tesi di laurea

Testing di Rich Internet Application basato su analisi di sessioni utente

2008/2009

relatore

Ch.mo prof. Porfirio Tramontana

correlatore

Ch.mo ing. Domenico Amalfitano

candidato

Massimiliano Fattorusso

Matr. 885/252

Problematica affrontata:

Testing di Rich Internet Application (RIA) Ajax-based

Obiettivi:

- **Proporre una strategia di generazione di Test Suite a partire da user session data.**

- **Verificare automaticamente l'efficacia di una Test suite nell'individuare i difetti iniettati appositamente all'interno di una RIA**

Il processo di produzione di Test Suite da user session data

Mediante il tool di Reverse Engineering per RIA:
CreRIA

Vengono estratti i dati di input (es. text, checkbox) e gli eventi utente

A ciascun traccia d'esecuzione corrisponde un Test Case.

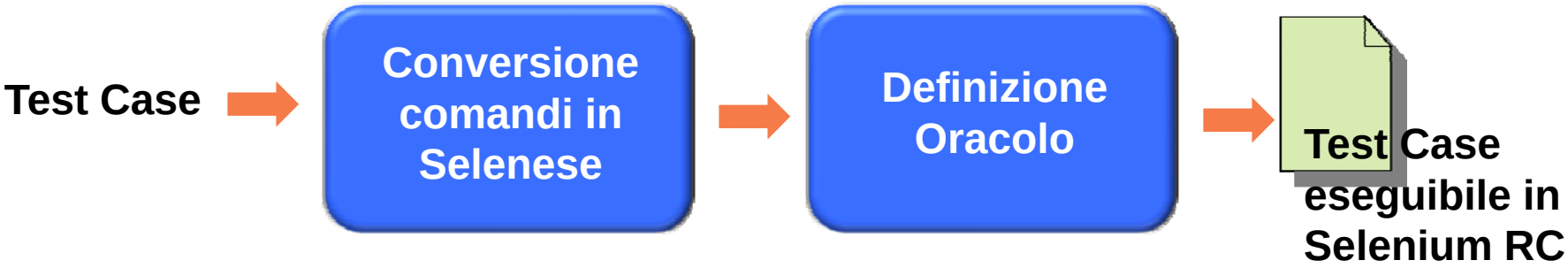
Caratterizzato da:

Stato Iniziale: l'interfaccia 'home' della RIA da analizzare

Precondizioni: lo stato delle risorse (ad es.: database) all'inizio dell'analisi

Problema: Automatizzare l'esecuzione dei Test Case

Soluzione: Eseguire i test in ambiente Selenium RC, che consente di avviare un browser ed effettuare i test in automatico.

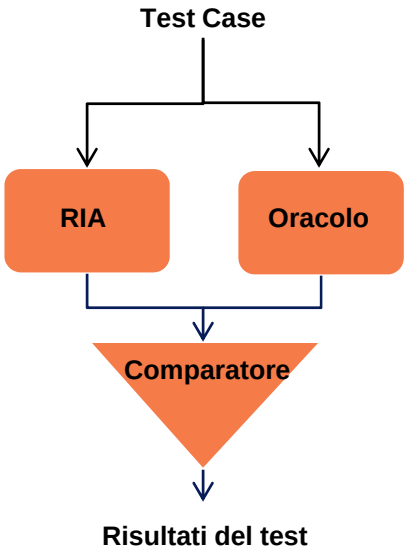


Linguaggio per impartire i comandi esecutivi ed i valori di input a Selenium RC

Es. click
clickAndWait
type

La sequenza di interfacce percorse durante l'esecuzione del test deve coincidere con la sequenza attesa.

E.s. la sequenza che si verificava sull'applicazione originale durante la raccolta delle tracce



La comparazione tra l'oracolo ed il sistema da testare avviene verificando opportuni asserts (asserzioni)

Le asserzioni sono verificate mediante metodi messi a disposizione dalle librerie di Selenium RC quali: assertEquals e assertTrue.

