too.
- Specific testing activities involving the client side elaborations of the RIA are needed.
- A possible approach:
- Adopting the testing approaches used for event-driven software systems for the aims of testing the UI of RIAs.

Models For Event Based Testing

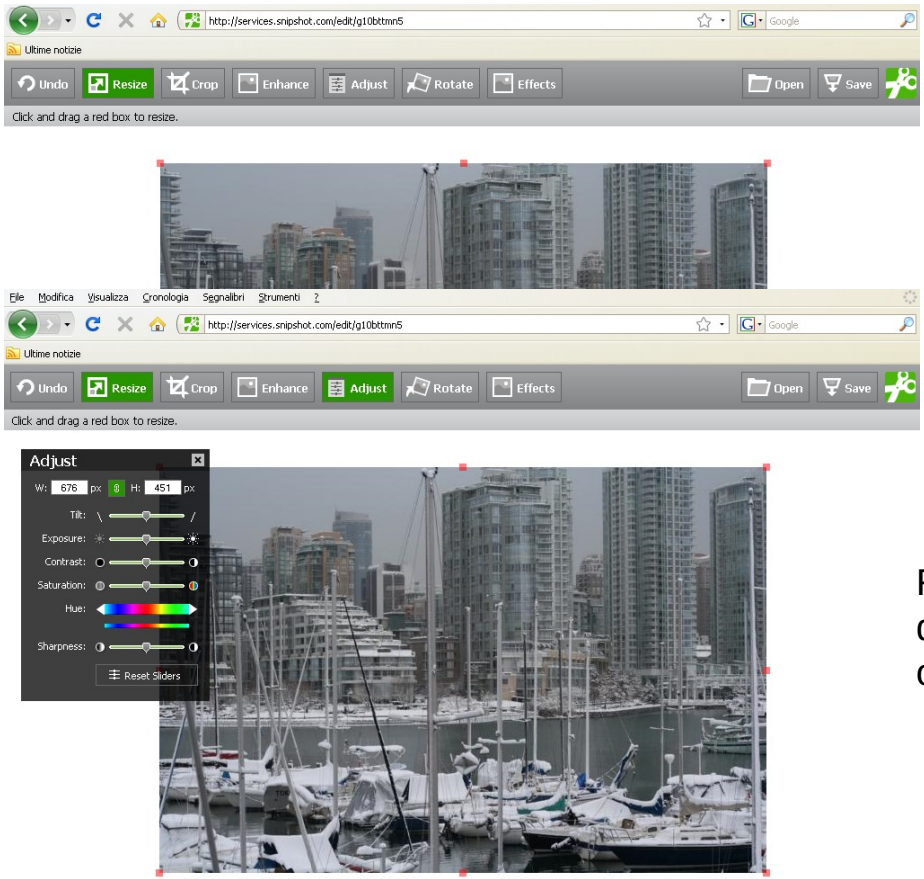
- Event based testing techniques of GUI of desktop applications are based on models such as:
  - Event Flow Graphs.
  - Event Interaction Graphs.
  - Finite State Machines.
- Due to the similarities between RIAs and GUIs, all these models may be used to model the behaviour of an RIA.
- We proposed of using FSM for representing the behaviour of RIAs.

