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

tesi di laurea

Reverse Engineering di Rich Internet Applications basate su AJAX

Anno Accademico: 2006/2007

relatore
Ch.mo prof. Anna Rita Fasolino

correlatore
Ch.mo prof. Porfirio Tramontana

candidato
Domenico Amalfitano
Matr.41/3793

Scopo del lavoro di tesi

- Contesto: Reverse Engineering di RIA basate su AJAX.
- Proporre uno o più modelli per descrivere il comportamento di una Rich Internet Application (RIA) basata su AJAX, per supportare processi di testing, manutenzione, migrazione.
- Proporre un processo di reverse engineering per la ricostruzione dei modelli proposti.
- Sviluppare un tool capace di automatizzare il processo di reverse engineering.

Rich Internet Application

Differenze con le Web Application Tradizionali

Web Applications Tradizionali

- **Natura click – and – wait**
  - **L'utente resta in attesa della risposta del server, dopo aver effettuato una richiesta.**
  - **Richieste sincrone.**
- **Quasi tutto il processing necessario viene effettuato sul server.**
- **I dati visualizzati devono essere sistemati all'interno del formato della pagina HTML prima che possano essere presentati all'utente sulla macchina client. Per questo motivo, le pagine Web devono essere ricaricate ogni volta che un utente debba visualizzare dei set differenti di dati.**
- **Poca interattività con l'utente finale.**

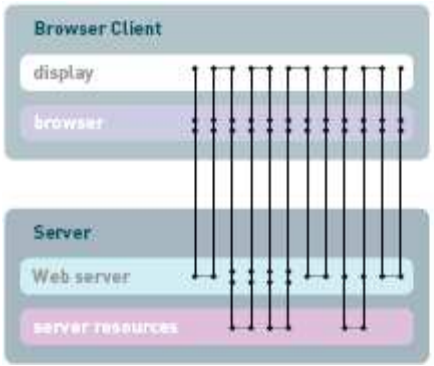
RIA

- **Elimina la natura click – and – wait.**
- **Quasi tutto il processing viene trasferito sul client.**
- **Scambio di piccole quantità di dati con il server per l'aggiornamento delle interfacce utente.**
- **Possibilità di effettuare richieste asincrone, non bloccanti.**
- **Elevata interattività con l'utente finale.**

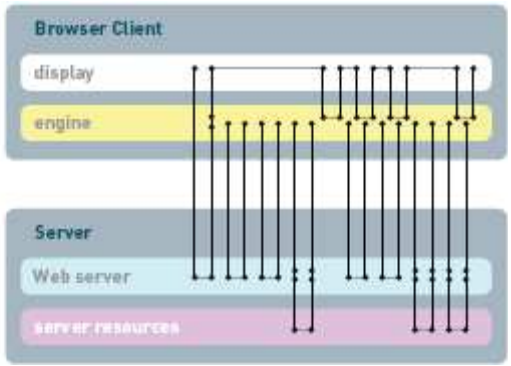
Rich Internet Application

Client – Side Engine

- **Tutte le RIA introducono un layer logico detto Client-Side engine.**
- **Questo motore può avere implementazioni differenti.**
- **È scaricato all'inizio di una sessione.**
- **Permette alla RIA di avere delle caratteristiche e funzionalità simili alle applicazioni desktop.**
- **L'informazione può essere precaricata dal server in anticipo, prima dell'*input* da parte dell'utente.**
- **In risposta ad un *input*, la visualizzazione può essere aggiornata in maniera incrementale, invece che in una volta sola.**
- ***Input* multipli da parte dell'utente possono essere validati e accumulati sul client prima di essere inviati al server.**



Modello di comunicazione di una Web application tradizionale



Modello di comunicazione di una RIA

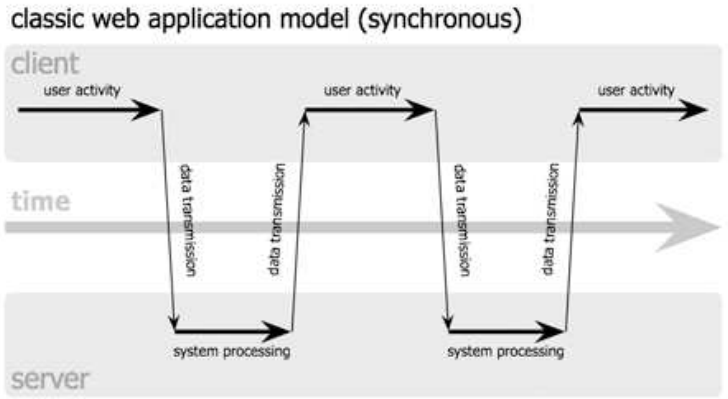
- **Le risposte ad alcuni *input* dell'utente possono essere generate senza la comunicazione con il server.**
- **I processi gestiti in precedenza dai server possono essere scaricati sul *client desktop*.**
- **Una richiesta al server può essere non bloccante per il client, l'utente può continuare a utilizzare l'applicazione.**

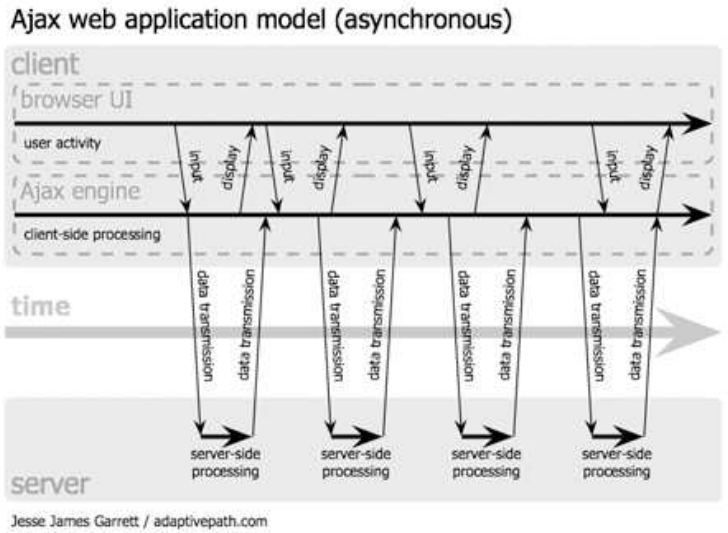
AJAX

- AJAX acronimo di Asynchronous JavaScript and XML.
- Tecnica e non tecnologia di sviluppo Web per creare applicazioni Web interattive.
- Un insieme di tecnologie, ognuna delle quali apporta le proprie prerogative:
  - HTML e CSS per il *markup* e lo stile della pagina.
  - Il DOM per la visualizzazione e l'interazione dinamica dell'utente con l'interfaccia della Web application.
  - L'oggetto XMLHttpRequest per l'interscambio asincrono dei dati tra il browser e il Web server.
  - XML per lo scambio e la manipolazione dei dati.
  - JavaScript per legare tutto insieme.
- Oltre ad XML esistono altri formati per lo scambio di dati, ad esempio testo semplice, HTML preformattato, JSON.
- AJAX supporta il modello di comunicazione precedentemente presentato.

AJAX

- Il client – engine per RIA basate su AJAX è l'AJAX Engine.
- Scritto in JavaScript.
- Permette interazione asincrona utente – applicazione, indipendente dalla comunicazione col server.
- Le risposte che non necessitano di risposta da server vengono gestite direttamente dal motore.
- Richiede al server solo i dati necessari, c'è una piccola quantità di informazione scambiata tra client e server.
- La pagina è aggiornata dinamicamente anziché completamente caricata.



