Rich Internet Application testing: open issues, questions, proposals

Domenico Amalfitano
Anna Rita Fasolino
Porfirio Tramontana

Dipartimento di Informatica e Sistemistica
University of Naples Federico II, Italy
Context and Motivation

- Context:
  - Rich Internet Application Testing

- Motivation:
  - Identifying a set of open issues and questions regarding RIA testing
  - Outlining possible solutions for a joint discussion.
Outline

- RIA representation models and Reverse Engineering techniques
- RIA testing techniques based on execution traces
- Supporting tools
RIA user interface representation models

- **Event-Interaction-Graph** [A. M. Memon and Q. Xie. Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software. *IEEE TSE*, 2005]
  - Models the flow of events on the RIA UI

- **Finite State Machine (FSM) Model**
  - Provides an abstraction on the UI of the RIA made up of states and transitions
1. FSM models the behaviour of an RIA.
2. FSM represents all the elaboration states where the RIA receives any input solicitation by its user.
3. Each state of the RIA is described by the client Interface shown to the user at that interaction time.
4. Each client interface is characterized only by the sub-set of its widgets that are ‘clickable’ or, more in general, that have a registered event listener and a corresponding event handler.
5. Transitions are associated with user events that trigger the RIA migration towards a new state.
The FSM Reverse Engineering problem has been addressed in some recent papers:


Papers [1, 2, 3] describe a process based on the analysis of execution traces of the RIA, that generates a FSM model that needs to be validated manually at the end of the process.

Paper [4] describes an iterative variant of this process, where the manual validation of the FSM is performed at each process iteration.

This process will be illustrated in the following slides.
The iterative Reverse Engineering Process [4]

1. **User Interaction**: a user interacts with the RIA and triggers an event

2. **Extraction**: information about current interface, fired user event and the user interface that is obtained after the event processing are automatically retrieved and persistently stored

3. **Abstraction**: heuristic criteria evaluate the degree of similarity of the current user interface with the previously produced ones and propose a classification of the interface

4. **Concept Assignment**: the software engineer validates the classifications proposed by the heuristic criteria and accepts or refuses them. If the classification is refused, he has to propose the correct concept to be assigned. In this way, the expert incrementally reconstructs a FSM modeling the behavior of the RIA GUI

5. **Termination criterion evaluation**: event coverage, or coverage of known scenarios, or the effort (e.g., the time spent) devoted to the whole process can be considered.

![FSM Model Diagram]
An example of obtained FSM

- The FSM Model of the application TuDu:
  http://app.ess.ch/tudu/welcome.action
Some details about the interface clustering approach

Examples of widgets extracted from a RIA interface and used for interface clustering:

Test Ajax Application

An Example of Client Interface and the selected widgets:

1 – Form
2 – Button
3 – Button
4 – Button
5 – Button
6 – Link
7 – Link
## Examples of widget attributes

<table>
<thead>
<tr>
<th>Id_widget</th>
<th>Attribute</th>
<th>Value</th>
<th>Xpath</th>
<th>Unindexed xpath</th>
<th>Unindexed xpath ID</th>
<th>Active</th>
</tr>
</thead>
</table>
Heuristic criteria for interface clustering

- **C1)** two interfaces are equivalent and can be clustered if they have the same number of widgets with the same subset of attributes.

- **C2)** two interfaces are equivalent and can be clustered if they have the same number of *enabled* widgets with the same subset of attributes.

- **C3)** two interfaces are equivalent and can be clustered if they contain:
  - *the same set of containers having the same type of widgets.*

- **C4)** two interfaces are equivalent and can be clustered if they contain *the same set of containers with the same ID, containing the same type of widgets.*

- A container is implemented by tags such as DIV, TABLE, etc…
Examples of heuristic clustering criteria

Applications

- Interface I₁ is equivalent to interface I₂ according to C₁, while they are not equivalent according to C₂
- Interface I₂ is equivalent to interface I₃ according to C₃, while they are not equivalent according to C₂
- Interface I₃ is equivalent to interface I₄ according to C₃, while they are not equivalent according to C₄
A first observation

- The proposed reverse engineering process relies on a set of RIA execution traces which are manually collected.
- In order to improve the efficiency of the process, execution traces may be collected automatically. Our current work is addressing the following questions:

  - **Question 1:** is it possible to reconstruct the FSM model using a set of artificially built execution traces, such as those obtained using a RIA ripping (crawling) technique?

