State-Based Testing of Ajax Web Applications

A. Marchetto, P. Tonella and F. Ricca

CMSC737

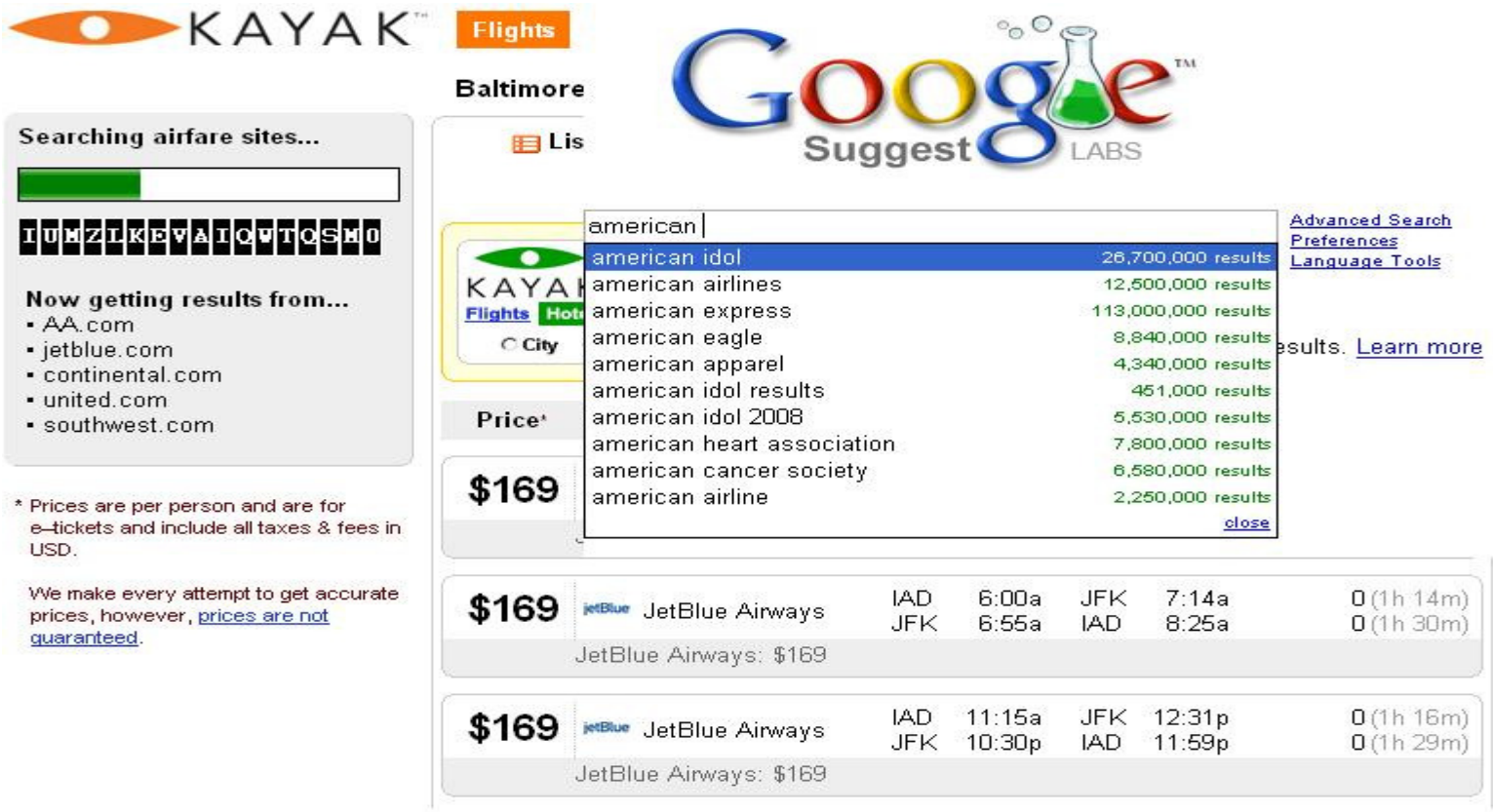
Spring 2008

Shashvat A Thakor

Outline

- Introduction
- Ajax Overview
- Ajax Testing
  - Model Extraction
  - Semantic Interactions
  - Test Case Derivation
  - Asynchronism warnings
- Experiment Results
- Conclusions and Future Work