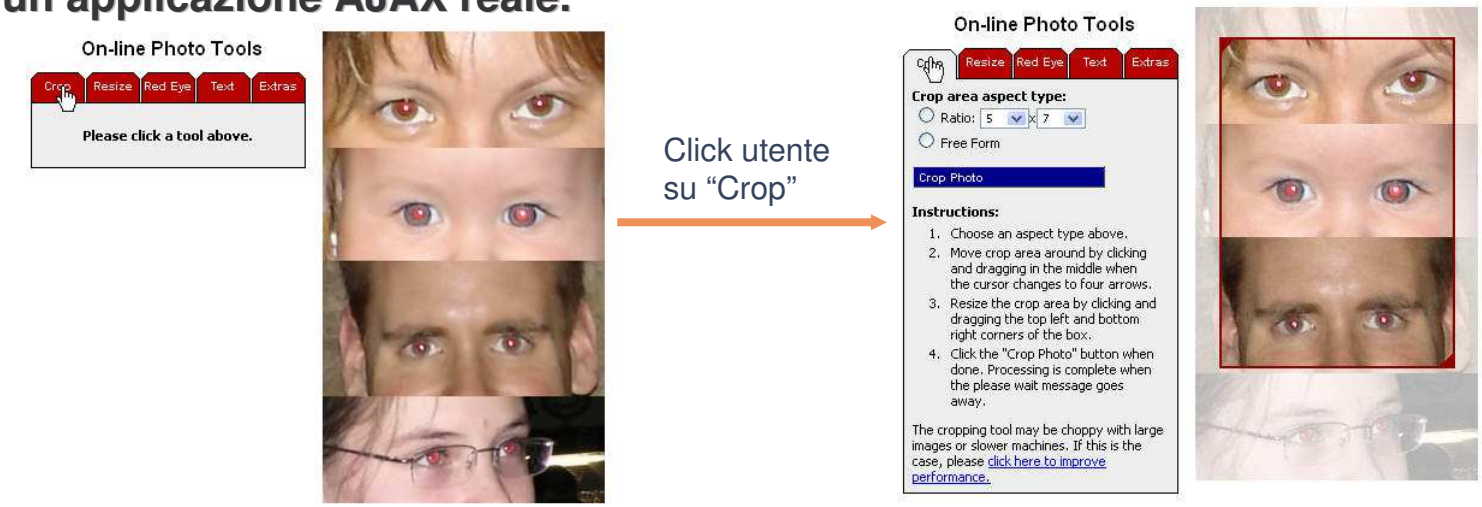


Modellazione

- AJAX ha cambiato il modo di sviluppare le applicazioni. Invece di pensare in termini di sequenze di pagine Web, gli sviluppatori possono programmare le proprie applicazioni sfruttando una singola interfaccia utente (UI) che viene aggiornata sempre nella stessa pagina Web.
- La singola pagina Web, che è in effetti l'interfaccia utente dell'applicazione, è composta da componenti individuali che possono essere inseriti e/o aggiornati e/o rimossi in maniera indipendente, in questo modo l'intera pagina non deve essere ricaricata dopo ogni azione da parte dell'utente che richiede un aggiornamento dell'interfaccia.
- Il problema trattato è quello di descrivere il comportamento dinamico dell'interfaccia della RIA.
- Modellare tale comportamento mediante FSM, dove ogni stato caratterizza l'interfaccia utente dell'applicazione.
- Le FSM sono definite formalmente da una quintupla M=(S,I,O,T,Φ), dove:
  - S, I, O sono rispettivamente gli insiemi finiti degli stati, degli ingressi e delle uscite;
  - S_0 è lo stato iniziale;
  - T è la funzione di transizione $S \times I \rightarrow S$ che specifica lo stato prossimo come una funzione dello stato corrente e dell'ingresso;
  - Φ è la funzione di uscita $S \times I \rightarrow O$ che specifica l'uscita risultante da una transizione.

Adattamento delle FSM alle RIA

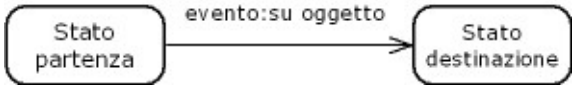
- **Per modellare il comportamento dinamico di una RIA basata su AJAX utilizzando una FSM bisogna mettersi dal punto di vista dell'*end user:***
  - **Le transizioni sono scatenate da eventi dovuti all'interazione che l'utente ha con l'interfaccia.**
  - **Gli eventi sono scatenati dall'interazione dell'utente con i componenti che definiscono l'interfaccia.**
  - **La gestione di un evento può, o meno, dar luogo ad una variazione di stato.**
  - **Ogni evento che causa una variazione dell'interfaccia in termini di inserimento e/o cancellazione di *widget* innesca sempre una transizione di stato.**
  - **Ogni stato della FSM rappresenta l'interfaccia utente dell'applicazione, trattandosi di RIA, l'interfaccia non è altro che una pagina Web.**
  - **La pagina Web che si interfaccia con l'utente finale ha sempre lo stesso indirizzo *url*, quello che cambia dinamicamente è il suo contenuto, variando soltanto alcuni componenti senza ricaricare l'intera pagina.**
- **Esempio di variazione dell'interfaccia a seguito di un evento utente per un'applicazione AJAX reale.**



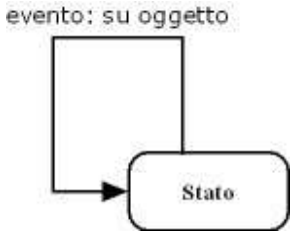
Adattamento delle FSM alle RIA - 2

- **Si tracciano delle linee guida da seguire per poter descrivere la dinamica dell'interfaccia utente tramite *State Diagram*:**
  - **Lo stato iniziale S_0 è la pagina iniziale della RIA. In pratica è la prima pagina a cui si accede attraverso l'URL, del tipo www.indirizzo_web_application.app:**

  - **Le transizioni tra gli stati avvengono solo ed esclusivamente a seguito di eventi; si evidenziano lo stato di partenza e lo stato di destinazione e la transizione che avviene a seguito dell'evento attivato su un particolare oggetto:**

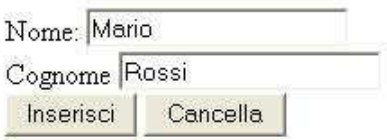


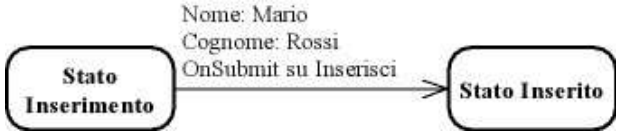
  - **Ci possono essere eventi che non causano transizione di stato oppure che causano una transizione tra stati equivalenti:**



Adattamento delle FSM alle RIA - 3

- **Le transizioni sono causate sempre da eventi preceduti o meno da un insieme, eventualmente vuoto, di valori assunti da uno o più elementi di *Input* della pagina:**



