

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Specialistica in Scienze di Internet

Registrazione e replicazione di procedure Web con una Firefox Extension

Tesi di Laurea in Laboratorio di Programmazione Web

Relatore:
Chiar.mo Prof.
DAVIDE ROSSI

Presentata da:
ANDREA DI TORO

Sessione III
Anno Accademico 2009-2010

Indice

| | | |
|-------|--------------------------------------|----|
| 1 | Introduzione | 1 |
| 2 | Mozilla Firefox | 3 |
| 2.1 | Gecko..... | 5 |
| 2.1.1 | Flusso dati in Gecko..... | 6 |
| 2.1.2 | Sistema dei componenti di Gecko..... | 9 |
| 2.2 | XPCOM..... | 11 |
| 2.2.1 | Architettura dei Componenti..... | 13 |
| 2.2.2 | Definizione nuovi componenti..... | 16 |
| 2.2.3 | Usare Componenti XPCOM | 20 |
| 2.3 | XULRUNNER | 23 |
| 3 | Firefox Extension..... | 27 |
| 3.1 | Introduzione alle estensioni..... | 28 |
| 3.2 | Chrome | 30 |
| 3.2.1 | Chrome URL..... | 32 |
| 3.2.2 | Registrazione Chrome..... | 33 |
| 3.2.3 | Chrome Package..... | 35 |

| | | |
|-------|---|----|
| 3.2.4 | Chrome overlay..... | 36 |
| 3.5 | Localizzazione..... | 39 |
| 3.6 | Distribuzione | 40 |
| 3.6.1 | XPI | 41 |
| 3.6.2 | Install.rdf..... | 43 |
| 4 | Ambiente di sviluppo..... | 47 |
| 4.1 | Impostare Firefox come ambiente di sviluppo | 48 |
| 4.2 | Komodo Edit | 50 |
| 4.3 | Javascript Debugger..... | 52 |
| 5 | Descrizione del contesto applicativo | 59 |
| 5.1 | Lavori Correlati | 61 |
| 6 | WebRecorder | 63 |
| 6.1 | Introduzione ai concetti di “procedura” e “step”..... | 63 |
| 6.2 | Scelta tecnologica..... | 64 |
| 6.3 | Descrizione dell’applicazione | 65 |
| 6.3.1 | Home page | 66 |
| 6.3.2 | Rec page..... | 69 |
| 6.3.3 | Edit page..... | 70 |
| 6.3.4 | Play page | 74 |
| 6.3.5 | Option page | 75 |
| 6.4 | Case test..... | 76 |
| 6.5 | DOM event vs HTTP activity..... | 77 |
| 6.6 | Procedure basate su eventi DOM | 80 |
| 6.6.1 | Identificazione del documento | 81 |
| 6.6.2 | Identificazione dell’elemento DOM | 82 |
| 6.6.3 | Identificazione dell’evento DOM | 86 |

| | | |
|-------|--|-----|
| 6.6.4 | Identificare eventi in dialogBox..... | 91 |
| 6.6.5 | Dagli eventi DOM a una procedura | 92 |
| 6.7 | Monitorare Eventi DOM | 96 |
| 6.8 | Riprodurre eventi DOM | 98 |
| 6.9 | Architettura WebRecorder..... | 102 |
| 7 | Conclusioni | 107 |
| 8 | Indice delle figure | 109 |
| 9 | Bibliografia | 111 |

1 Introduzione

In questo lavoro di tesi è stata sviluppata una Firefox Extension per la registrazione e la replicazione di procedure sul Web. Si tratterà a fondo l'ambiente tecnologico nel quale è stata sviluppata l'applicazione e il contesto in cui si inserisce una Firefox Extension.

Illustreremo il problema che intendiamo risolvere con la nostra estensione, riportando una serie di lavori correlati che cercano, con diversi approcci, di risolvere il nostro stesso problema. Illustreremo il lavoro trattando approfonditamente l'approccio da noi utilizzato, mostrandone i vantaggi e i limiti.

Nel capitolo due abbiamo riportato una trattazione dei principali componenti di Firefox. Sarà mostrato il ruolo di Gecko all'interno del browser e il sistema XPCOM, che permette l'inclusione dei componenti esterni, fondamentale per l'estensione del browser. Nel capitolo tre seguirà una trattazione delle estensioni Firefox e illustreremo come il meccanismo Chrome permette di estendere l'interfaccia e caricare nuove applicazioni nel browser. Arriveremo alla fine della prima parte della tesi proponendo un ambiente di

sviluppo per le Firefox Extension, composto da un editor ed un debugger Javascript.

Nel capitolo cinque illustreremo il problema e i lavori correlati, mostrandone la complessità e le diverse soluzioni attualmente implementate. Infine, nel capitolo sei, sarà discusso il nostro approccio alla soluzione del problema, mostrando come la nostra idea si è evoluta durante tutto il processo di sviluppo. Saranno analizzati i problemi incontrati spiegando come siamo arrivati alla soluzione finale. Infine, concluderemo con il capitolo sette, discutendo dei problemi di standardizzazione del Web che limitano l'efficacia della nostra soluzione.

2 Mozilla Firefox

Mozilla Firefox è uno tra i Web browser più famosi al mondo, viene utilizzato circa da un utente su cinque ed è di fatto il secondo browser più diffuso.

Deve la sua enorme popolarità al fatto di essere un browser open source, veloce, facile da usare e soprattutto altamente personalizzabile. La combinazione di queste caratteristiche ha favorito la formazione di un'enorme comunità di sviluppo, che ha da subito contribuito alla creazione di temi, estensioni e plugin facendo di Firefox un browser in grado di adattarsi alle specifiche esigenze di ogni utente.