Introduction

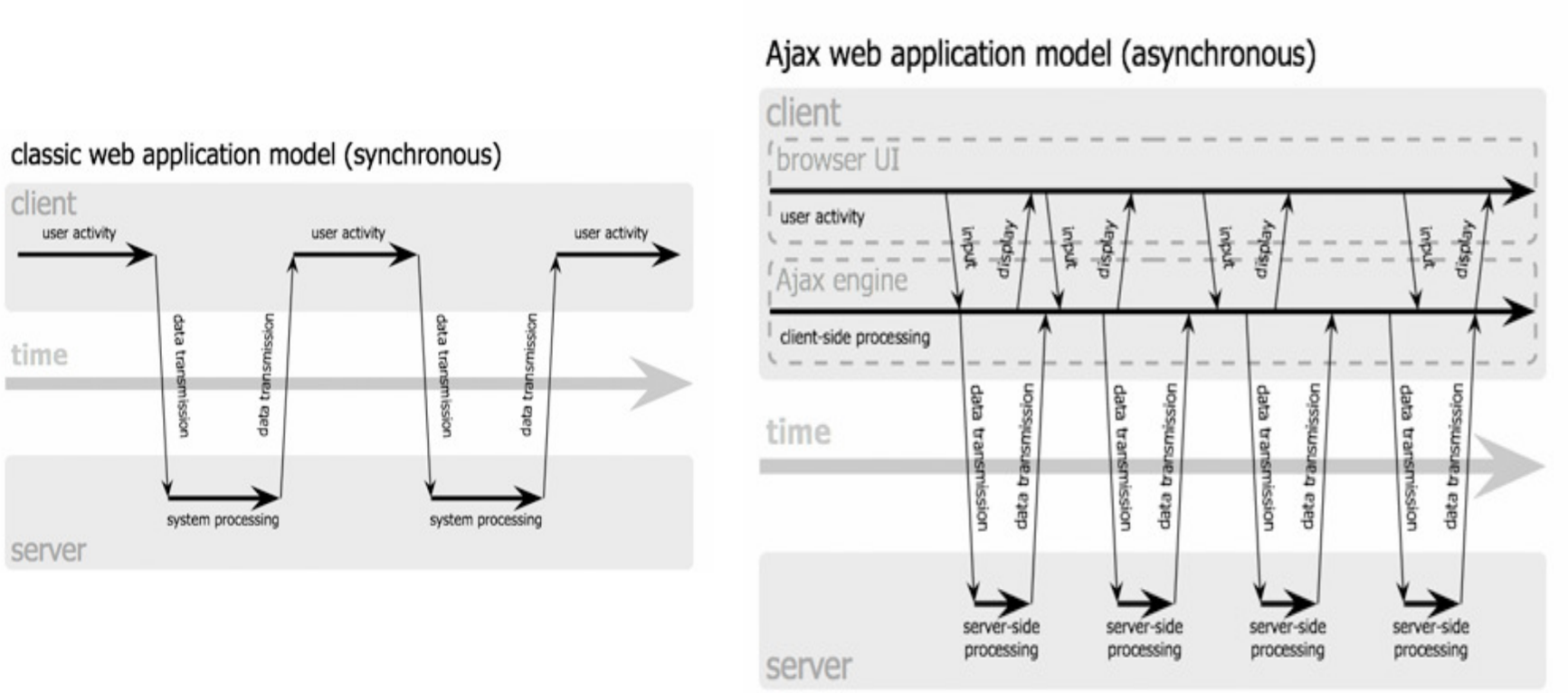


What is so special about these web-sites?

Asynchronous Javascript And XML

- A group of technologies:
  - HTML and CSS for presentation
  - Dynamic display and interaction using the Document Object Model (DOM)
  - Asynchronous data retrieval from server using *XMLHttpRequest*
  - Data interchange and manipulation by XML
  - Javascript for binding everything together.