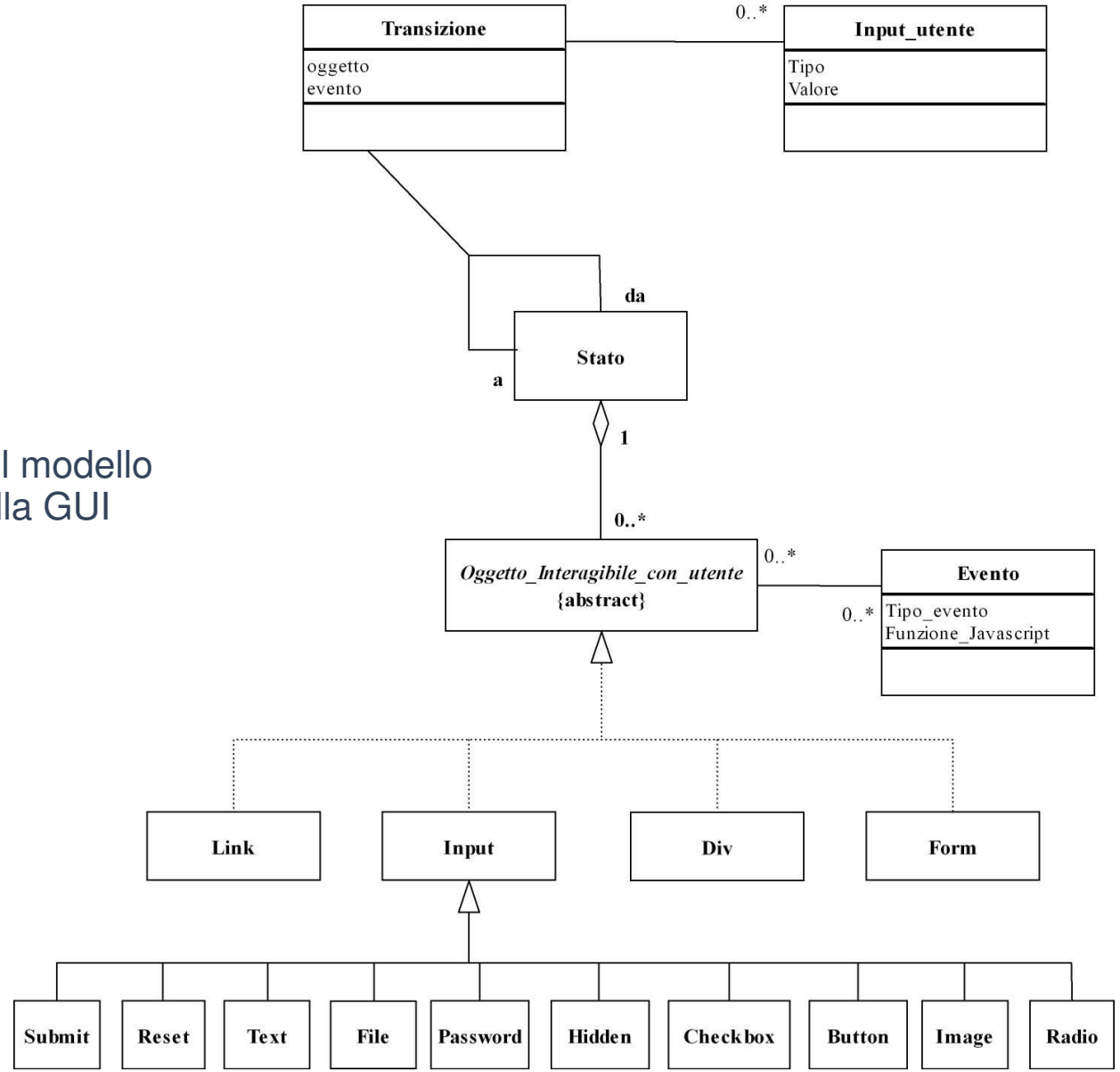


- **Uno stato che non ha eventi scatenabili può essere considerato come stato finale:**

- **Non sono presenti uscite, in questo tipo di FSM c'è solo transizione tra gli stati.**
- **Lo state diagram che dà una visione immediata e intuitiva di come evolvono le interfacce utente, da solo non è sufficiente a descrivere completamente il comportamento di una RIA; bisogna:**
  - **Modellare la struttura di ogni singolo stato.**
  - **Descrivere in maniera più precisa le transizioni che avvengono tra due stati successivi.**

Modellazione del Comportamento Dinamico della RIA - 1

Class Diagram del modello della dinamica della GUI

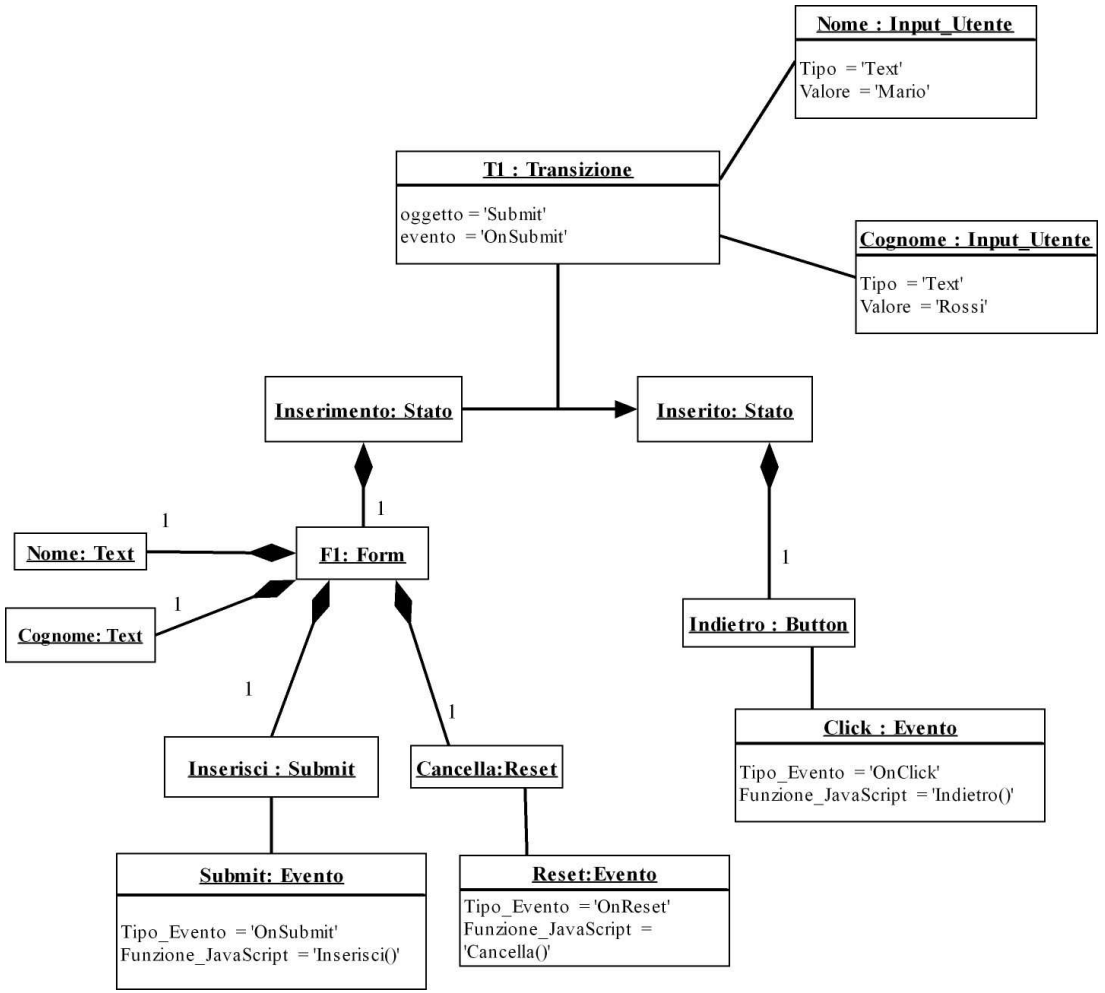


Modellazione del Comportamento Dinamico della RIA - 2

- Si introduce un class diagram che dà una descrizione completa del comportamento dell'applicazione, raggruppando sia la descrizione statica della struttura di ogni singolo stato della FSM e sia la descrizione del comportamento dinamico dell'interfaccia della RIA.
- Ogni stato rappresenta la specifica interfaccia utente che nel caso delle RIA è una pagina Web.
- Ogni stato a sua volta è composto da oggetti del DOM di cui si considerano soltanto gli oggetti che possono interagire con l'utente, a questi oggetti possono essere associati degli eventi di cui si considera il tipo dell'evento e la relativa funzione JavaScript che gestisce l'evento.
- Degli oggetti che possono realizzare la classe astratta degli oggetti che hanno la capacità di interagire con l'*end user* sono stati presi in considerazione quelli che al momento possono interagire con l'utente finale mediante *mouse* o *keyboard*.
- Se due stati hanno gli stessi oggetti con cui l'utente può interagire e se a questi oggetti sono abbinati eventi con identiche funzioni JavaScript associate, allora con buona approssimazione gli stati che descrivono l'interfaccia sono equivalenti se non addirittura identici.
- Due stati sono associati mediante una transizione che è caratterizzata dall'evento scatenante e l'oggetto su cui è stato scatenato.
- Ad una transizione possono essere associati eventuali valori di input inseriti da parte dell'utente.