Firefox in realtà nasce come un ramo sperimentale di un altro progetto denominato Mozilla Application suite nato per rimpiazzare Netscape Navigator, oramai troppo lento per essere competitivo con i nuovi browser[7]. Così Netscape prima, AOL dopo e infine Mozilla Foundation, promuovono un nuovo progetto open source, che prevedeva la riscrittura totale di tutto il codice del vecchio Navigator, con l'obiettivo di creare una suite di programmi [7]:

- Modulare.
- Estendibile dagli utenti.
- Standard compliant W3C.
- Multiplatforma.

Grazie all'enorme successo di Firefox, Mozilla Foundation nel 2003 annuncia di voler spostare la propria attenzione da Mozilla Application Suite a Firefox e Thunderbird, considerati ora progetti di punta. Nonostante la versione 1.0 avesse dei seri problemi, le successive versioni si diffusero in fretta fino ad arrivare oggi al rilascio della 4.0 beta.

Il browser Firefox è composto da una collezione di librerie C++ disegnate per essere assemblate, combinate tra di loro e soprattutto per poter essere riusate dagli sviluppatori di temi, estensioni e plugin. Di fatto quella del riuso dei componenti è stata una delle caratteristiche che ha contribuito alla creazione di un enorme comunità di sviluppo. Attraverso il modello XPCOM è possibile sia utilizzare componenti messi a disposizione da Mozilla, che importare componenti esterni, permettendo di fatto l'estensione del browser aggiungendo nuova logica e nuove funzionalità.

Il componente più importante del browser è sicuramente il motore di rendering Gecko, anch'esso scritto in C++ che garantisce un rendering veloce ed affidabile. Disegnato per supportare Web standard e sviluppato rispettando le linee guida Mozilla Foundation, quindi multiplatforma, standard compliant e modulare. E' stato progettato esplicitamente con l'obiettivo di produrre un layout engine che potesse essere riusato in differenti applicazioni come Web browser, content presentation e sistemi client server, infatti è il motore di rendering di tutti i prodotti Mozilla Foundation.

Mozilla non è solo un browser. Da alcuni viene anche considerato come un framework per costruire Web Application usando tecnologie standard come CSS, XML, XUL, Javascript, RDF. Gecko, il motore di rendering, può essere usato come parte del framework insieme alle tecnologie XPCOM e XPCONNECT che permettono di riusarne componenti e moduli. In aggiunta permettono di importare componenti esterni scritti in C++, Java, Python[2].

2.1 Gecko

Gecko è sicuramente il componente principale del browser, è il rendering engine che si occupa di visualizzare pagine Web e interfacce XUL. Questo significa che Gecko riceve in ingresso contenuti e informazioni di formattazione e crea in output la loro rappresentazione grafica.

Gecko viene usato anche per il rendering di interfacce XUL, la tecnologia utilizzata per tutte le interfacce Mozilla. In altre parole significa che Gecko non viene utilizzato solo per il rendering del documento Web, ma anche per quello dell'intera interfaccia grafica di Firefox, offrendo il rendering per sidebar, scrollbar, toolbar e ogni altro elemento grafico di Mozilla Firefox [7].

Lo sviluppo di Gecko è stato controllato da Mozilla Foundation e rispetta tutte le linee guida della comunità. E' stato sviluppato con lo specifico obiettivo di essere riusato in tutte le applicazioni Mozilla, di fatto è il motore di rendering di tutta la Suite Mozilla, SeaMonkey, Camino e fu usato anche in Netscape fino alla versione 6.

Gecko è in continuo sviluppo, attualmente è stata rilasciata la versione 2 beta, fornita su browser Firefox 4 beta. Questa versione include importantissime novità come supporto HTML 5 e CSS 3.

Disegnato per essere standard compliant W3C le principali tecnologie implementate sono [9] :

- HTML4 (supporto parziale HTML5).
- CSS Level 2.1 (supporto parziale CSS 3).
- JavaScript 1.8 (ECMAScript 3, supporto parziale ECMAScript 5).
- DOM Level 1 and 2 (supporto parziale DOM 3).
- XML 1.0.
- XHTML 1.0.
- XSLT e XPath, implementato in TransformiX.
- MathML.
- XForms .
- RDF.

2.1.1 Flusso dati in Gecko

I contenuti e le informazioni di formattazione arrivano via rete e vengono indirizzate al layout engine. Nel seguente diagramma si riporta il flusso dei dati all'interno del single-threaded di Gecko.