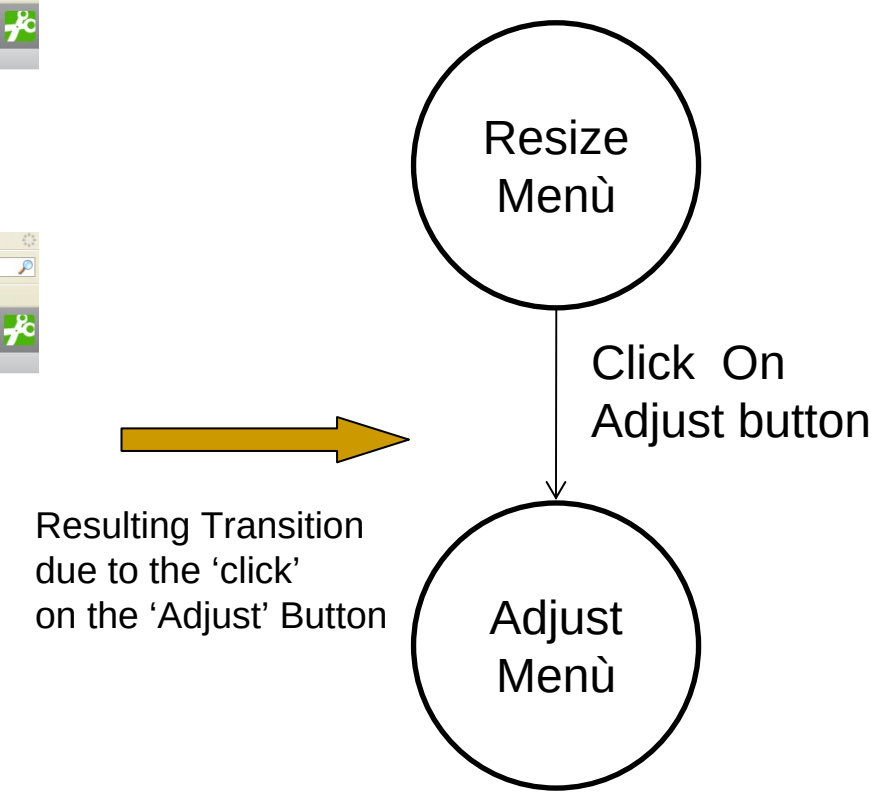
FSM Reverse Engineering Technique

- We have proposed a reverse engineering approach for obtaining the FSM that:
  - Is based on the analysis of the execution traces of the RIA;
  - solves the problem of FSM states and transitions explosion by heuristic criteria that aim at clustering equivalent states and transitions of the FSM.

1. D. Amalfitano, A. R. Fasolino, P. Tramontana, Reverse Engineering Finite State Machines from Rich Internet Applications, (WCRE 2008).
2. D. Amalfitano, A. R. Fasolino, P. Tramontana, Experimenting a Reverse Engineering Technique for Modelling the Behaviour of Rich Internet Applications, (ICSM 2009).
3. D. Amalfitano, A. R. Fasolino, P. Tramontana, An Iterative Approach for the Reverse Engineering of Rich Internet Applications, (IARIA ICIW 2010).