Esempio di Transizione tra due stati

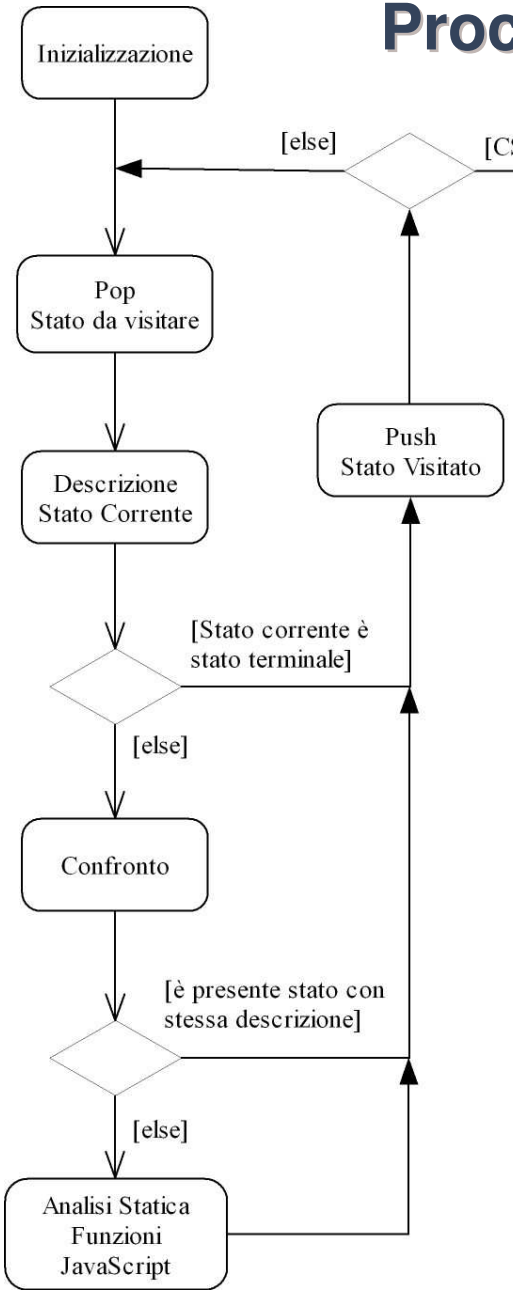
Modellazione tramite Object Diagram della transazione



Processo di Reverse Engineering

- Lo scopo del processo di reverse engineering è quello di astrarre i due modelli introdotti precedentemente.
- Si pongono le seguenti ipotesi:
  - Le Rich Internet Application che vengono prese in considerazione sono state costruite sfruttando esclusivamente AJAX; in particolare dal lato client si ha solo HTML e JavaScript, dal lato server si può avere qualsiasi tecnologia.
  - Poiché nelle RIA basate su AJAX il codice JavaScript viene completamente precaricato all'avvio dell'applicazione, si può supporre che ogni funzione JavaScript, con un dato nome, non subisca variazioni di codice durante l'esecuzione dell'applicazione.
  - Ci si pone dal punto di vista dell'utente finale; non si ha, quindi, accesso a tutto il codice sorgente dell'applicazione, in particolare agli script lato server, ma soltanto al codice HTML + JavaScript che definisce l'interfaccia utente. Il primo accesso che si ha è alla pagina iniziale della RIA.
- L'approccio seguito, data la natura delle Rich Internet Applications, è stato quello di integrare sia tecniche di analisi statica che dinamica.
  - L'analisi statica ha il duplice compito di analizzare e descrivere puntualmente ogni stato dell'interfaccia attraverso gli oggetti del DOM di cui è composto e tutte le funzioni associate agli eventi degli oggetti dell'interfaccia al fine di individuare quelle funzioni capaci di scatenare transizioni di stato.
  - L'analisi dinamica deve consentire la generazione delle possibili sequenze di interfacce.

Processo di Reverse Engineering - 2



Il processo di *reverse engineering* parte dalla pagina iniziale della RIA con lo scopo di individuare tutti gli stati potenzialmente raggiungibili dall'applicazione man mano che sono scatenati gli eventi – utente sulle GUI. Durante la sua esecuzione ci saranno degli stati, che rappresentano le interfacce utente, etichettati ed indicati o come "visitati" o "da visitare".
Le strutture dati utilizzati durante l'esecuzione del processo sono:

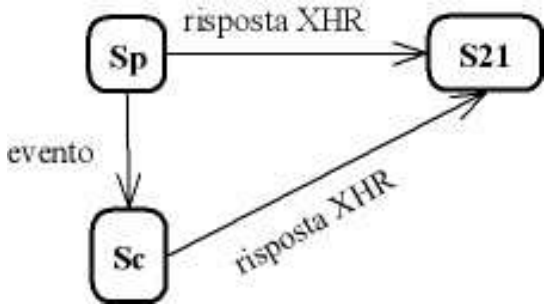
- Una coda degli stati da visitare definita come CSDV
- Una coda degli stati visitati definita come CSV
- Ogni stato è descritto in forma tabellare.
- È presente un'altra tabella, unica per tutto il processo, degli stati prossimi

Dopo una preliminare fase di "Inizializzazione" in cui si inizializza CSV come coda vuota e CSDV come coda contenente il solo stato iniziale, il processo eseguirà iterativamente i seguenti passi fondamentali, fin quando la coda degli stati da visitare non è vuota:

- Pop - Si preleva il primo stato da visitare dalla CSDV. Si porta effettivamente la RIA nello stato da visitare che diventa lo stato corrente.
- Descrizione - Si effettua una descrizione dell'interfaccia dello stato corrente. Verifica se è uno stato terminale.
- Confronto - Si controlla se tra gli stati già visitati in precedenza è presente uno stato con la stessa descrizione dell'interfaccia dello stato corrente.
- Analisi Statica - Si effettua l'analisi statica delle funzioni relative ai gestori degli eventi trovati nella descrizione dell'interfaccia.
- Push - Si inserisce opportunamente l'etichetta dello stato corrente nella CSV. Si aggiornano opportunamente le tabelle descrittive

Processo di Reverse Engineering - 3

- **Descrizione**
  - Vengono compilate le prime tre colonne della tabella descrittiva dello stato corrente, andando a recuperare le informazioni necessarie dal documento HTML che la definisce.
  - Si analizza staticamente il codice disponibile sul lato client.
- **Analisi Statica**
  - Si analizza staticamente il codice di tutte le funzioni relative ai gestori degli eventi degli oggetti dell'interfaccia corrente.
  - Se dall'analisi statica di una funziona relativa ad un evento si evince che questa o una sua *subfunction* chiamata modifica l'interfaccia corrente, allora l'evento scatenante fa sempre cambiare stato all'applicazione; si dà un'etichetta momentanea al potenziale stato prossimo, indicandolo nella descrizione tabellare dello stato corrente. Inoltre si aggiunge una riga alla tabella degli stati prossimi. Si aggiunge lo stato potenziale alla CSDV.
  - Se la funzione chiamata non causa variazione di stato si inserisce un valore predefinito nella colonna dello stato prossimo; non si aggiorna né la tabella degli stati prossimi né la CSDV.
  - Se il gestore dell'evento (ovvero la funzione considerata) effettua una chiamata a XHR; si analizza staticamente il codice della funzione che gestisce la risposta dal server questa può causare o meno una transizione di stato; per tenere conto della presenza di una chiamata XHR si aggiunge un valore nel campo della colonna XHR (no oppure sì) relativa alla funzione considerata.
  - Lo stato corrente, inoltre, eredita nella sua descrizione tabellare le transizioni di stato dovute a chiamate a XHR dai suoi stati precedenti, data la natura asincrona delle richieste XHR.
    Non si aggiorna la tabella degli stati prossimi.