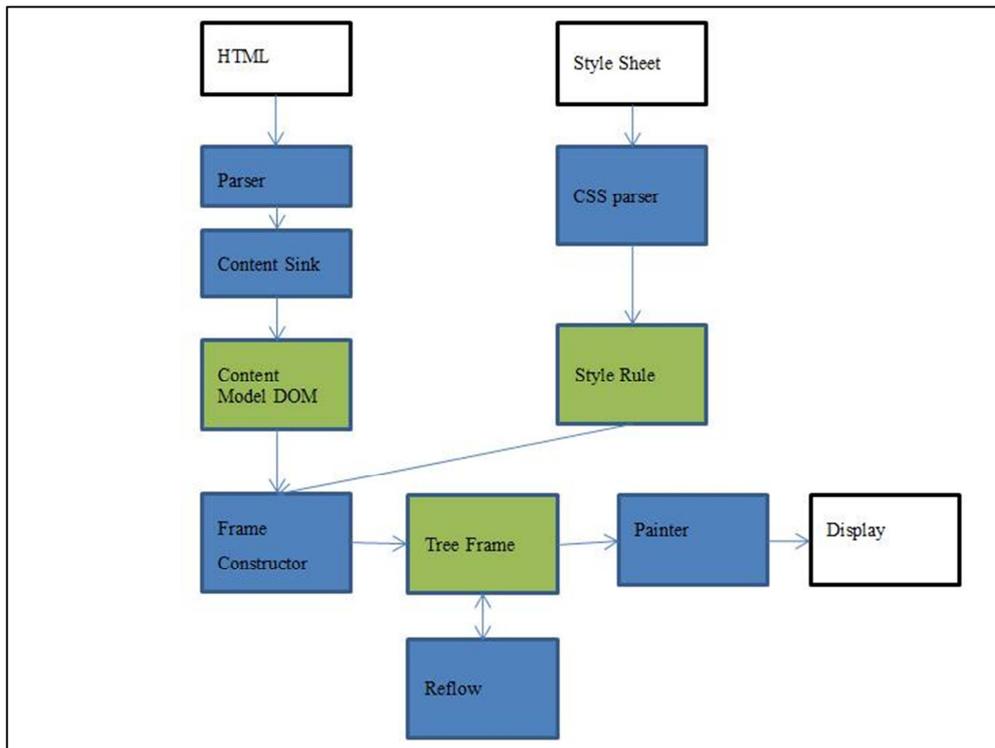


Figura 1: Gecko data Flow

Nel diagramma di Figura 1, rappresentante il flusso dati all'interno di Gecko, sono state riportate tutte le operazioni che vengono eseguite sui dati dal loro arrivo fino alla loro visualizzazione.

Nella parte iniziale del diagramma, il documento HTML viene dato al parser di Gecko. Da questa fase si ottiene il "Document", cioè la rappresentazione ad albero del documento HTML. La sua struttura ed i suoi attributi sono basati sullo standard W3C Document Object Model. Da questo punto in avanti ogni modifica ai dati e ai contenuti potrà essere effettuata solo attraverso le DOM APIs, le quali permettono di manipolare il Data Model e modificare quindi la struttura dell'albero.

I CSS, contenenti informazioni di formattazione, alla stessa maniera vengono dati al parser, generando così oggetti contenuti le "Style Rule". Come per il DOM, anche per le Style Rule da questo punto in avanti potranno essere modificate solo usando appositi APIs.

I modelli ottenuti dalle fasi precedenti arrivano al "Frame Constructor", il quale si occupa della costruzione e del posizionamento degli elementi nella pagina.

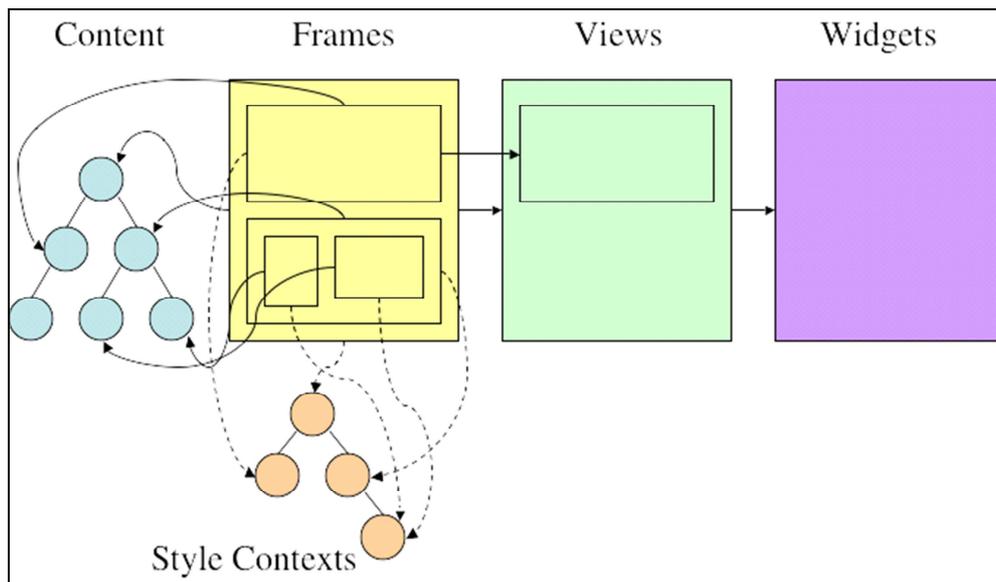


Figura 2: Fasi del frame constructor

Nel diagramma di Figura 2, vengono riportate le fasi interne del Frame Constructor, dove la costruzione e il posizionamento dei frame è dato dal merge dei due modelli prodotti precedentemente. A questo punto del data flow si producono i "Frame", cioè una semplice astrazione dei box con i quali vengono visualizzati gli elementi HTML.

Come si può vedere nella parte iniziale del diagramma 2, gli elementi DOM sono inseriti all'interno dei frame tenendo in considerazione gli stili di posizionamento float e fissi già pre-calcolati, che permettono un primo posizionamento degli elementi. Nella fase successiva avviene l'effettivo accoppiamento tra gli Style Context che non contengono più riferimenti geometrici, ma solo indicazioni sul flusso di posizionamento. Nella terza fase si passa alle Views dove viene assegnato lo z-order e la trasparenza dei vari frame. Infine, vengono inseriti all'interno di un widget Native Window per la visualizzazione a monitor.

In questo processo, gli elementi di posizionamento fisso sono definiti fuori dal flusso permettendo un pre-posizionamento dei frame, inseriti nella loro posizione "originale". In seguito intervengono gli elementi di Style Rule che influiscono sul flusso di posizionamento ed infine gli elementi che modificano z-order e trasparenza. Quindi, un accoppiamento tra DOM e Style Rule che avviene per step.

Da questo processo viene prodotto il Frame Tree, sostanzialmente la rappresentazione grafica ottenuta dal merge dei due modelli.