Synchronous vs. Asynchronous



Source: Adaptive Path

Ajax Testing

- Ajax faults are associated with
  - Asynchronous messages
    - Unintended interleaving of server messages
    - Swapped callbacks
  - DOM manipulation
    - Incorrect DOM state
- Model Ajax applications by Finite State Machine (FSM)
  - States: DOM instances
  - Transitions: effects of callback executions

Model Extraction

- Dynamic analysis + static code analysis + manual validation step
- *Cart* application written in Ajax
- Starts with execution traces
- State abstraction function to get DOM states
- Only Method invocations changing DOM get selected as transitions
- Determine set of methods possibly affecting the DOM by means of a static code analysis – trace them

Trace	Event sequence
1	add
2	rem
3	add, add, rem
4	add, add, rem, rem, rem
5	add, empty
6	add, empty, rem

Figure 1. Traces for Cart (events only)

DOM element	Abstraction
DIV | SPAN | P	*null* | *empty*
TEXTAREA	*null* | *empty* | *notEmpty*
FORM	*null* | *notNull*
OL | UL	*null* | #LI=0 | #LI>0
TABLE	*null* | (#TD|#TR)=0 | (#TD|#TR)>0
INPUT type=text	*null* | *empty* | *notEmpty*
INPUT type=button	*null* | *notNull*
A	*null* | *notNull*
IMG	*null* | *notNull*
LI	*null* | *empty* | *notEmpty*
INPUT type=radio	*null* | *notNull*
SELECT	*null* | *empty* | *sel=1* | …
INPUT type=text name=total	*null* | *total=0* | *total>0*

Figure 2. Fragment of the default state abstraction function (top) and Cart-specific abstraction (bottom)

Model Extraction

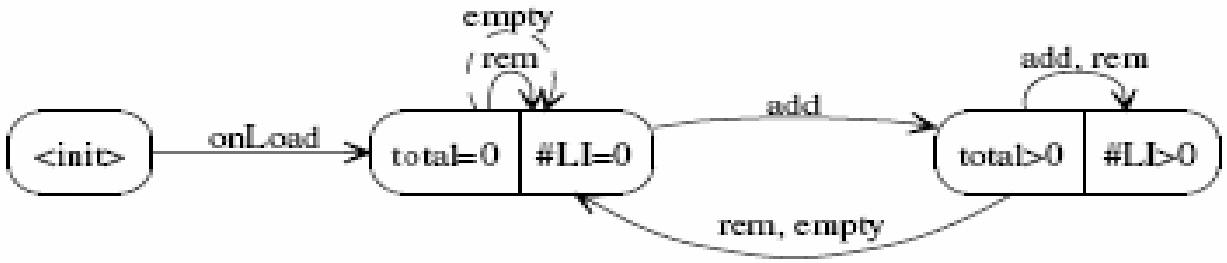


Figure 3. FSM for Cart application

- No execution trace covers dashed transition
- Hence, manual step to produce augmented FSM consisting of missing states and transitions
- Manual work depends upon number and quality of traces

Semantic Interaction

Definition 1 (Semantically interacting events) *Events e_1 and e_2 interact semantically if there exists a state S_0 such that their execution in S_0 does not commute, i.e., the following conditions hold:*

$$S_0 \Rightarrow_{e_1;e_2} S_1$$
$$S_0 \Rightarrow_{e_2;e_1} S_2$$
$$S_1 \neq S_2$$

where S_0 is any state in the FSM of the Ajax application.

- For *Cart* example, add and rem interact semantically, since *<add,rem> ≠ <rem,add>*

Test Case Derivation

- Sequence of semantically interacting events + input values from a database → executable test case
- Execute using Selenium Tool
- Pass/Fail
  - Compare concrete state sequence w.r.t. state sequence of FSM
  - Checking the output value against a provided oracle
  - Whether application crashed or not!
- Rationale : Substantial reduction of test suite size by considering only semantically interacting events – hence event sequence of length 2 is more useful than covering all the paths of length 2

Asynchronous Warnings

Nominal (no reordering): $\langle r_1; c_1; r_2; c_2 \rangle$
AsyWarn1 (swapped callbacks): $\langle r_1; r_2; c_2; c_1 \rangle$
AsyWarn2 (dependent request): $\langle r_1; r_2; c_1; c_2 \rangle$

Detectected by

- AsyWarn2 → Dependency analysis

- AsyWarn1→ Sequentialize using request queue

How to protect?