  - Nella tabella degli stati prossimi si può notare inoltre la presenza di una colonna di *input*. La presenza di tale colonna è stata prevista perché è possibile che la funzione effettui una modifica dell'interfaccia in base ai valori assunti da altri oggetti ad esempio *Input* di tipo *Text* o *Password* o *Select* che l'utente compila in maniera opportuna prima di scatenare l'evento.

Processo di Reverse Engineering - 4

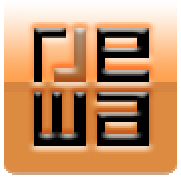
Oggetto	Evento	Funzione	Stato prossimo	XHR
Button	onclick	Function1()	S11	No
Button	ondblclick	Function22()	S21	sì
Form	onsubmit	Function2()	No	No
	XHR		S21	Sì
Image	onclick	Function12()	No	sì

Tabella descrittiva stato corrente

Stato prossimo	oggetto	Evento	Stato partenza	input
S12	button	Onclick	Sc	No
S88	button	ondblclick	Sa	No
S12	form	Onsubmit	Sk	Password errata
S23	form	Onsubmit	Sk	Password corretta
S101	image	Onclick	Sk	No

Tabella stati prossimi

REWA Tool

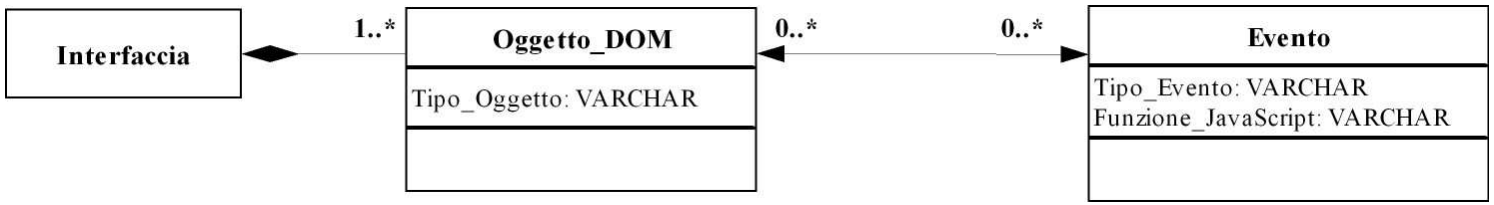


- **Software capace di automatizzare il processo di reverse engineering introdotto.**
- **È un'estensione per Firefox, aggiunge al browser nuovi elementi di interfaccia e nuove funzionalità.**
- **Le estensioni sono composte da una serie di componenti in:**
  - ***XML-based User-interface Language (XUL):*** **utilizzato per definire la struttura e il contenuto dell'interfaccia di un'applicazione.**
  - ***Cascading Style Sheets (CSS):*** **utilizzati per creare *il look and feel* dell'applicazione.**
  - ***JavaScript*: utilizzato per creare le funzionalità dell'applicazione. Anche altri linguaggi di *scripting* possono essere utilizzati al posto di JavaScript, come ad esempio *Python*, *Perl* e *Ruby*.**
  - ***Cross-Platform Install (XPInstall)*: utilizzato per impacchettare le applicazioni in modo che possano essere installate su varie piattaforme.**
  - ***eXtensible Binding Language (XBL)*: utilizzato per creare *widget* riutilizzabili tramite una combinazione di XUL e JavaScript.**
  - ***XPCOM/XPConnect/XPIDL*: utilizzati per permettere a JavaScript e, potenzialmente, a tutti gli altri linguaggi di *scripting* di accedere e utilizzare librerie in C/C++.**

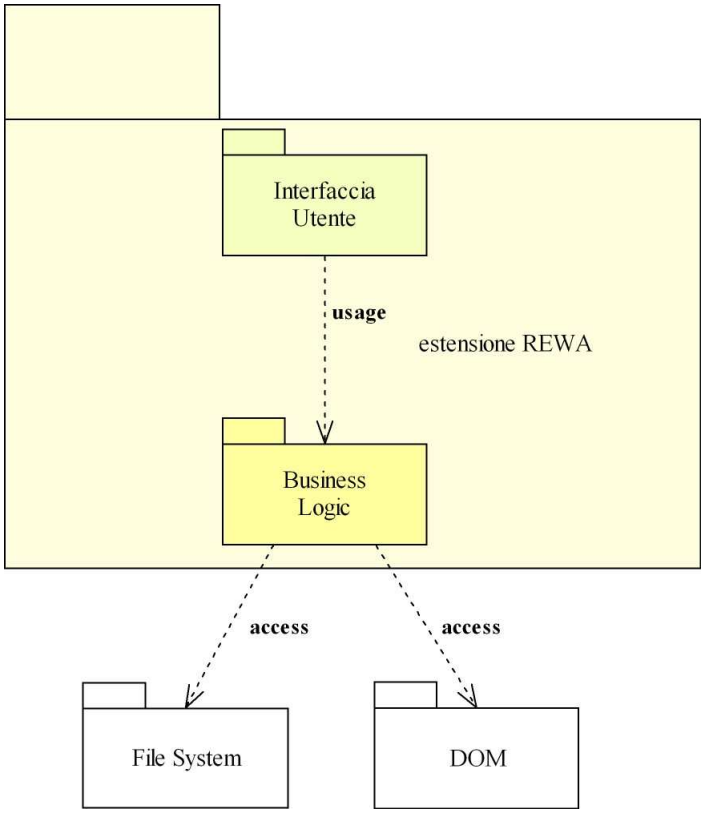
REWA Tool - 2

- Esegue il passo di Descrizione del processo di reverse engineering.
- Casi d'uso:
  - Descrivi Interfaccia – dal codice html che definisce l'interfaccia corrente si estrapolano le terne oggetto – evento – funzione JavaScript associata. Gli oggetti vengono cercati in base ai tag che li definiscono oppure in base ai valori assunti dal campo type se si tratta di oggetti di tipo input. Le informazioni vengono salvate in un file testo del tipo rewa*i*.txt, con i crescente, all'interno di una cartella presente sul desktop, tale directory viene creata la prima volta che si descrive una RIA ed ha il nome dell'url dell'applicazione.
  - Visualizza Descrizione Interfaccia – visualizza in una barra laterale del browser il contenuto del file rewa*i*.txt appena ottenuto.
  - Configurazione Analizzatore – inserisce una nuova coppia tag (oppure type del tag input) – evento nel file coppie.txt.