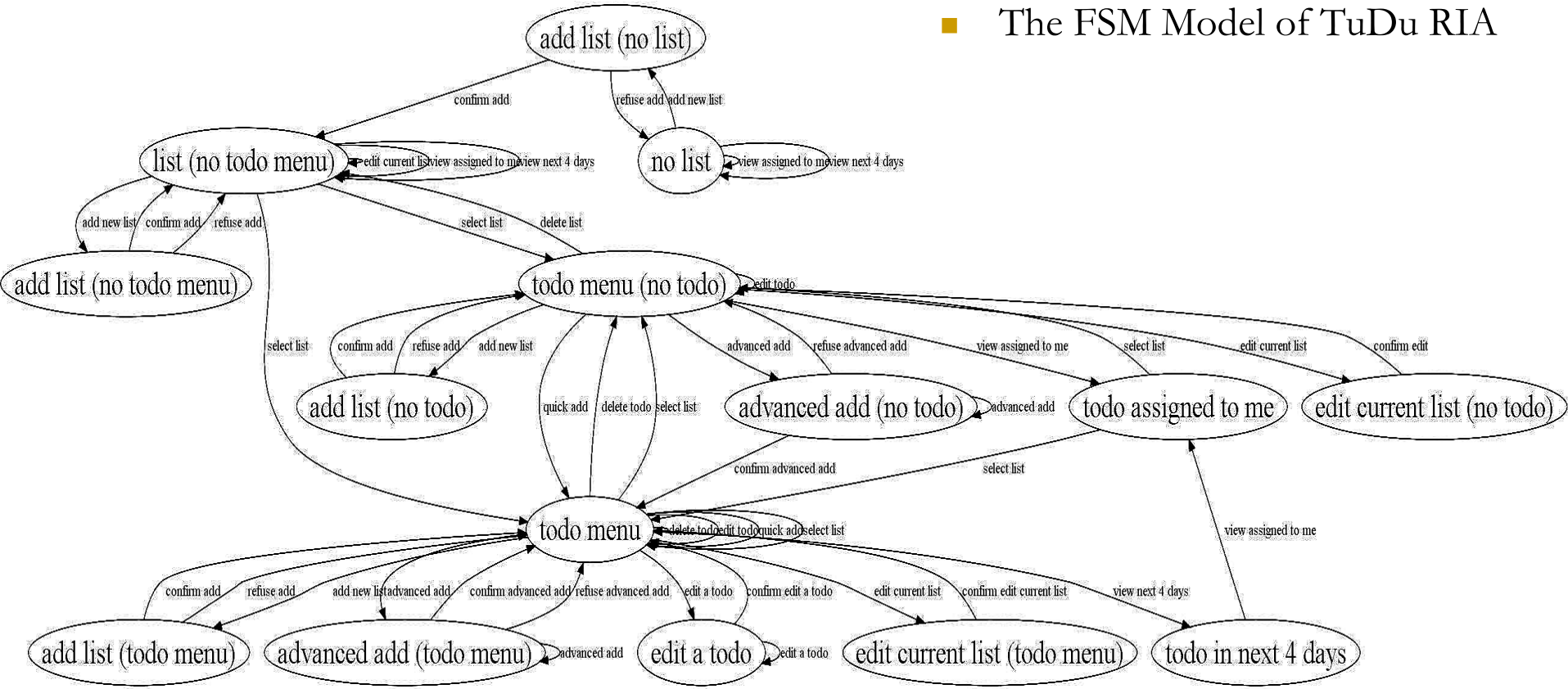
FSM - an example of transition





Example of a reverse engineered FSM Model

- The FSM Model of TuDu RIA



User-session based testing: a possible approach for RIA testing?

- Some state-based testing techniques relying on the FSM have recently been proposed for RIA testing.
- User-session based testing has not yet been used in the context of RIAs
- This approach aims at automatically generating test cases composed of **event sequences** which are deduced by analysing user interactions with a version of the application.
- Already applied with success for:
  - traditional Web application testing,
  - GUI automated testing.

The proposed technique

- We decided to investigate user-session based testing in the context of RIAs.
- We propose a testing technique that is based on the following activities:
  1. Collection of a set of execution traces of the application;
  2. Test suite generation;
  3. Test suite reduction.

1. Execution trace collection

- An execution trace is defined as a sequence of couples: …, <Interf. $State_i$, $event_i$>, <Interf. $State_{i+1}$, $event_{i+1}$>, …
- Two different approaches have been considered for collecting execution traces:
  - A **manual approach** based on the analysis of the interactions with an RIA of real users or testers.
  - An **automatic approach** based on Crawling techniques.

2. Test Suite Generation

- The test suite is generated by transforming each execution trace into a test case.
- The transformation requires that two problems are solved:
  - A) definition of pre-conditions of each test case;
  - B) definition of the Test Oracle.

2.A Pre-conditions of a test case

- In general the behaviour of an RIA depends on the current state of the application data as well as by its environment and session data.
- Our solution:
  - before recording each execution trace, we set the RIA in pre-defined states.
  - These states will provide the pre-conditions of the related test cases.

2.B The Test Oracle problem.

- A testing oracle is needed to define the PASS/FAIL result of a test case execution.

- We decided to evaluate test case results by checking the occurrence of JavaScript crashes.
  - No JS crash → Test Passes.
  - JS crash → Test Fails.

3. Test Suite Reduction

- Given a test suite, the reduction technique computes its minimal set of test cases assuring the same coverage of the original test suite.
- Three **reduction techniques M1, M2, and M3** have been proposed, that consider different types of coverage:
  - M1 covers the same set of **FSM states** covered by the original suite;
  - M2 covers the same set of **FSM transitions** (or events) covered by the original suite;
  - M3 covers the same set of **JS code components** (such as functions) as the original test suite.

Experiment

- We performed an experiment for evaluating the proposed testing approach.
- We considered twelve combinations of different execution trace collection and reduction techniques.

Technique	Execution trace collection	Reduction Technique
B1	By user sessions	-
B2	By crawling	-
B3	By user sessions and crawling	-
T1	By user sessions	M1
T2	By user sessions	M2
T3	By user sessions	M3
T4	By crawling	M1
T5	By crawling	M2
T6	By crawling	M3
T7	By user sessions and crawling	M1
T8	By user sessions and crawling	M2
T9	By user sessions and crawling	M3

The testing techniques considered in the experiment

Research Questions

- The experiment was designed to address the following research questions:
  - ***RQ1.*** How effective are the testing techniques B1, B2, and B3 (without reduction)?
    - We decided to evaluate the effectiveness of the B1, B2, and B3 techniques in terms of the **coverage** and **fault-detection** they provide.
  - ***RQ2.*** How effective are the reduction-based T1… T9 techniques with respect to the B1, B2, and B3 techniques?
    - This question concerns the relationship about the performance of the B1, B2, and B3 techniques with respect to the T1…T9 techniques, in terms of the coverage and fault-detection they provide.

