UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II
Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

tesi di laurea

Un sistema per il Crawling di Rich Internet Applications

Anno Accademico 2008/2009

relatore

Ch.ma prof. Anna Rita Fasolino

correlatore

Ch.mo prof. Porfirio Tramontana

candidato

Vincenzo Morra

Matr. 885/403

Obiettivi e Motivazioni

- **Rich Internet Applications:**
  - **Applicazioni guidate dagli eventi, che si prestano ad essere modellate con una FSM;**
  - **Generate Dinamicamente, per cui è necessario adottare di tecniche di analisi dinamica per lo studio del comportamento della RIA;**
  - **Comunicazione asincrona.**
- **Sfide dell'ingegneria del software:**
  - **Applicazione delle tecniche di testing;**
  - **Applicazione delle tecniche di Reverse Engineering;**
  - **Scoperta automatica della struttura e dei contenuti delle RIA.**
- **Sviluppo di un crawler per l'esplorazione di RIA**

Il Crawler

- **Esplorazione automatica di RIA.**
- **Generazione automatica di tracce di esecuzione della RIA, del tutto equivalenti alle tracce generate manualmente da sessioni utente.**
- **Generazione automatica della FSM rappresentante il comportamento della GUI della RIA**
  - **il modello scelto è la FSM poiché è il più adatto per modellare i sistemi interattivi guidati dagli eventi.**

Algoritmo di Crawling

- **Visita in ampiezza;**
- **Ad ogni evento, ovvero interazione con un widget, è associata una transizione;**
- **Una traccia viene chiusa quando viene incontrata un'interfaccia già intercettata in precedenza;**
- **Per poter interagire con un widget bisogna ogni volta ricreare le condizioni nelle quali è stato incontrato originariamente.**

```
1.  Procedure start(URLHP,CX,ModExec)
2.  Browser = InitRobotBrowser()
3.  SMM = InitStateMachineManager()
4.  NextStatesToVisit = InitNextStatesToVisitList()
5.  for widgetset  widgetsettocrawl
6.      Browser.Crawl(state,widgetset)
7.  end for
8.  while !NextstatesToVisit.isEmpty()
9.      nextstate=NextStatesToVisit.poll()
10.     If GoToState(nextstate) then
11.         for widgetset  widgetsettocrawl
12.             Browser.Crawl(nextstate,widgetset)
13.         end for
14.     end if
15. end while
16. end procedure

17. procedure Crawl(state,widgetset)
18. Widgets = SMM.getWidgets(widgetset)
19. while !Widgets.isEmpty()
20.     Widget = Widget.poll()
21.     if GoToStateLocal(state) then
22.     WidgetInteract= RetreiveWidget(Widget.XPath)
23.     newinterface =
HandleEvent(Widget.event,WidgetInteract)
24.     isnewstate=HandleNewState(newinterface)
25.     if isnewstate then
26.         NextStatesToVisit.add(newinterface)
27.     end if
28.   end if
29. end while
30. end procedure
```

Problematiche del Crawler

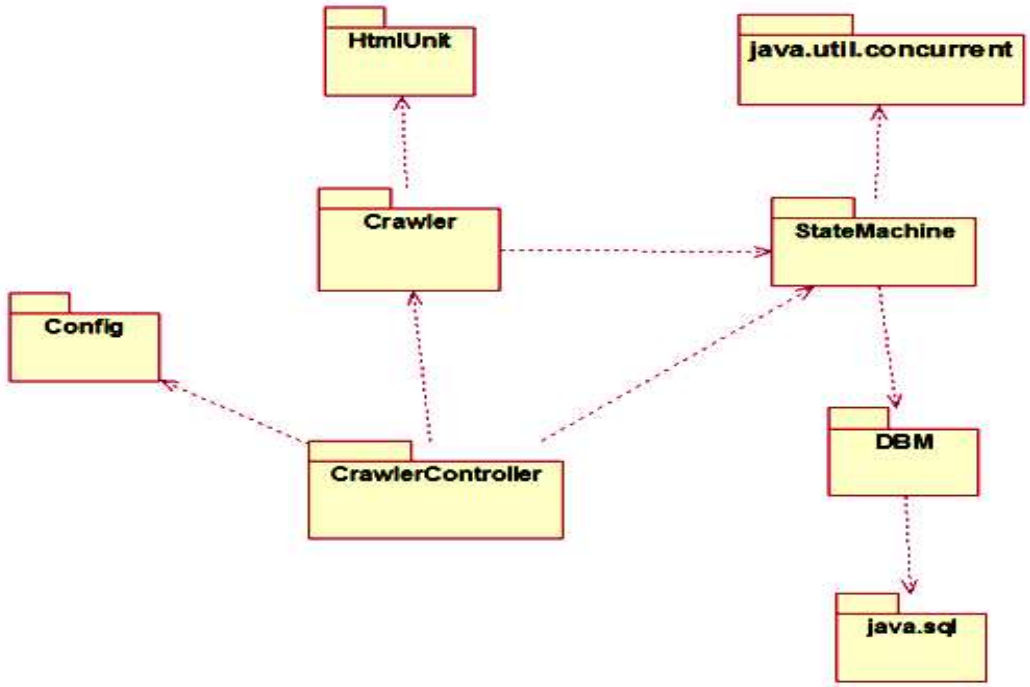
- **Esplosione degli stati durante l'esplorazione automatica della RIA:**
  - **Il Crawler risolve tale problematica utilizzando una euristica che valuti l'equivalenza funzionale tra interfacce per terminare la cattura di una traccia di esecuzione. Ogni volta che si ritorna ad un'interfaccia equivalente ad una già esplorata viene interrotta l'esplorazione della traccia di esecuzione.**
- **Ripristino delle precondizioni necessarie all'esplorazione della RIA:**
  - **Affinché le tracce generate e la FSM siano coerenti bisogna ripristinare ad ogni esecuzione lo stato della RIA, dove per stato si intendono le risorse e le variabili di sessione gestite dal web server.**