- System Domain Model dell'estensione



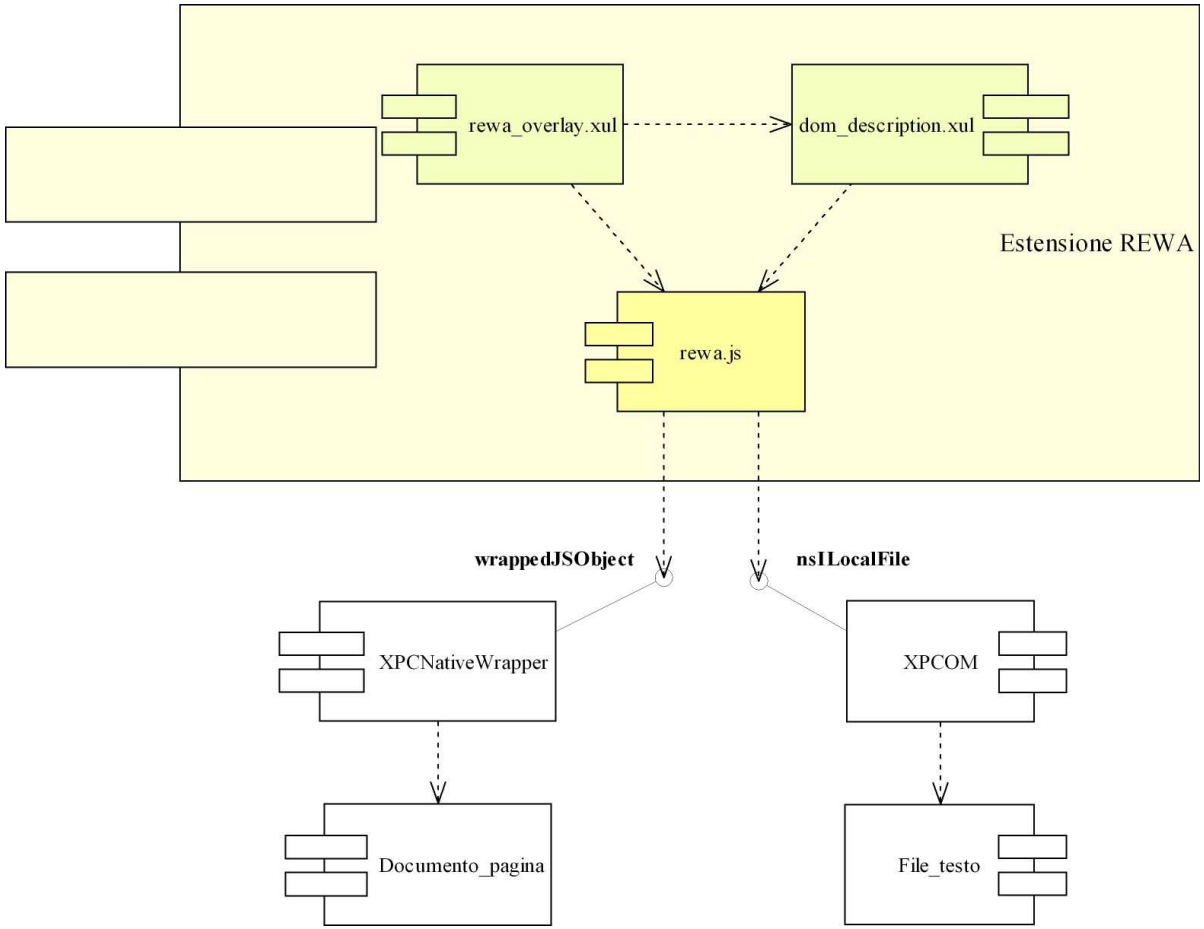
REWA Tool - 3



Package diagram dell'estensione

- **Estensione è strutturata con un pattern architetturale multilayer.**
- **Nella parte più alta dell'applicazione è presente l'interfaccia utente.**
- **L'interfaccia utilizza la Business Logic che fornisce all'applicazione le sue funzionalità.**
- **La Business Logic deve accedere sia al DOM, per ottenere le informazioni necessarie a descrivere l'interfaccia utente, che al File System per creare file e cartelle e per scrivere all'interno dei file.**

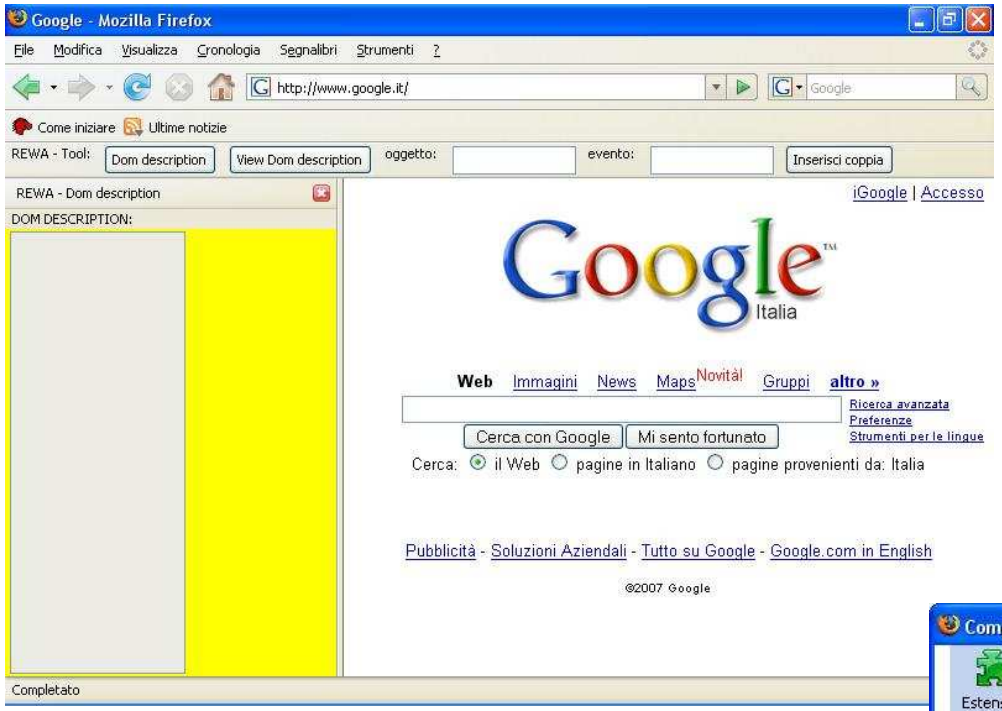
REWA Tool - 4



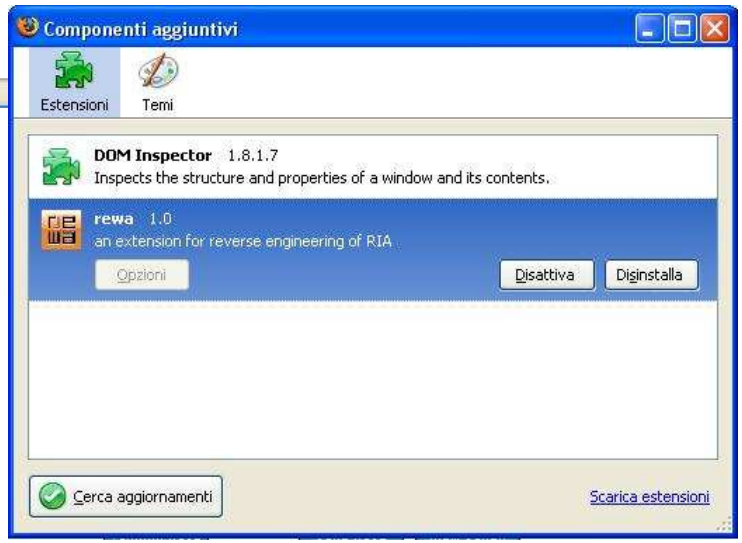
Component Diagram dell'estensione

- **L'interfaccia utente è realizzata attraverso due file .xul**
- **I due file .xul descrivono esclusivamente l'interfaccia utente, le vere funzionalità dell'applicazione sono contenute nel file JavaScript rewa.js che contiene tutte le funzioni per eseguire i casi d'uso richiesti.**
- **L'applicazione deve poter accedere al *File System* per scrivere e leggere da file. JavaScript utilizza le interfacce messe a disposizione dai componenti di XPCOM, per leggere e scrivere nei file, XPCOM mette a disposizione l'interfaccia *nsILocalFile*.**
- **L'estensione deve inoltre accedere al documento HTML. Molte delle informazioni del documento sono nascoste agli script dell'estensione, per accedere a questi dati si sfrutta l'interfaccia *wrappedJSObject* dell'oggetto *XPCNativeWrapper* che nasconde alcune proprietà e metodi, definiti tramite JavaScript, degli oggetti del documento agli script delle estensioni.**