  - **Question 2:** how should the reverse engineering process be modified?
A possible variant of the Reverse Engineering Process based on RIA crawling techniques

Similar to the iterative process, except for:
- The Interaction with the RIA is performed by automatic firing of user events;
  - For each RIA user interface, the set of firable events are stored in a list and triggered according to a given navigation strategy (such as depth first/breadth first ones);
- In the Abstraction step, just one heuristic criterion is used for the clustering (the criterion is a process input);
- The Concept Assignment step may not be executed at all.

The output of the process will include:
- a FSM=(S, T, E), S={ GUI state}   T={ transition }   E={event}
- A set of automatically generated execution traces.

Relevant Process factors:
- The event navigation strategy (breadth first or depth first)
- The criterion used to stop the exploration
- The interface clustering criterion.
Details about the RIA Crawling based process

- The analysis starts from the home page of the RIA (with a reset db and restarted Web server)
- For each RIA user interface, the set of firable events are stored in a list and navigated according to a depth first/breadth first strategy
- An event triggering causes the reaching of a new interface
  - If the interface is equivalent to an already visited one, according to the chosen heuristic equivalence criteria
    - The navigation is stopped
    - The sequence of events triggered from the home page to the current one are saved
    - The navigation restarts from the home page and follows the direction of an event that has not already been triggered
      - Optionally, another trace termination criterion based on a maximum depth could be considered
Some questions

- Using the proposed Reverse Eng. processes, we are able to obtain a set of RIA execution traces.

- **Question 3**: may we use the execution traces (either generated by RIA crawling, or by manual exploration of the application) for the aims of testing?

- **Question 3.1**: which test case generation technique can be proposed?
A first test case generation technique

- **Test generation technique**: Transform each execution trace into a test case.

- Test Case—Sequence of fired events and assertions
  - Fired events are described as events on widgets of an interface
  - Assertions may be of two types:
    - based on the verification of generic properties of the RIA behavior, such as:
      - absence of Javascript crashes;
      - absence of server response timeouts (e.g. for missing server resources);
    - Based on the verification of equivalence conditions between expected GUI states and actual ones.

- Test Case Set-up requires start operations such as: db reset to a known state, Web server restart, RIA home page reload
Testing problems associated with the proposed test generation technique

- **a)** Test Suite Minimization problem

- **b)** Assessing the effectiveness of test suite generation techniques

- **c)** Definition of test oracles
A) Test Suite Minimization problem

- Test suites generated by the proposed technique may need to be reduced in order to improve testing process efficiency.
- Which reduction techniques are applicable?
- Can we use classical reduction techniques to get minimal test suites that assure pre-defined coverage criteria?

- A reduction technique was proposed in Giuseppe A. Di Lucca, Anna Rita Fasolino, Porfirio Tramontana: A Technique for Reducing User Session Data Sets in Web Application Testing. WSE 2006: 7-13
Examples of coverage criteria usable for minimization aims

- With respect to the abstracted FSM:
  - FSM state coverage
  - FSM transition coverage
  - FSM event coverage
  - FSM K-length path coverage

- With respect to the application code:
  - Javascript function call coverage
  - Javascript function call sequence coverage [Memon et al. Call-Stack Coverage for GUI Test Suite Reduction, IEEE TSE 2008]
  - Javascript function code coverage
B) Assessing test suite effectiveness

- A possible approach:
  - assessing the effectiveness of test suites by evaluating their defect detection capability on a fault-seeded version of the RIA.