Problematiche del Crawler

- **Esecuzione del codice Javascript:**
  - **Il Crawler, dopo l'esecuzione di un evento, per poter verificare se lo stato prodotto in seguito all'esecuzione dell'evento sia equivalente ad uno stato precedentemente visitato deve attendere un tempo sufficiente affinchè possa considerare stabile l'interfaccia ottenuta, ovvero per ricevere la risposta, eseguire il codice Javascript e attendere l'iniezione nel DOM delle modifiche apportate.**

- **Analizzare RIA con stato:**
  - **Il Crawler ha il problema di mantenere lo stato delle risorse tra una esecuzione e l'altra della stessa interfaccia. Si risolve tale problematica introducendo una particolare modalità di esecuzione, che ha inizio con un particolare stato definito come parametro di ingresso e precedentemente raggiunto in una traccia. Il rilascio delle risorse acquisite, quindi l'abbandono dello stato, avviene con una predeterminata sequenza di uscita oppure riavviando il web server su cui è in esecuzione la RIA**

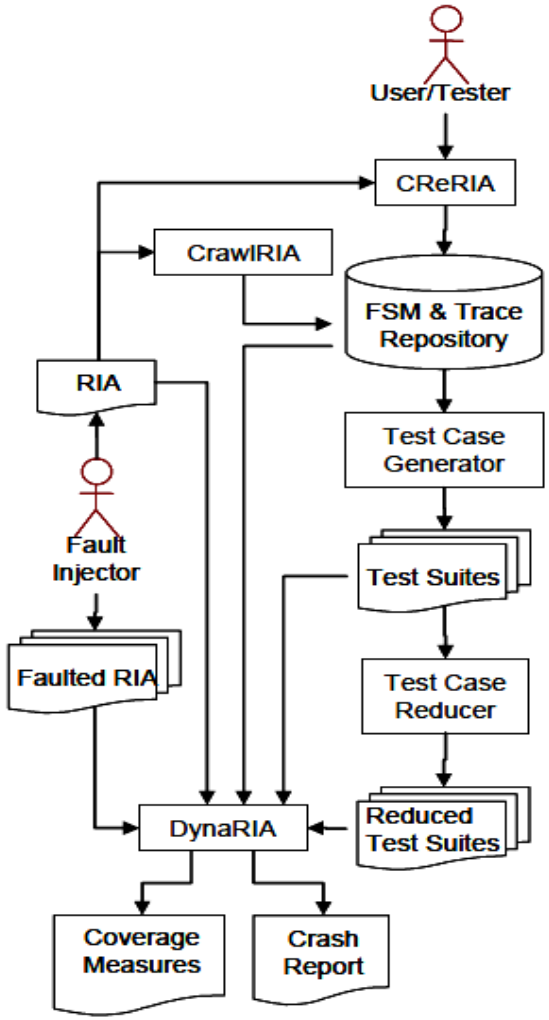
Architettura del Crawler



- HtmlUnit è un “GUI-Less browser for Java programs”; nello sviluppo del Crawler è stato utlizzato per interagire con le RIA.