*Un'asserzione è vera se in seguito all'esecuzione di un evento, presente in un test case, l'interfaccia di destinazione è **equivalente**, secondo un criterio a quella attesa (contenuta nel l'insieme delle tracce raccolte).*

Due interfacce sono equivalenti se i widget di I1 sono inclusi in I2, e i widget di I2 sono inclusi in I1

C1
Per ogni widget:

- Tipo di event listener
- Nome degli event handler
- IndexedPath *(/html[2]/body[1]/div[3]/ul[1])*

C2
Per ogni widget attivo, visibile e abilitato:

- Tipo di event listener
- Nome degli event handler
- IndexedPath

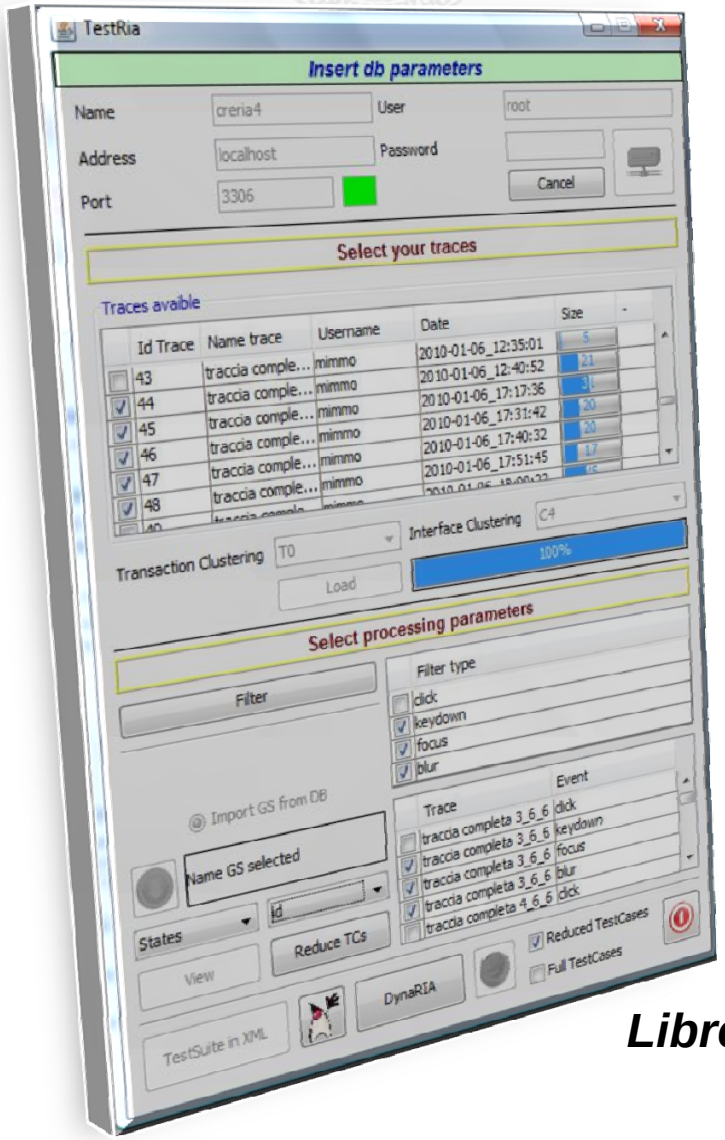
C3
Per ogni widget attivo, visibile e abilitato:

- UnindexedPath *(/html/body/div/ul)*

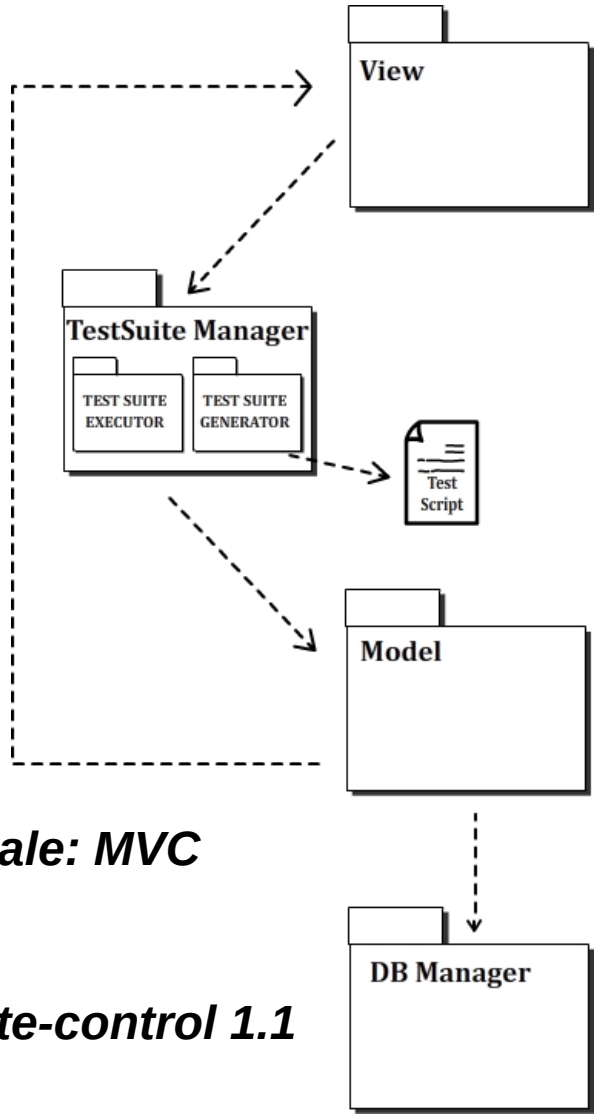
C4
Per ogni widget attivo, visibile e abilitato:

- Pathid *(html/body/div[@id='track']/ul/li/a)*

Criteri di equivalenza tra interfacce



Il tool TestRIA



Pattern architetturale: MVC

Linguaggio Java

Librerie Selenium-remote-control 1.1

Obiettivo della sperimentazione:

Valutare efficacia ed efficienza di una test suite nell'individuare i difetti iniettati appositamente all'interno di una RIA, al variare del criterio di equivalenza tra interfacce utilizzato per la produzione degli asserts.

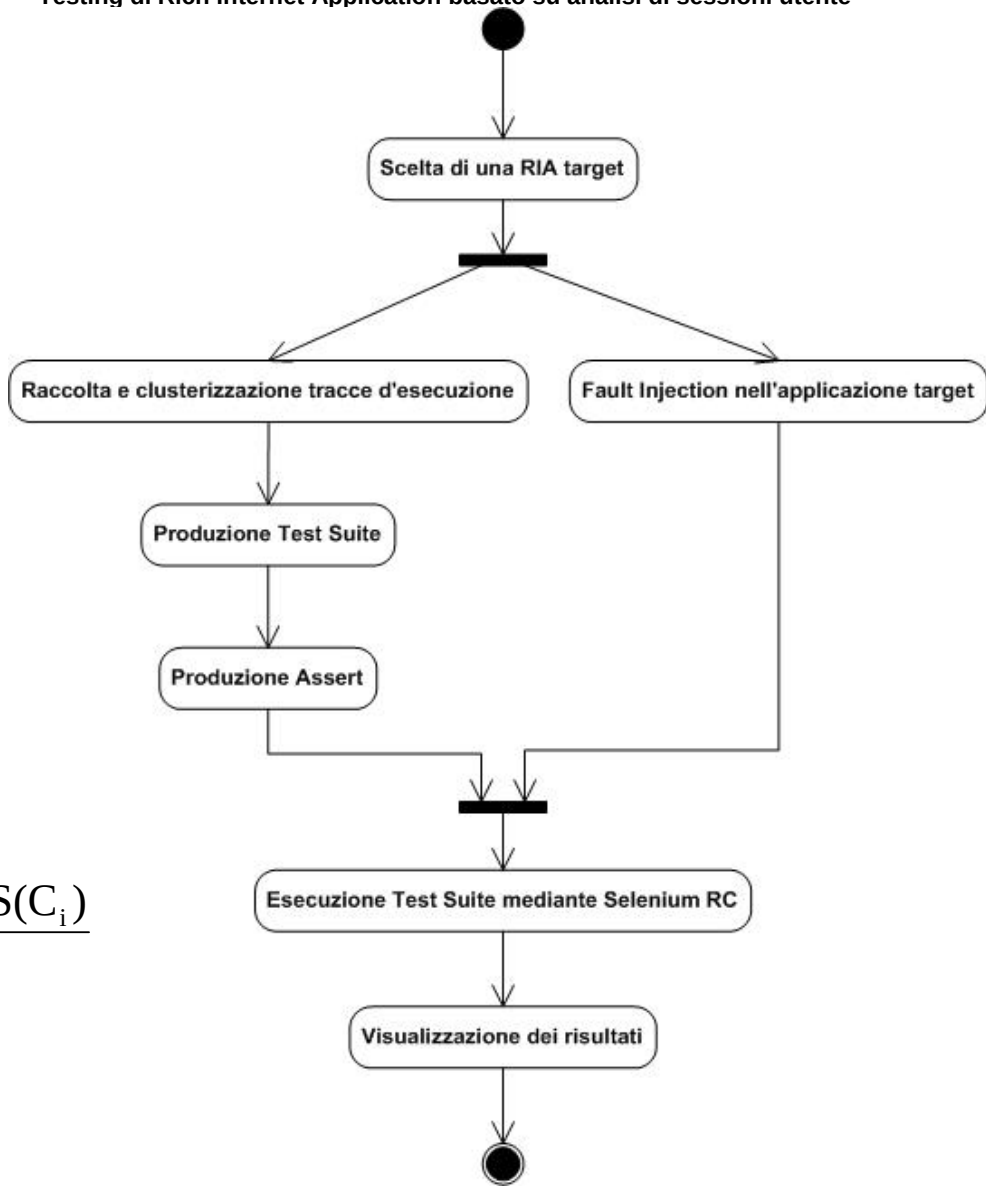
Metriche di valutazione:

$$M1(TS, C_i) = \frac{\#\,\text{failure scovati da } TS(C_i)}{\#\,\text{failure iniettati}}$$

$$M2(TS, C_i) = \frac{\#\,\text{media di failure scovati dai TC di } TS(C_i)}{\#\,\text{failure iniettati}}$$

Tempo d'esecuzione medio

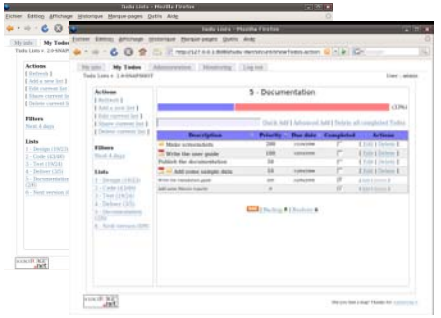
$$M3(TS, C_i) = tm_{TS(C_i)} = \frac{\sum_{i=1...n} t^{D_i}_{TS(C_i)}}{n}$$



RIA target:

Tudu Lists
Getting Things Done!
Tudu Lists v. 2.3

- Ajax-based, scritta in Java/JSP
- Open source
- *http://tudu.sourceforge.net*



Tracce d'esecuzione proposte:

- 33 tracce composte da 692 interfacce, raccolte da un alpha-tester
- Esercitano i casi d'uso conosciuti
- Hanno tutte le stesse condizioni di partenza
  - DB di Tudu vuoto
  - Home page

Difetti iniettati:

10 versioni difettate della RIA
2 tipologie:

State-based
- ***Js event handler manipulation*** (D3,D4,D6)
- ***Not-Js event handler manipulation*** (D1,D2,D5,D7)

Input validation (***errori semantici***) (D8,D9,D10)

Criteri d'equivalenza tra interfacce considerati:

C1,C2,C3,C4

Risultati della sperimentazione

Test Suite

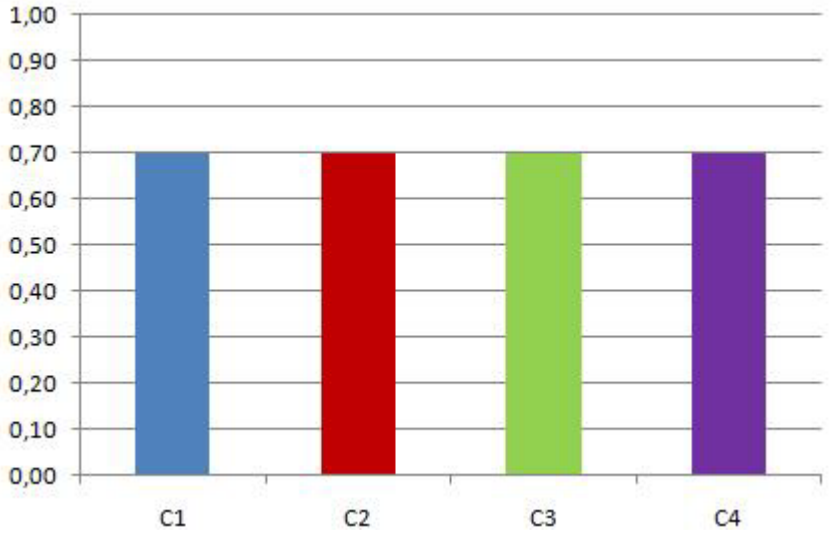
Failure:		C1	C2	C3	C4
State-based	D1	yes	yes	yes	yes
	D2	yes	yes	yes	yes
	D3	yes	yes	yes	yes
	D4	yes	yes	yes	yes
	D5	yes	yes	yes	yes
	D6	yes	yes	yes	yes
	D7	yes	yes	yes	yes
Input validation	D8	no	no	no	no
	D9	no	no	no	no
	D10	no	no	no	no

Yes= la test suite individua il malfunzionamento
No= la test suite non individua il malfunzionamento

I malfunzionamenti di tipo input validation non sono scovati.

L'efficacia della test suite nello scovare i malfunzionamenti non cambia al variare del criterio d'equivalenza scelto

$M1(TS, C_i)$



Analisi della percentuale dei test case che scovano i malfunzionamenti al variare del criterio d'equivalenza scelto

Difetto iniettato "Not-Js event handler manipulation" **(D1,D2,D5,D7)**

%C1 = %C2 = %C3 = %C4

La percentuale di test case in grado di scovare i malfunzionamenti è indipendente dal criterio usato

Difetto iniettato "Js event handler manipulation" **(D3,D4,D6)**

%C1 ≥ %C2 > [%C3 = %C4]

Con C1 e C2 si scova il malfunzionamento anche in test case non candidati , ovvero test case che non scatenano eventi sulle interfacce coinvolte nei failure.

Con C1 la percentuale di test case che scovano malfunzionamenti aumenta ulteriormente poiché sono analizzati anche widget non attivi.

Usando il criterio C1 c'è una maggiore percentuale di test case candidati a scovare malfunzionamenti.

Test Suite

Failure:		C1	C2	C3	C4
State-based	D1	72,7%	72,7%	72,7%	72,7%
	D2	97,0%	97,0%	97,0%	97,0%
	D3	90,9%	90,9%	75,6%	75,6%
	D4	75,6%	35,2%	27,2%	27,2%
	D5	18,2%	18,2%	18,2%	18,2%
	D6	69,7%	54,5%	30,3%	30,3%
	D7	24,2%	24,2%	24,2%	24,2%
Generic	D8	0,0%	0,0%	0,0%	0,0%
	D9	0,0%	0,0%	0,0%	0,0%
	D10	0,0%	0,0%	0,0%	0,0%

Percentuale di test case della test suite che scovano il malfunzionamento

Problema:

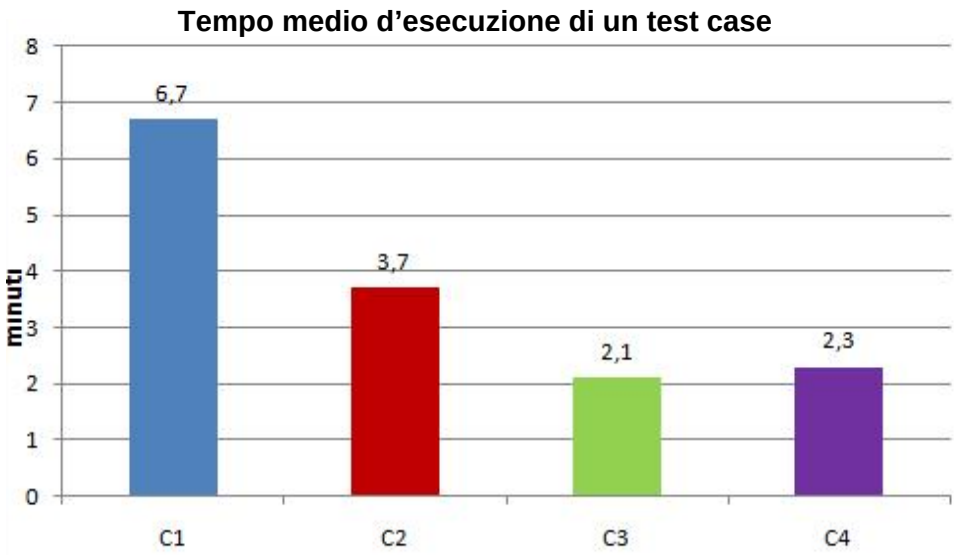
L'esecuzione dei test cases, ma più in generale dell'intera test suite, usando il criterio d'equivalenza C1 per la produzione degli asserts, presenta dei tempi quasi doppi rispetto agli altri criteri.

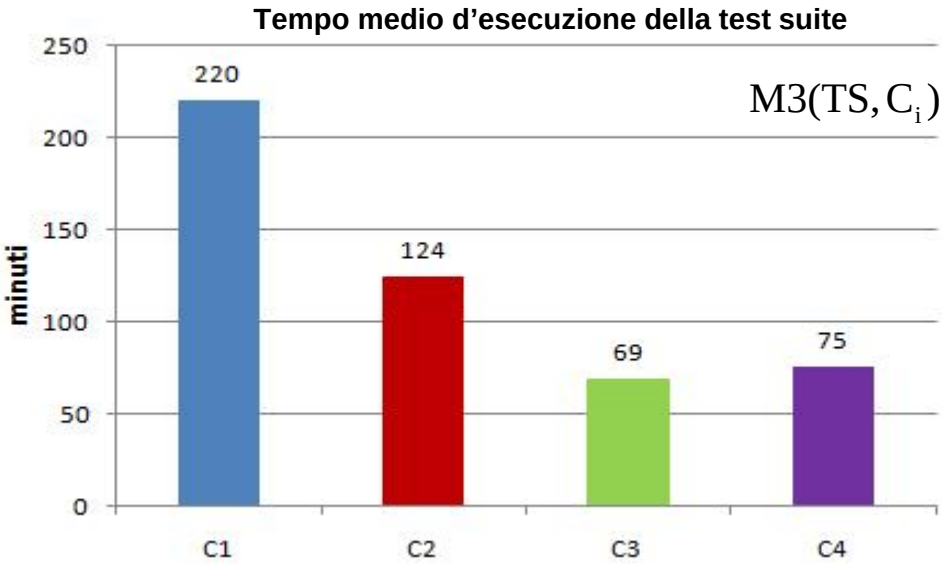
Causa:

Numerosi asserts da verificare a causa dell'elevata quantità di widgets (anche non attivi) da controllare

Soluzione:

Ridurre la cardinalità della test suite cercando di eliminare possibili test cases ridondanti, utilizzando un algoritmo di minimizzazione.





Conclusioni

Limitatamente alla sperimentazione effettuata, tutti i criteri hanno portato allo stesso numero di difetti trovati. I criteri come C1 si sono rivelati più lenti ma hanno rilevato più frequentemente i difetti (ma non più difetti degli altri).

Sviluppi futuri

Estendere la sperimentazione

- Su diverse RIA open source
- Usando tracce d'esecuzione raccolte da crawler
- Iniettando ulteriori difetti facendo riferimento a malfunzionamenti reali

Minimizzazione della Test Suite:

- Ci si attende che l'utilizzo di criteri di minimizzazione di test case (con diversi criteri di copertura) possano portare ad una riduzione del numero di test case da eseguire, e quindi ad una riduzione dei tempi a parità di efficacia