Measured variables

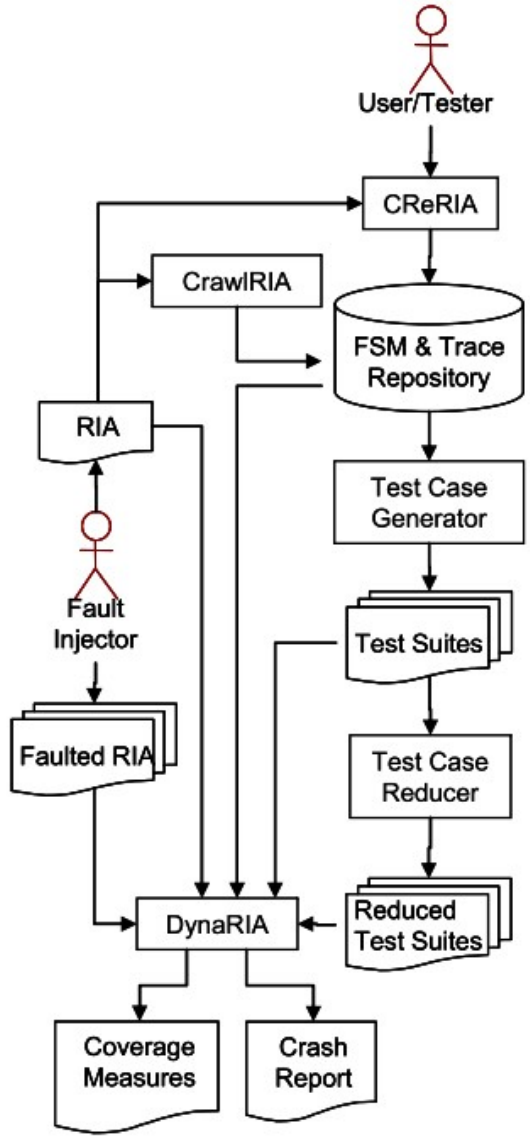
- The following variables are measured during the experiment:
  - **FSM State Coverage:** percentage of FSM states covered by at least one test case of the test suite.
  - **FSM Transition Coverage:** percentage of FSM transitions covered by at least one test case of the test suite.
  - **JavaScript function Coverage:** percentage of JavaScript functions executed during the test suite execution with respect to the number of script functions contained by the JavaScript modules of the application.
  - **JavaScript LOC Coverage:** percentage of JavaScript function LOC executed during the test suite execution with respect to the LOC of JavaScript functions of the application.
  - **Fault detection effectiveness:** percentage of faults detected by the test suite.
  - **Test Suite Size:** number of test suite test cases.
  - **Test Suite Event Size:** number of events exercised by the test suite test cases.

Supporting tools

- The experimental process has been carried out with the support of a set of tools.
  - **CReRIA & CrawlRIA:** both the tools are used to record executions traces, manually and automatically respectively. The user session traces (sequences of interfaces and events) and the corresponding paths on the abstracted FSM (sequences of states and transitions) are stored in the FSM & Trace Repository.
  - **Test Case Generator:** tool able to transform the execution traces stored in the FSM & Trace Repository in a test suite composed of executable test cases by the DynaRIA tool.
  - **Test Case Reducer:** tool able to reduce a test suite ***ts*** into a smaller one that satisfies the same ***ts*** coverage requirements.
  - **DynaRIA:** tool for dynamic analysis and testing of RIAs. It is able to execute the test cases produced by the Test Case Generator or the Test Case Reducer tool, to monitor their execution in a browser environment, to produce a report of detected crashes and to report the coverage measures.

Experimental Process

- The experimental process is shown in Figure.
- Chosen the subject RIA, execution traces are both manually collected (using the CreRIA tool) and automatically by the CrawlRIA tool. Produced traces are stored in the repository.
- The Test Case generator tool produces test cases from the collected execution traces.
- The Test Case Reducer applies the minimization techniques and produces reduced test suites.
- The produced test suites are submitted to the DynaRIA tool for the execution.
- The DynaRIA tool evaluates the results of all test case executions both ***on the original version of the RIA, and on a set of RIA versions in which an expert has injected faults.***



The experimental testing process

Subject application

- The subject application is ***Tudu***, an open source RIA offering 'todo' list management facilities.
- To evaluate the fault detection capability of proposed testing techniques, different types of fault have been injected in the JavaScript (JS) code of the subject application.
- 19 faults able to produce JS crashes were injected and 19 versions of Tudu were produced, each one containing just one fault.
- The faults were of different types, such as:
  - JS function call instructions with undefined, incorrect, or missing parameters;
  - JS syntax errors;
  - array out of bound errors,
  - server requests of missing resources or JS files.