Con riferimento al diagramma di Figura 1, possiamo vedere che il Frame Tree viene sottoposto all'operazione di Reflow, necessaria a tenere allineati i modelli DOM e Style Rule con il Tree Model, nel caso in cui ci siano nuovi flussi di informazioni. Modificando uno dei due modelli, il Frame Constructor segnerebbe come "Dirty" l'elemento del Tree Model non più allineato. L'operazione di Reflow attraverserebbe ogni nodo del Tree Frame e processerebbe ogni elemento contrassegnato come 'Dirty' fino a quando tutti gli elementi non sarebbero "Clean". Ogni elemento del Tree Frame ha un puntatore al suo corrispondente DOM e viceversa, quindi una manipolazione dei dati DOM attraverso le DOM APIs produrrà una conseguente modifica al Frame Tree.

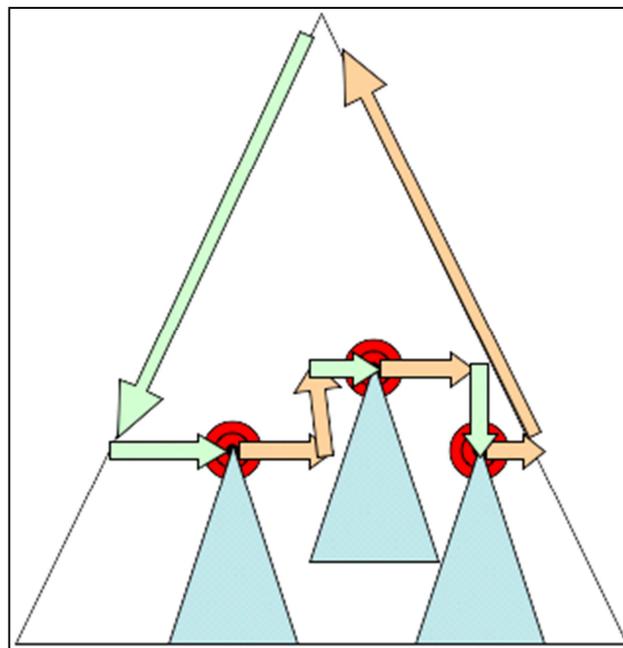


Figura 3: Reflow

Ogni nodo del Tree Model etichettato come “Dirty” viene quindi processato durante l'operazione Reflow, come mostrato in Figura 3. Vengono ricalcolate esattamente le posizioni e le dimensioni degli elementi frame e widget. Dati x , y , w , h (posizioni e dimensione) del frame contenente l'elemento Dirty, si ricalcolano i valori di x , y , w , h dei figli con posizionamento relativo, continuando così in maniera ricorsiva analizzando tutti i sottolivelli fino alla radice. Questo è necessario, dato che cambiando la posizione di un elemento potrebbe cambiare il posizionamento dei figli con posizionamento relativo.

Infine viene eseguita l'operazione di Painting e la visualizzazione finale del documento. Queste sono le uniche fasi dove Gecko richiede l'intervento del sistema operativo. L'attuale operazione di Painting viene gestita da gfx-submodule e widget-submodule, gli unici moduli a maneggiare la comunicazione con il sistema operativo. Questi due moduli rendono semplice la gestione delle problematiche relative all'interoperabilità con diversi sistemi operativi.

2.1.2 Sistema dei componenti di Gecko

Come detto in precedenza, Gecko è progettato per poter essere riusato come motore di rendering in tutti i progetti Mozilla. Dato che Gecko è piccolo, veloce, multiplatforma ed open Source, molte organizzazioni hanno deciso di utilizzarlo interamente o solo parzialmente come componente interno delle loro applicazioni. Ovviamente alcuni componenti Gecko sono principali, mentre altri sono opzionali, questo permette anche un'inclusione parziale. L'inclusione di Gecko non è l'unico modo per poterne riusare i componenti, ma è anche possibile anche riutilizzare le sue funzionalità creando una Extension installabile su applicativi che includono Gecko, come ad esempio una Firefox Extension[12].

Di seguito vengono elencati i componenti principali: [7]

- Document parser (HTML, XML).
- Layout engine con content model.
- Style system (CSS).
- JavaScript runtime (SpiderMonkey).

- Image library.
- Networking library (Necko).
- Platform-specific graphics rendering and widget sets for Win32, X, Mac.
- User preferences library.
- Mozilla Plug-in API (NPAPI) to support the Navigator plug-in interface.
- Open Java Interface (OJI), with Sun Java 1.2 JVM.
- RDF back end.
- Font library.
- Security library (NSS).

Lo sviluppo di un numero elevato di tecnologie ad-hoc, ha garantito la preservazione delle sue caratteristiche fondamentali quali:

- Modularità.
- Compatibilità con diverse piattaforme.
- Compatibilità con molteplici linguaggi di programmazione .

La conoscenza di queste tecnologie è fondamentale per poter capire come includere Gecko e come poter interagire con il rendering engine .[12]

XPCOM Cross-Platform Component Object Model, trattato più a fondo nel seguente paragrafo, sicuramente costituisce il meccanismo principale per il riuso dei componenti. Il cuore di XPCOM è il concetto di interfaccia, contenente una descrizione dei metodi e degli attributi per ogni componente. Ognuno di essi deve supportare una o più interfacce e garantirne l'implementazione dei metodi. Per rendere il sistema più astratto possibile le interfacce sono definite in uno speciale linguaggio di IDL (Interface Definition Language). Questo permette di importare all'interno del sistema anche componenti scritti in diversi linguaggi di programmazione.

Il sistema di XPCOM, su cui si basa Gecko, permette quindi di riutilizzare agevolmente i suoi componenti, rendendone particolarmente semplice il concetto di inclusione e di estensione. Di seguito vengono riportate le principali interfacce di Gecko:

- nsIWebBrowser: viene utilizzato per includere Gecko e per accedere al DOM durante la navigazione .