- AsyWarn2 →Disable GUI widget associated with request depending upon a callback not yet executed

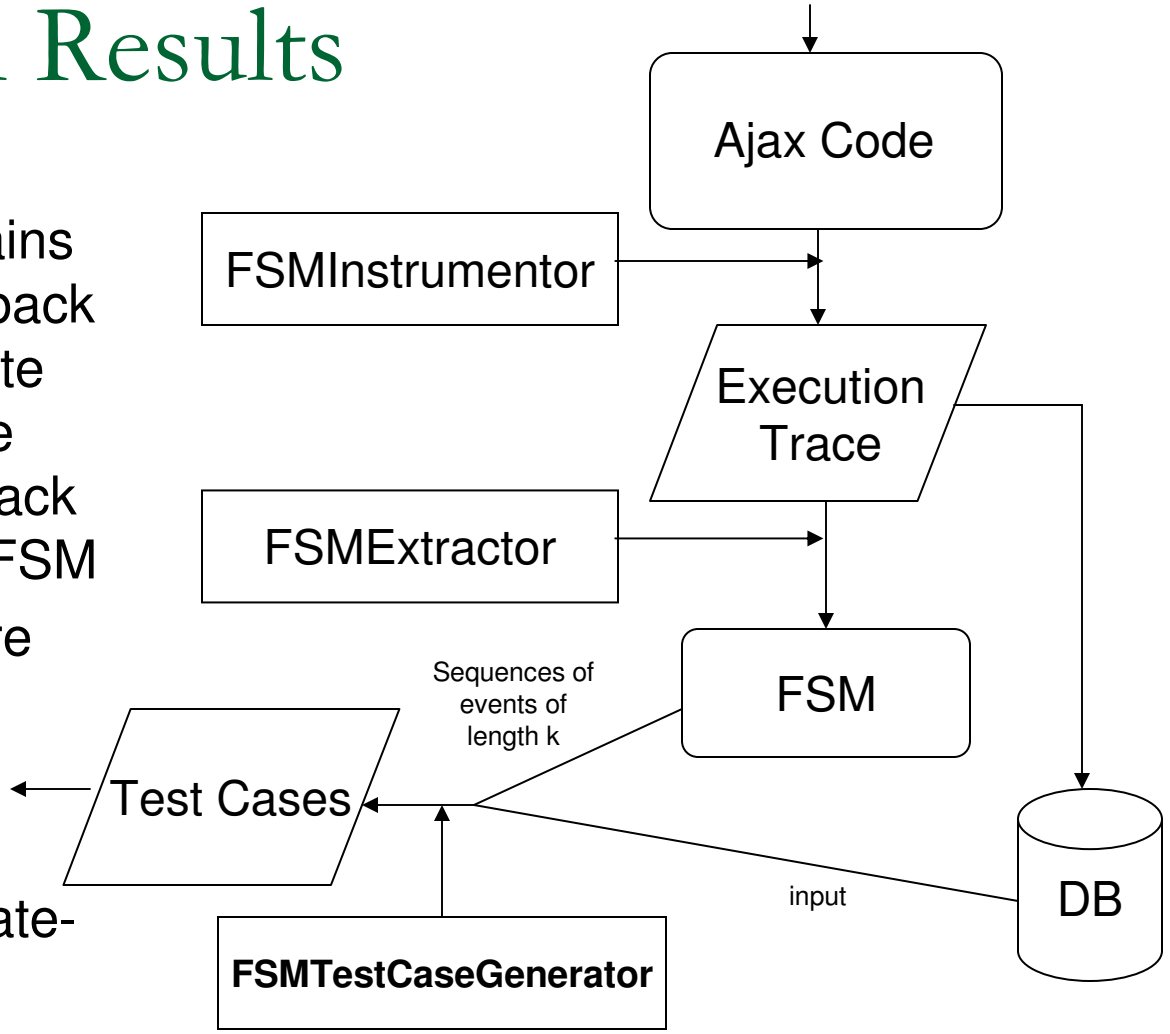
Experimental Results

- Goal:

  To study the advantages, effectiveness and effort involved in testing Ajax application taking into consideration semantic interactions, compared to Path Coverage Criteria
- Application under test: Tudu
- Tools
  - FSMInstrumentor
  - FSMExtractor
  - FSMTestCaseGenerator

Experimental Results

- Execution trace contains events triggering callback execution and concrete state of the Web page before and after callback execution, useful for FSM
- 21 different faults were considered for Tudu, hence 21 different versions of Tudu
- Generic as well as state-based faults



Experimental Results

- Size reduction achieved

	Test Cases		
Test Criterion	Sequences	Semantic Sequences	Ratio
Functionality coverage	27	*13*	*0.48*
FSM coverage	56	*13*	*0.23*
k=2	431	95	0.22
k=3	3616	623	0.17
k=4	29076	4428	0.15
k=5	223500	29641	0.13

Table 2. Semantic sequences and reduction

- k is length of the sequence
- Substantial test suite size reduction achieved for semantically interacting sequences

- Faults revealed

	Coverage		Semantic Sequences		
Id	Funct.	FSM	*k*=2	*k*=3	*k*=4
Generic Faults					
1	y	y	y	y	y
2	y	n	n	n	n
3	y	y	y	y	y
4	y	n	n	n	n
5	y	n	n	n	n
6	n	n	n	y	y
Tot	5	2	2	3	3
State-based Faults					
7	y	n	n	n	n
8	y	n	y	y	y
9	n	n	n	n	n
10	n	n	n	n	n
11	y	n	n	n	y
12	y	n	n	n	y
13	y	y	y	y	y
14	y	n	y	y	y
15	y	y	y	y	y
16	y	y	n	y	y
17	y	n	n	n	n
18	n	n	n	n	n
19	n	n	n	n	y
20	n	y	n	y	y
21	y	n	n	y	y
Tot	10	4	4	7	10

Table 3. Revealed faults

Sequences of 3 different GUI-events necessary to find the bug

- Functional coverage and semantic interactions appear to be complementary testing techniques (#7,#17 – #19,#20).

Experiment Results

- Traces needed for FSM Construction

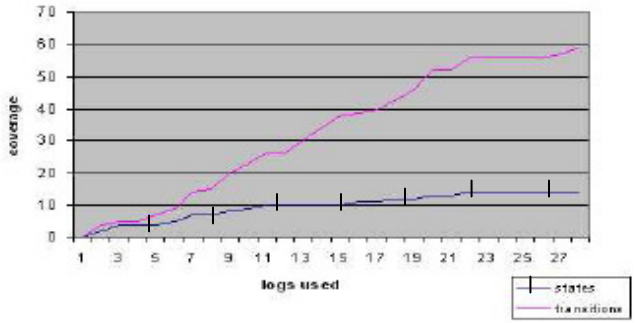


Figure 4. States and transitions added to FSM by number of traces

- Both the curves reach a plateau around trace number 22.
- Manual efforts increase when the number of traces are insufficient to build complete FSM

- Manual Refinement Steps
  - Refinement steps can be automated partially or completely

Conclusions

- FSM can be used to describe Ajax applications, whose states are abstraction of DOM of the page manipulated by Ajax code and transitions are callback executions triggered by asynchronous messages received from the web server.
- Quality of traces play an important role in constructing accurate FSM as well as the level of automation achieved.
- The fault exposing power of semantic sequences grows with their length
- The experimental results were specific to Tudu, hence difficult to generalize to other applications having different characteristics from Tudu.

Future Work

- Investigate techniques to support the generation of longer sequences
- Adding data flow analysis required for asynchronous warning generation in the developed tools which can be useful to get the knowledge about dependency between callbacks and DOM elements.

Thank you!

Difference between and Ajax and GUI applications

- Asynchronous communication is present
- Callbacks are activated both by user events and by server messages
- Interface is manipulated by Ajax code through the DOM

Glossary

- DOM
  - Document Object Model
  - A platform- and language-independent standard object model for representing HTML or XML and related formats
- Abstract States
  - Refer Adabu Tool