- Faults can be defined:
  - According to a RIA fault model, or
  - By analysing known defects of the RIA retrieved from its CVS.

- Testing Effectiveness evaluation can be based on the number of detected faults.
C) Testing Oracle definition

- Which types of testing oracles can be used?
- Which types of assertions are usable?

Assertions may specify:
- A) generic properties of the RIA behavior, such as:
  - Absence of Javascript crashes;
  - Absence of server side exceptions (such as response timeouts for missing server resources);
- B) specific properties of the RIA behavior that can be deduced from client interface analysis, such as:
  - equivalence conditions between expected GUI states and actual ones.
Validation

- The outlined test generation approach, and possible solutions to the listed problems should be investigated by experiments.
- Experimental activities will have to be designed and executed with tool support.
Most of the testing activities can be supported by our prototype tools:

- a tool for recording user sessions (and generating the FSM of the RIA) (CReRIA) (already available)
- A crawler tool for generating execution traces automatically (CRAWLRIA) (In Progress)
- A tool for transforming execution traces into Selenium-executable test cases, and implementing a test suite minimization technique (TestRIA) (in progress)
- A tool that replicates execution traces and performs several coverage analyses, and detects run-time Javascript errors and server exceptions (DynaRIA) (already available)
Tool support (2/3)

- **CreRia**
  - Supports the interactive and automatic reverse engineering processes
  - Collects user session traces and assesses their coverage degree as:
    - #user events fired on analysed interfaces/#user events that are firable

- **CrawlRia**
  - Crawls a RIA according to a given heuristic clustering criterion, and produces a set of session traces

- **TestRia**
  - Transforms traces (obtained either by the crawlRia or by CReRia) in executable test cases (as Selenium executable tests)
  - Minimise test suites according to coverage criteria
Tool support (3/3)

- **DynaRIA**
  - Executes collected traces in order to detect:
    - JS errors
    - Network exceptions
  - Extracts a log of the sequence of executed scripts and network requests (synchronous and/or asynchronous) during trace execution
  - Assesses the Javascript coverage of an execution trace as:
    - #JS function executed/# JS functions
    - #JS code line executes/#JS code lines
  - Generates interaction diagrams (e.g. sequence diagrams) of the traced executions
  - Generates a EFG of traced executions
A snapshot from CreRia TOOL

Add a new list

Description:

Allow RSS publication? ☐

[ Submit ]  [ Cancel ]

URL: http://ipp.cs.vt/ippu/ippwelcome.action
User Name: SoftwareEngineeringGroup
Trace Name: T2ExperimentH2

Automatic Suggestion:
- Core

Recorded Interfaces in the Current Trace:
- C1 % 100
- C2 % 100
- C3 % 100
- C4 % 100

Insert a Label For the Current State:
- no label

C1 Suggestion:
- Add C1 Suggestion

T12 Suggestion:
- Add T12 Suggestion

T2C1 Suggestion:
- Add T2C1 Suggestion

T1C2 Suggestion:
- Add T1C2 Suggestion

T1C3 Suggestion:
- Add T1C3 Suggestion

T1C4 Suggestion:
- Add T1C4 Suggestion

T2C2 Suggestion:
- Add T2C2 Suggestion

T2C3 Suggestion:
- Add T2C3 Suggestion

T2C4 Suggestion:
- Add T2C4 Suggestion

- DB connect done
- DB connection open
- T1C1 completion completed
- Event 001 handler terminated
- DB building phase captured
- DB connection open
- T1C1 completion done

T2C1 completion completed
testCaseRia GUI: A snapshot
JS function call - Network request in a sequence diagram model (3 - level view)
JS function call - Network request in a sequence diagram model (multi-level view)
Conclusions

- We wait for your feedback about these ideas.
- We would like to present some of these ideas in a position paper for TESTBEDS.
- We are trying to obtain some preliminary experimental data to discuss at TESTBEDS.
- We may present some of our tools either in a tool demo in TESTBEDS, or in a slideshow to share via Web.