- nsINavigation: viene utilizzato per caricare una nuova URI e per accedere ai dati di sessione .
- nsWebBrowserPersist: viene utilizzato per salvare degli URI su file.
- nsISHistory: permette di accedere alla cronologia e di cancellarla.
- nsProgressListener: viene utilizzato per ricevere notifiche su eventi che possono verificarsi all'interno di Gecko, come per esempio sul traffico http.
- nsISupports: è un 'interfaccia che ogni componente è costretto a implementare. Un oggetto può supportare anche più di una interfaccia ma deve offrire un'implementazione per i metodi di nsISupports. Il primo metodo è queryInterface() serve a verificare se un oggetto implementa un certo tipo di interfaccia.

2.2 XPCOM

XPCOM Cross Platform Component Object Module è un framework per lo sviluppo di software modulare, offre strumenti per creare, assemblare e manipolare componenti a run-time. E' il cuore dei sistemi Mozilla, permette di estendere e includere Gecko. Ha reso famosa la famiglia di applicazioni Mozilla per la loro facilità di estensione e di personalizzazione.

Il sistema XPCOM permette di creare componenti in C++, Javascript, Java e Python, riusarli all'interno del sistema garantendo l'interoperabilità anche tra componenti creati in differenti linguaggi di programmazione. Possiamo affermare che funge da proxy tra tutti i componenti. In aggiunta XPCOM offre un buon livello di astrazione e permette ai componenti di lavorare sui seguenti sistemi operativi:

- Microsoft Windows .
- Linux.
- HP-UX.
- AIX.
- Solaris.

- OpenVMS.
- MacOS.
- BSD.

L'idea alla base di XPCOM è quella di fornire un framework di lavoro indipendente dalla piattaforma e dal linguaggio.

Il cuore del sistema è composto dal Service Manager e dal Component Manager, insieme questi due servizi forniscono un punto di accesso centralizzato per tutti i componenti [13].

Il Component Manager tiene traccia dei componenti correntemente installati e delle librerie necessarie per la creazione di uno specifico componente. I componenti sono identificati in due possibili modi, un 128-bit UUID conosciuto come class ID, oppure con un CID, nome breve conosciuto come Contract ID. Il component manager offre elementi per verificare la registrazione del componente, crearne istanze e altri metodi per la gestione in generale.

Il Service Manager si occupa di offrire accesso ai servizi e lavora in maniera differente rispetto al Component Manager. Quando un componente ne usa un altro, tipicamente ne crea una nuova istanza ogni volta che ne richiede le funzionalità. Questo non è l'unico modo possibile per accedere ai servizi di un componente. Ci sono casi in cui un oggetto può rappresentare semplicemente un servizio e non è necessario crearne una nuova istanza ogni volta che si voglia usarlo. Questo concetto viene espresso perfettamente dal pattern Singleton, ampiamente utilizzato all'interno dell'architettura XPCOM attraverso il Service Manager.

Il Category Manager viene utilizzato per la gestione delle interfacce, si occupa di gestire l'intero elenco di interfacce disponibili, dividendole in base a categorie di appartenenza per velocizzare le operazioni di ricerca.

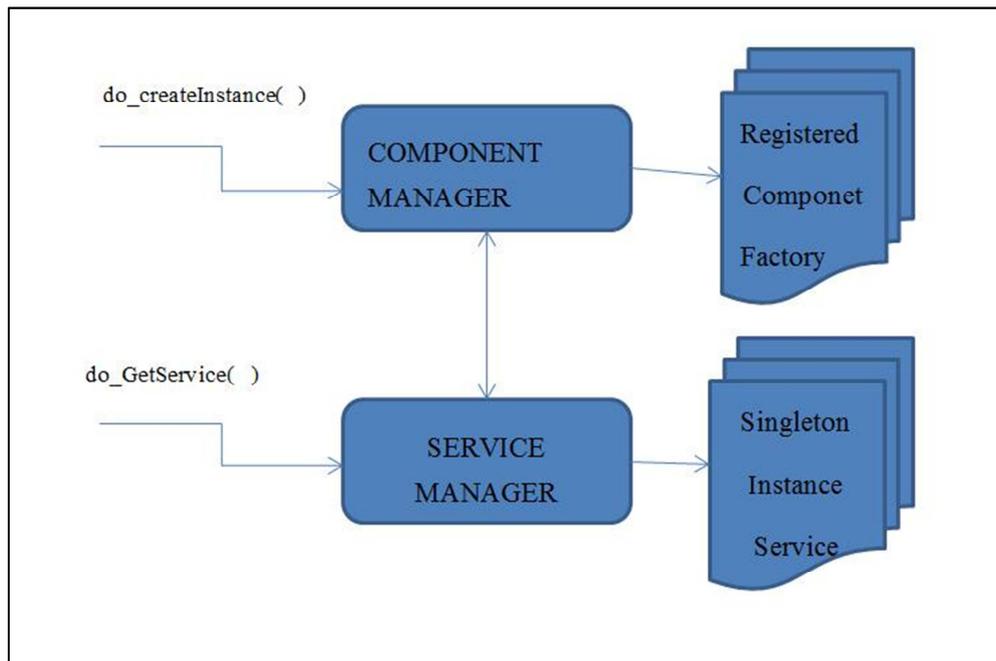


Figura 4: XPCOM

2.2.1 Architettura dei Componenti

Come detto in precedenza il sistema XPCOM fornisce un sistema centralizzato per l'interoperabilità dei componenti, ovviamente per poter rendere disponibile un componente all'interno di un sistema XPCOM, è necessario svilupparlo in modo che sia accessibile e compatibile con l'architettura generale .

Il cuore del sistema si basa sul concetto di interfaccia e di Type Library. Entrambi servono a fornire una descrizione dettagliata del componente, permettendo al Service Manager e al Component Manager di potervi accedere, crearne istanze e renderlo disponibile all'intero sistema. Tramite la definizione di interfaccia in un linguaggio che possa essere interpretato da XPCOM, è possibile dare informazioni sul nostro componente, sui suoi metodi, i suoi attributi e altre meta-informazioni utili a capire le caratteristiche del componente. Questo permette il funzionamento del meccanismo di scambio dati. Stabilendo un formato comune, la stessa interfaccia può essere supportata da diversi componenti scritti anche in differenti linguaggi di programmazione.