Utilizzo REWA



Interfaccia di Firefox dopo l'istallazione dell'estensione



Componenti aggiuntivi di Firefox dopo l'istallazione dell'estensione

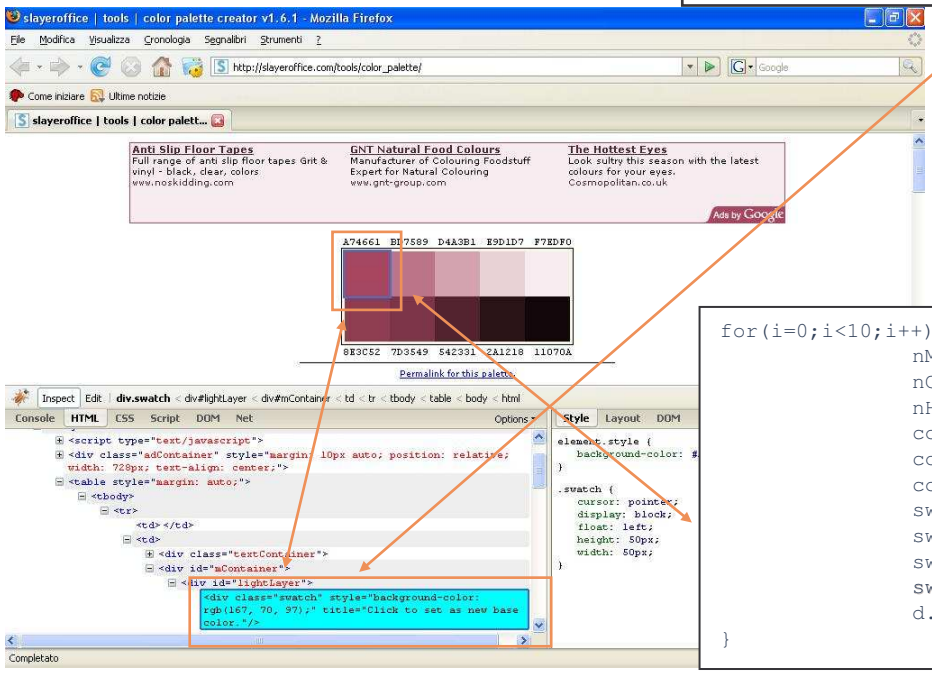
Utilizzo REWA - 2

- Si è utilizzata l'estensione su un'applicazione di cui si conosce la struttura.
- All'inizio si sono considerate le coppie presenti nel file coppie.txt:

```
a onclick div onclick pippo pippo a pippo password onclick /
```

- L'applicazione presenta anche oggetti a cui gli eventi sono associati via JavaScript.

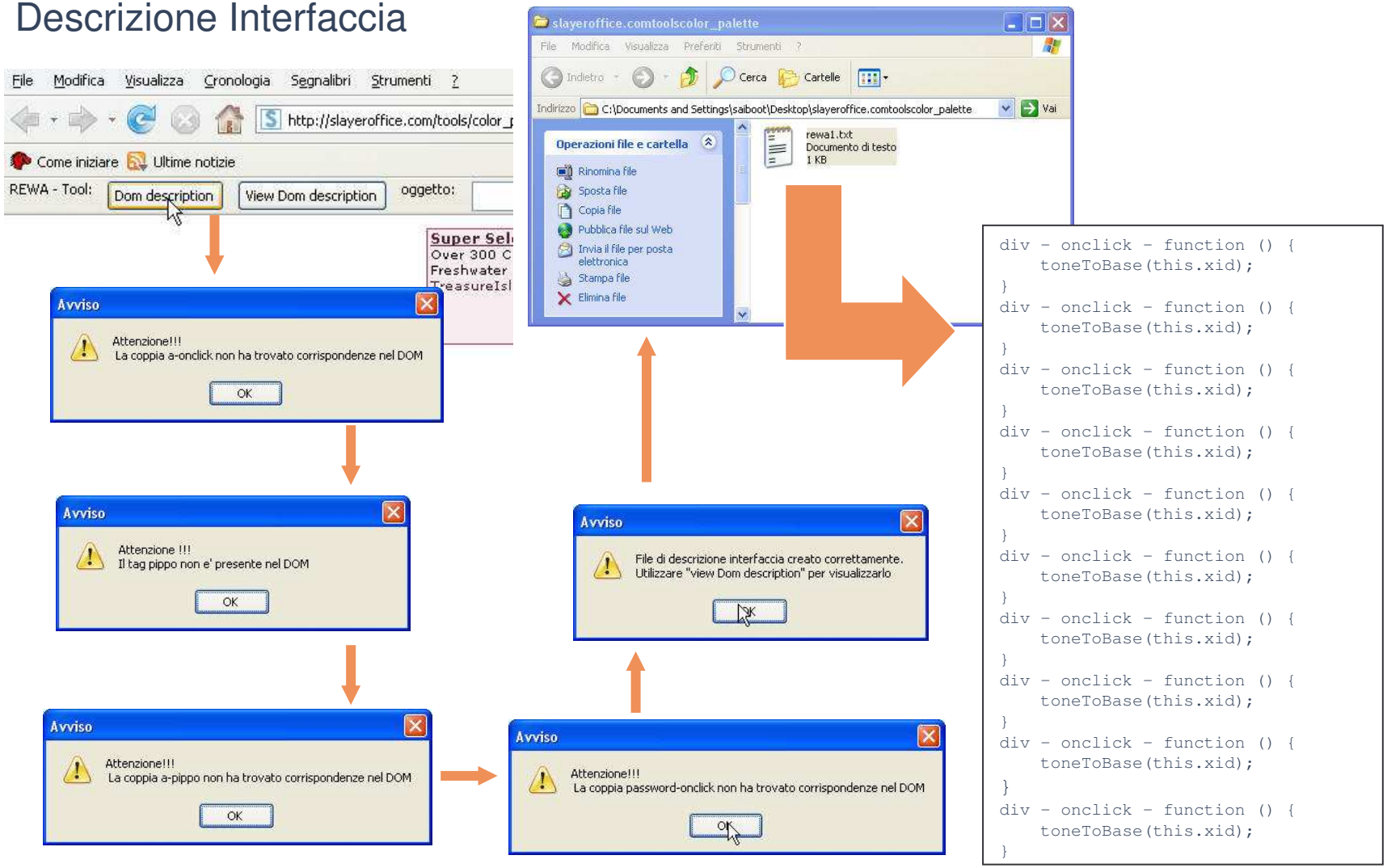
```
<div class="swatch" style="background-color: rgb(167, 70,
97);" title="Click to set as new base color."/>
```




```
for(i=0;i<10;i++) {
                nMask = i<5?lightRGB:darkRGB;
                nColor=setColorHue(baseColor,opArray[i],nMask);
                nHex = toHex(nColor[0])+toHex(nColor[1])+toHex(nColor[2]);
                colorValues[i]=new Array();
                colorValues[i][0] = nHex;
                colorValues[i][1] = nColor;
                swatch[i].style.backgroundColor = "#"+nHex;
                swatch[i].title = "Click to set as new base color.";
                swatch[i].xid=i;
                swatch[i].onclick=function(){ toneToBase(this.xid); }
                d.getElementById("hex"+i).innerHTML = nHex;
}
```