Experimental Data Collection

- 203 execution traces were automatically collected by CrawlRIA.
- 21 user sessions were manually collected with CReRIA to exercise all the application's known use cases and their scenario.
- Collected traces were used for automatically abstracting a reference FSM model of the application.
- This model was used for the test suite reduction

Collected Interfaces	2223
Triggered Events	1999
FSM States	19
FSM Transitions	61
Defined JS Function #	1018
Defined JS Function LOC	6150

Overview information about collected execution traces

…Experimental Results

	US	US-M1	US-M2	US-M3
Test Case #	21	3	9	10
Event #	518	81	232	235
Covered States	19	19	19	19
Covered States %	100%	100%	100%	100%
Covered Transitions	56	40	56	56
Covered Trans. %	91,8%	65,6%	91,8%	91,8%
Covered Functions	172	160	163	172
Covered Funct. %	16,9%	15,7%	16,0%	16,9%
Covered LOC	1016	967	980	992
Covered LOC %	16,5%	15,7%	15,9%	16,1%
Revealed faults #	19/19	16/19	19/19	19/19

Data about **user session** test suites

	CR	CR-M1	CR-M2	CR-M3
Test Case #	203	5	20	23
Event #	1481	42	134	273
Covered States	14	14	14	14
Covered States %	73,7%	73,7%	73,7%	73,7%
Covered Transitions	35	16	35	35
Covered Trans. %	57,4%	26,2%	57,4%	57,4%
Covered Functions	160	141	153	160
Covered Funct. %	15,7%	13,9%	15,0%	15,7%
Covered LOC	949	875	929	937
Covered LOC %	15,4%	14,2%	15,1%	15,2%
Revealed faults #	17/19	9/19	17/19	17/19

Data about test suites from **crawled** traces

	HY	HY-M1	HY-M2	HY-M3
Test Case #	224	3	21	24
Event #	1999	81	261	283
Covered States	19	19	19	19
Covered States%	100%	100%	100%	100%
Covered Trans.	61	40	61	61
Covered Trans.%	100%	65,6%	100%	100%
Covered Funct.	192	160	164	192
Covered Funct.%	18,9%	15,7%	16,1%	18,9%
Covered LOC	1042	967	987	1022
Covered LOC%	16,9%	15,7%	16,0%	16,6%
Revealed faults #	19/19	16/19	19/19	19/19

Data about test suites obtained from both **user sessions and crawled (hybrid)** traces

Fault detection results of the test suites without reduction (R1)

- The User Session based testing technique detected all 19 injected faults
- The Crawler based testing techniques detected 17/19 faults
- The hybrid technique (user session + crawler) obtained the same results of the user session based one.

Coverage results of the test suites without reduction (R1)

- The User Session based technique covered all states, 91.8% transitions, and 172 JS functions.
- The Crawler based technique covered 73.7% states, 57.4% transitions, and 160 JS functions.
- The hybrid technique covered 100% states, 100% transitions, and 192 JS functions.

Fault detection and coverage results of the test suites with reduction (R2)

- The techniques with the best **fault detection** were the ones based on FSM transitions and JS function coverage.
- As to the **coverage** the best technique was the one based on the JS function coverage.

Conclusion…

- In the paper we have presented a testing technique for RIAs that transforms execution traces of an existing application into executable test cases.
- For achieving the technique's scalability, we employed a test suite selection technique that reduces the size of obtained test suites.
- For exploring the feasibility and effectiveness of the technique an experiment involving an open-source RIA application was carried out.
- In the experiment, different approaches (both human-based, and automatic) for execution trace collection and several criteria for reducing the test suites were analyzed .

…Conclusion

- The experiment showed that test suites produced automatically by means of a crawler are not more effective than suites derived from user session traces, but the former ones have the advantage of being automatically obtained.

- The more effective testing strategy is the hybrid one that combines test cases obtained by both approaches:
  - test cases automatically obtained by an RIA crawler should be used for discovering the most of application defects.
  - test cases based on user session data could be employed to obtain a wider coverage of defects.

Future works

- The validity of obtained experimental results is reduced, due to several limitations of the experiment we performed, such as
  - the single RIA application involved,
  - the small number of collected user sessions;
  - the single user involved in the collection;
  - the single initial state of the application during trace collection.
  - faults injected in the application were just of a particular type (i.e. faults causing JavaScript crashes),
  - faults affecting the RIA behaviour without causing crashes were not considered;
  - the technique adopted for abstracting the FSM model of the RIA provides just an approximate model of the RIA behaviour.
- To overcome these limitations, further investigations and a wider experimentation will be carried out in future work.

Thank you for your attention.