XPIDL è un linguaggio di IDL, Interface definition language, ed è utilizzato per definire interfacce nella piattaforma XPCOM. Nonostante presenti

alcune limitazioni sull'estendibilità, è molto potente e permette la creazione di tipi di dati. XPIDL permette la generazione di file XPT, contenenti la definizione binaria delle interfacce, questo è il punto chiave che permette la coesistenza di componenti scritti in linguaggi differenti come Javascript o C++. Quindi, XPCOM può ottenere accesso ad un altro componente solo conoscendo nel dettaglio la sua interfaccia e i tipi di dati da essa utilizzata.

Il sistema non effettua nessun controllo per verificare la corrispondenza tra l'interfaccia e il componente, ma sono i componenti che dovrebbero garantire il rispetto dell'accordo. Di seguito si riporta la definizione di un'interfaccia in XPIDL.

```
#include "nsISupports.idl"
[scriptable, uuid(f728830e-1dd1-11b2-9598-fb9f414f2465)]

interface nsIScreen : nsISupports
{
    void GetRect(out long left, out long top, out long width, out long
height);
    void GetAvailRect(out long left, out long top, out long width,
out long height);
    readonly attribute long pixelDepth;
    readonly attribute long colorDepth;
};
```

Listato 1

Con la parola chiave *interface* inizia la dichiarazione dell'interfaccia, in questo caso si chiama *nsIScreen*, segue la dichiarazione dei metodi e degli attributi. Nell'header del file sono presenti meta-informazioni, come *scriptable* indica la possibilità di essere supportata da un component Javascript, *UUID* indica l'identificatore univoco dell'interfaccia. L'inclusione dell'interfaccia *nsISupport* indica che i metodi e gli attributi definiti in essa vengono importati all'interno di *nsIScreen*.

Una limitazione di questo linguaggio è che non permette l'estensione di interfacce e non permette di creare metodi con lo stesso nome e con input differenti (come succede in molti altri linguaggi di programmazione Java,C++).

Un singolo componente può supportare anche più interfacce per questo motivo deve fornire metodi per:

- Determinare se un componente supporta una determinata interfaccia
- Cambiare da un'interfaccia all'altra

Tutti i componenti XPCOM devono supportare un' interfaccia base chiamata *nsISupport* ,considerata l'interfaccia madre di tutte le interfacce. Obbligando così a fornire un'implementazione del metodo *QueryInterface*, utilizzato per verificare se un componente supporta o meno una determinata interfaccia e per passare da un'interfaccia all'altra.

```
interface nsISupports
{
    void QueryInterface(in nsIIDRef uuid, out nsQIResult result);
    nsrefcnt AddRef();
    nsrefcnt Release();
};
```

Listato 2

I metodi *AddRef* e *Relise* servono per la gestione del ciclo di vita di un componente. Tengono traccia di quante *reference* ha una determinata interfaccia necessarie a capire quando è possibile cancellare un componente. Nel caso in cui un oggetto abbia più puntatori verso se stesso, significa che più di un client sta utilizzando i suoi servizi ed è estremamente pericoloso procedere con la cancellazione dell'oggetto. E' quindi fondamentale, tenere traccia di quanti puntatori sono stati forniti, per evitare la distruzione dell'oggetto finché ci sono altri componenti che potrebbero richiederne servizi. I metodi *Addref* e *Relase* rispettivamente incrementano e decrementano un contatore, quando il contatore è a zero , l'oggetto si autodistrugge.

Rilasciare un puntatore ad una interfaccia permette di liberare spazio nella memoria, se questo non venisse fatto, il componente inutilizzato continuerebbe ad occupare spazio, portando ad una gestione scorretta della memoria. Il sistema di tracciamento dei riferimenti è quindi un contratto tra un componente client e il componente server, dove il primo si impegna a richiedere servizi con la corretta gestione dei puntatori ed il server che si impegna a offrirne

una implementazione.

In XPCOM i puntatori sono puntatori ad interfaccia, la differenza con un puntatore normale è che un *interface pointer* sicuramente punterà ad un oggetto che supporta l'interfaccia *nsISupport*, quindi avrà la definizione dei metodi *AddRef*, *Release* e *QueryInterface*.

Come detto in precedenza, ad ogni interfaccia viene assegnato un identificatore generato da un tool `uuidgen`. `UUID` `universally unique identifier` è un unico numero a 128 bit, che nel contesto di un interfaccia viene chiamato `IID` `interface identifier`. L'identificatore viene utilizzato da molti metodi, ad esempio *QueryInterface* per identificare in maniera univoca la interfacce supportate.

`CID` invece è un numero a 128 bit, che identifica univocamente una classe o un componente. La lunghezza del `CID` può renderne complicato l'uso nel codice, per questo viene usato un `Contract ID`, una stringa facile da ricordare che può essere usata per avere un richiedere un componente al `Component Manager`. Ne segue un esempio:

```
"@mozilla.org/network/ldap-operation;1"
```

Un `Contract ID` è composto dal:

dominio/modulo/component name/ numero di versione

La differenza tra un `CID` ed un `Contract ID` sta nel fatto che un `Contract ID` non si riferisce ad una specifica implementazione, un `Contract ID` specifica solo un gruppo di interfacce che vuole implementare e dei `CID` che vuole.

2.2.2 Definizione nuovi componenti

In questa sezione andremo ad illustrare come creare un nuovo componente e come renderlo disponibile all'interno dell'architettura XPCOM. Le operazioni di creazione di un componente si possono riassumere come elencato di seguito:

- Generare un class-id UUID.
- Creare un Contract ID.
- Definire la propria interfaccia in XPIDL.
- Compilare la propria interfaccia e generare il binario XPT.
- Creare il componente che supporti l'interfaccia.
- Registrare il componente all'interno del sistema XPCOM.