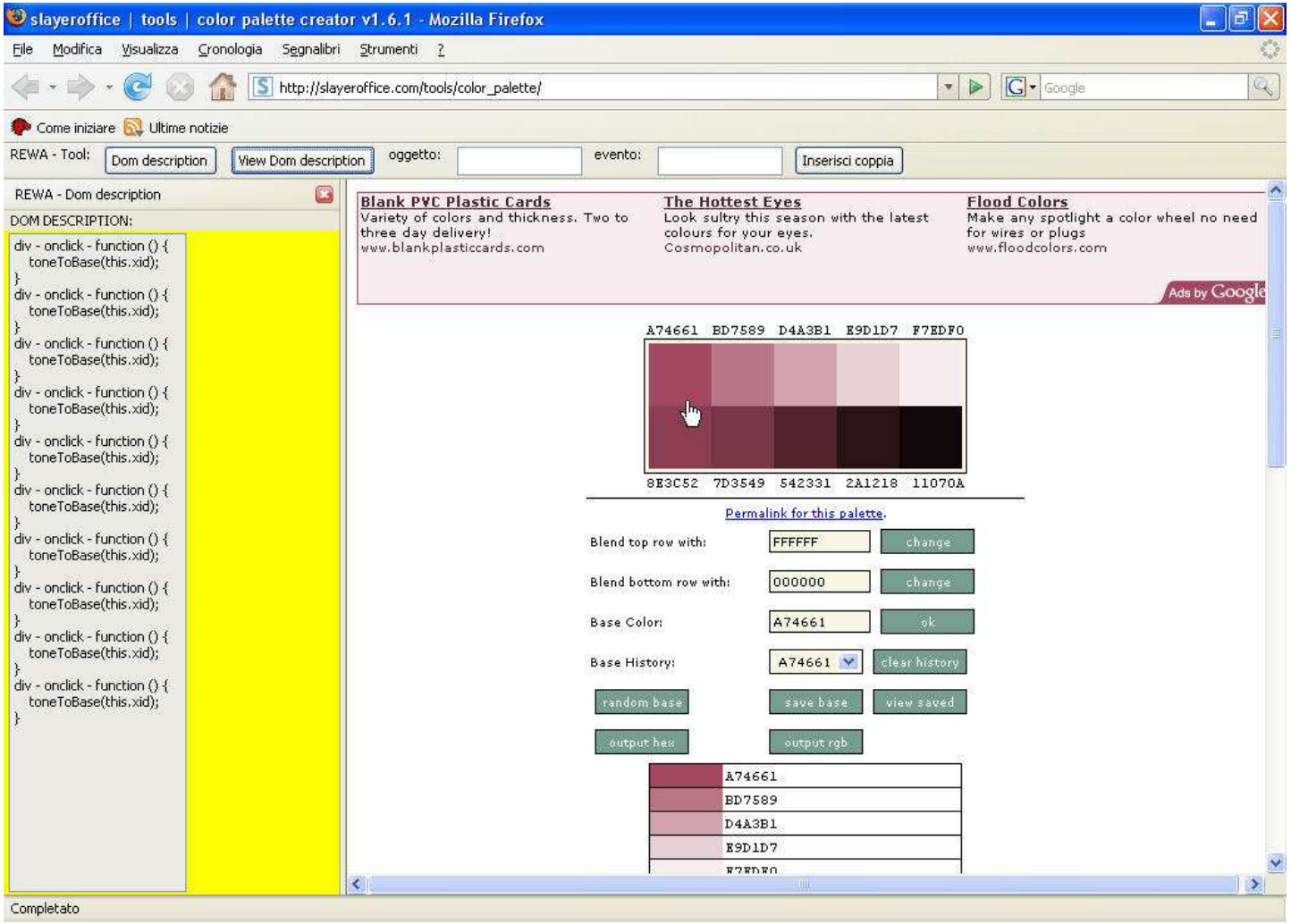
Utilizzo REWA - 3

Descrizione Interfaccia



Utilizzo REWA - 4

Per visualizzare immediatamente il contenuto del file ottenuto dalla descrizione dell'interfaccia corrente.



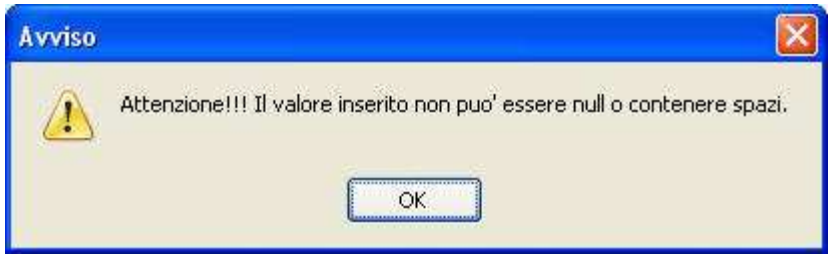
Utilizzo REWA - 5

Configurazione analizzatore, si aggiungono le coppie tag – evento

- button - onclick
- a - href

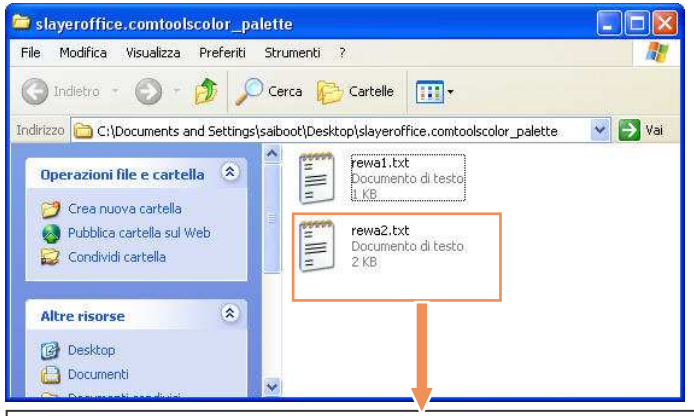

```
a onclick div onclick pippo pippo a pippo password onclick a
href button onclick /
```

Si possono avere i seguenti messaggi di errore:



Utilizzo REWA - 6

Descrivendo nuovamente l'interfaccia con le coppie tag - evento inserite




```
a - href - http://slayeroffice.com/?c=/content/tools/color_palette.html
a - href - http://slayeroffice.com/tools/color_palette/color_palette.js
a - href - http://slayeroffice.com/tools/color_palette/color_palette.css
a - href - http://slayeroffice.com/
a - href - mailto:steve@slayeroffice.com
a - href - http://slayeroffice.com/
button - onclick - function onclick(event) {
    changeBlend(0);
}
button - onclick - function onclick(event) {
    changeBlend(1);
}
button - onclick - function onclick(event) {
    createSwatches();
}
button - onclick - function onclick(event) {
    clearBaseHistory();
}
button - onclick - function onclick(event) {
    randomBaseColor();
}
button - onclick - function onclick(event) {
    saveBase();
}
button - onclick - function onclick(event) {
    viewSaved();
}
button - onclick - function onclick(event) {
    showVals("hex");
}
button - onclick - function onclick(event) {
    showVals("rgb");
}
```

```
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
div - onclick - function () {
    toneToBase(this.xid);
}
a - href - http://slayeroffice.com/tools/color_palette/?hex=A74661
a - href - http://www.stuffandnonsense.co.uk/archives/creating_colour_palettes.html
```