Applicazione del Crawler al processo di Testing

- Il processo di testing è stato proposto dal gruppo di ingegneria del software del DIS ed è in fase sperimentale;
- CReRIA: tool che permette ad un utente esperto di interagire con la RIA e memorizzare le tracce di esecuzione ed astrarre la FSM della RIA;
- Test Case Reducer: tool che genera test suite equivalenti alle test suite di ingresso adottando differenti criteri di riduzione;
- DynaRIA: tool che esegue le test suite di ingresso restituendo le misure dei criteri di copertura e crash che si sono verificati durante l'esecuzione dei test case.



Caso di studio: TUDU

- Criteri di copertura da applicare alle test suite prodotte:
  - Copertura degli stati della FSM;
  - Copertura delle transizioni della FSM;
  - Copertura delle funzioni Javascript;
  - Copertura delle LOC Javascript;
  - Fault detection;
  - Dimensione della test suite;
  - Dimensione degli eventi della test suite
- 3 tecniche di generazione di test suite:
  - G1: Sessioni utente;
  - G2: Automatica tramite il Crawler;
  - L'unione delle 2 precedenti.
- 4 tecniche di riduzione di test suite:
  - M1: riduzione che tiene conto degli stati della FSM della RIA;
  - M2: riduzione che tiene conto delle transizioni della FSM della RIA;
  - M3: riduzione che tiene conto delle funzioni Javascript della RIA;
  - M4: nessuna riduzione.

Caso di studio: TUDU

- **Dati relativi alle test suite prodotte da G1:**

	G1	G1M1	G1M2	G1M3
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

- **Dati relativi alle test suite prodotte da G2:**

	G2	G2M1	G2M2	G2M3
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

- **Il tempo di esplorazione è di circa 10 ore;**
- **203 tracce generate;**
- **La lunghezza massima delle tracce è di 6 stati**
- **La lunghezza media di una traccia è di circa 4 stati;**
- **754 interfacce sono state intercettate.**

Caso di studio: TUDU

- **Dati relativi alle test suite prodotte dall'unione di G1 e G2:**

	G1∪G2	(G1∪G2) M1	(G1∪G2) M2	(G1∪G2) 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

- **Considerazioni:**
  - **Le test suite prodotte da G1 offrono una copertura degli stati e dei fault maggiori rispetto alle test suite prodotte da G2, ciò è dovuto al criterio di terminazione di una traccia presente nell'algoritmo di crawling;**
  - **La copertura delle funzioni e dei LOC Javascript sembra essere inefficiente, ma bisogna tener conto che tali percentuali sono dovute all'inutilizzo nella versione attuale di TUDU di molte funzioni Javascript presenti nelle librerie esterne comunque presenti in TUDU. Quindi da tali percentuali non si può dedurre che le test suite siano inefficienti.**

Caso di studio: TUDU

- **Considerazioni:**
  - **Per quanto riguarda la copertura delle transizioni, G1 risulta migliore a G2, in particolare G1 offre una copertura del 91,8% rispetto al 57,4% di G2, questo risultato è indice dell'incapacità del Crawler di simulare il comportamento di un utente esperto nell'interazione con la RIA;**
  - **In generale G1 offre una copertura maggiore rispetto a G2 rispetto a molto criteri di copertura, però vi sono alcuni criteri in cui G2 una maggiore efficacia; in particolare G2 offre test suite più grandi, ovvero con più test case, e una maggiore copertura degli eventi scatenabili sulla RIA, ovvero delle possibili interazioni con la RIA;**
  - **G1 ha dei costi molto maggiori rispetto a G2 in quanto prevede l'intervento di un ingegnere del software esperto della RIA per la cattura delle tracce di esecuzione, cosa che non avviene in G2 in cui le tracce sono generate automaticamente dal Crawler;**
  - **L'unione di G1 e G2 consegue i risultati migliori nella copertura dei criteri, inoltre il costo è equivalente al costo di G1.**

Conclusioni e Sviluppi Futuri

- **Conclusioni:**
  - **Lo sviluppo del Crawler è stato eseguito con successo ed i risultati ottenuti dalle sperimentazioni rispecchiano le ipotesi effettuate in fase di analisi.**
- **Sviluppi Futuri:**
  - **Estendere il crawling anche alle tecnologie Flash e Silverlight;**
  - **Sviluppare nuove strategie di crawling, sia in termini di algoritmo di visita che di interazioni con le RIA;**
  - **Applicazione del Crawler al testing regressione;**
  - **Esecuzione parallela del Crawler.**