Al termine di queste operazioni il componente sarà disponibile all'interno del sistema e potrà essere riutilizzato come servizio o come oggetto.

Come detto in precedenza, XPCOM fornisce un'astrazione ai componenti permettendo l'interoperabilità anche tra componenti creati in differenti linguaggi di programmazione. E' necessario però, affrontare un discorso particolare per quanto riguarda i linguaggi di script, ad esempio Javascript necessita di un servizio aggiuntivo per poter interagire con la piattaforma.

XPCONNECT (Cross Platform Connect) è la tecnologia che permettere l'interazione tra la piattaforma XPCOM e componenti Javascript, di fatto funge da ponte con la piattaforma. XPCONNECT opera in maniera assolutamente trasparente, permette a metodi dichiarati in interfacce XPIDL di essere richiamati come parte di un oggetto Javascript. Questo concetto è importantissimo nei sistemi Mozilla, in quanto Javascript viene utilizzato come "Lingua franca" e la maggior parte delle interfacce e delle estensioni viene scritta in questo linguaggio.[18]

Per creare un nuovo componente XPCOM è necessario quindi definire l'interfaccia. Un buon metodo, è quello di visualizzare tutte le interfacce disponibili nel sistema e verificare che non esistano già delle interfacce con i nostri requisiti. L'elenco delle interfacce installate nel sistema, può essere visualizzato con XpcomViewer, uno strumento che permette di analizzare le interfacce e i loro metodi. Nel caso in cui non trovassimo nulla di utilizzabile, possiamo procedere alla creazione di un'interfaccia in XPIDL. Ne segue un esempio.

```
#include "nsISupports.idl"
```

```
[scriptable, uuid(xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx)]  
interface nsIHelloWorld : nsISupports  
{  
    string hello();  
}
```

Listato 3

Nella prima linea dell'esempio, vediamo che è stata importata l'interfaccia nsISupport, definisce i metodi base che devono essere implementati da tutti i componenti presenti nel sistema. Nella linea successiva si assegna un UUID e si definisce l'interfaccia come scriptable, questo permette alla nostra interfaccia di essere implementata anche da Javascript. Nella parte centrale riportiamo l'elenco dei metodi e delle proprietà.

Adesso l'interfaccia deve essere compilata in formato binario XPT per poter essere registrata e usata all'interno di XPCOM. La compilazione può avvenire usando il compilatore presente nella Gecko SDK.

A questo punto, possiamo iniziare con la definizione del componente Javascript, fornendo un'implementazione di tutti i metodi presenti nelle interfacce *HelloWorld* e *nsISupport*. Segue la definizione di un componente:

```
//reference all'interfaccia nsIHelloWorld  
const nsIHelloWorld = Components.interfaces.nsIHelloWorld;  
  
//reference all'interfaccia nsISupport  
const nsISupports = Components.interfaces.nsISupports;  
  
//id del componente pregenerato con un apposito tool.  
const CLASS_ID =  
Components.ID( "{xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx}" );  
//cid del componente  
const CONTRACT_ID = "@dietrich.ganx4.com/helloworld;1";  
  
HelloWorld.prototype = {  
  
    hello: function() {  
        return "Hello World!";  
    }  
};
```

```

},

QueryInterface: function(aIID)
{
    if (!aIID.equals(nsIHelloWorld) &&
        !aIID.equals(nsISupports))
        throw Components.results.NS_ERROR_NO_INTERFACE;
    return this;
}
};

```

Listato 4

Nella parte superiore del listato 4, avviene la definizione degli ID e la dichiarazione dei riferimenti alle interfacce supportate, fondamentale per permettere ai metodi come *QueryInterface* di poter lavorare. Quindi, si fornisce un'implementazione del metodo *hello* e del metodo *QueryInterface* che verifica se un componente implementa una certa interfaccia eseguendo un confronto tra il IID in ingresso e quello delle interfacce supportate. Successivamente è necessario definire il metodo *factory*, ne segua la definizione.

```

var HelloWorldFactory = {
    createInstance: function (aOuter, aIID)
    {
        if (aOuter != null)
            throw Components.results.NS_ERROR_NO_AGGREGATION;
        return (new HelloWorld()).QueryInterface(aIID);
    }
};

```

Listato 5

L'oggetto *HelloWorldFactory* verrà indicizzato nell'array globale *Component.Classes*, che come mostrato nel listato 6, servirà a creare un'istanza del nostro componente accedendo ai metodi *createInstance* e *getService*.

```
Components.classes["@dietch.ganx4.com/helloworld;1"].createInstance(Components.interfaces.nsiHelloWorld);
```

Listato 6

L'ultima fase è quella del processo di registrazione che permette di aggiungere componenti al sistema XPCOM, l'operazione può essere eseguita in diversi modi:

- Utilizzando XPInstall, installando un estensione o un plugin vengono automaticamente aggiunti anche i suoi componenti all'interno del sistema XPCOM e saranno successivamente accessibili da tutto il sistema.
- Utilizzando il programma regxpcom distribuito con Gecko SDK, eseguendolo da linea di comando è possibile aggiungere e rimuovere componenti.
- Utilizzando il metodo di auto-registrazione del Service Manager. Permette l'installazione di un componente solo nel momento in cui viene avviata l'applicazione e la disinstallazione nel momento in cui verrà chiusa. Il componente però deve implementare i metodi *registerSelf* e *unregisterSelf* che ne permettano le operazioni di aggiunta e rimozione dal sistema.

2.2.3 Usare Componenti XPCOM

Un discorso importante da affrontare prima di passare all'uso di componenti, è capire quando un componente è effettivamente disponibile all'interno del sistema. Per questo è necessario comprendere come avviene l'inizializzazione del sistema XPCOM. Tutto dipende dall'applicazione in questione, ad esempio un'applicazione che include Gecko potrebbe avviare XPCOM da un'azione dell'utente o all'avvio dell'applicazione stessa, tutto dipende da quando viene chiamato il metodo dell'inizializzazione definito negli API. In aggiunta, negli API sono definiti anche metodi che permettono la configurazione del sistema, come la localizzazione di specifiche directory. I passi che vengono eseguiti al momento dell'inizializzazione sono i seguenti

1. L'applicazione avvia XPCOM.
2. XPCOM avvia una notifica di inizio StartUp.
3. XPCOM trova e processa il Component Manifest.
4. XPCOM trova e processa type library manifest.
5. Se ci sono nuovi componenti li registra.
 - a. XPCOM chiama Autoregistration Start.
 - b. XPCOM registra i nuovi componenti.
 - c. XPCOM chiama Autoregistration End.
6. XPCOM completa l'inizializzazione e notifica la fine operazione.

Il *Component Manifest* è un file utilizzato per la persistenza delle informazioni riguardanti i componenti presenti nel sistema. Il file in questione si chiama `compreg.dat` e per ogni componente contiene:

- Dimensione.
- Locazione su disco.
- Class ID .
- ContractID.

Il componente Type Library Manifest è un file che si chiama `XPTI.dat` contiene locazione e path di tutte le librerie interne nel sistema. Il file precisamente contiene :

- Locazione di tutti i Type Library file.
- Mapping tra tutte le interfacce e le type library da essa definite.

Usando i dati in questi due file, XPCOM sa esattamente quali componenti sono stati installati, su quali interfacce si basano e la rappresentazione binaria di ogni interfaccia.

Chiarito il concetto di avvio di XPCOM, è possibile illustrare come riutilizzare tutti i componenti. Di seguito viene riportato un esempio di utilizzo di un componente da un programma Javascript.

```
1 var Cc = Components.classes;  
2 var rc = Cc["@mozilla.org/registry;1"];
```

Nella linea 2 usiamo il CID del componente per recuperarne una reference attraverso il component manager.

```
3 var Ci = Components.interfaces;
4 var rs = rc.getService(Ci.nsIRegistry);
```

Nella linea 4 usiamo il Service Manager per ottenere un servizio dall'oggetto creato.

Component.Classes è un oggetto che contiene una lista di reference agli oggetti factory dei componenti XPCOM, che sono stati precedentemente installati e registrati nel sistema. Una volta recuperate la reference al factory attraverso il Componente Manager, i componenti possono essere istanziati o usati come servizio. Le proprietà del *Component.Classes* sono indicizzate con *ContractID*, altrimenti avendo a disposizione il CID utilizzando il metodo *Components.classesByID* .

E' possibile verificare anche l'esistenza di componenti come riportato di seguito.

```
if (!("@some/bogus/class;1" in Components.classes))
    // do something...
```

A questo punto abbiamo una reference al componente, e bisogna decidere se utilizzarlo come nuova istanza o servizio, richiamando sul factory uno dei seguenti metodi:

```
createInstance();
getService();
```

Determinare se un componente deve essere istanziato o usato come servizio non è possibile a run-time, solitamente lo si può determinare analizzando la documentazione.

Il *Component.Interfaces* usato nell'esempio precedente alla linea 3 è un oggetto che contiene tutte le interfacce disponibili nel sistema definite sui file *.idl* e etichettate come *scriptable* .

Accedere a dei componenti da C++ non è molto diverso anche se non si coinvolge la tecnologia XPCONNECT. Segue un esempio:

```
nsCOMPtr<nsIServiceManager> servMan;  
nsresult rv=NS_GetServiceManager(getter_AddRefs(servMan));
```

Poi è possibile richiedere un servizio:

```
rv  
=servMan->GetServiceByContractID("@mozilla.org/cookieManager",  
NS_GET_IID(nsICookieManager), getter_AddRefs(cookieManager));
```

Come evidenziato dai precedenti esempi utilizzare componenti è molto semplice e può essere fatto senza molte difficoltà. Il servizio XPCONNECT opera in maniera assolutamente trasparente lasciando la possibilità di riutilizzare facilmente tutti i componenti XPCOM.

2.3 XULRUNNER

XULRunner è un pacchetto runtime Mozilla, può essere utilizzato per l'avvio di applicazioni XUL + XPCOM, anche ricche come Firefox e Thunderbird. Fornisce il meccanismo per installare, aggiornare e disinstallare tali applicazioni. XULRunner fornisce inoltre libxul, una soluzione che permette di incorporare le tecnologie Mozilla in altri programmi.

L'obiettivo di XULRunner è fornire una piattaforma per il rilascio di applicazioni XUL (in particolare Firefox e Thunderbird) e soprattutto fornire un meccanismo per incorporare componenti. Sostanzialmente, XULRunner, può essere considerata come l'evoluzione naturale di Mozilla Application Suite. Attualmente, la piattaforma è ancora in fase di sviluppo e non è ancora ben chiaro il suo ruolo, come verrà rilasciata una versione definitiva e quando. La versione disponibile è ancora una versione di anteprima per sviluppatori, quindi per rilasciare applicazioni Mozilla consiglia che XULR venga rilasciato privatamente insieme ad ogni applicazione.

Le seguenti funzionalità sono già state implementate:

- Funzionalità di Gecko.

- XPCOM.
- Funzionalità di rete.
- Il motore di rendering Gecko.
- Supporto alle modifiche e alle transazioni DOM (senza interfaccia utente).
- Crittografia.
- XBL (XBL2 in futuro).
- XUL.
- SVG.
- XSLT.
- XML Extras (XMLHttpRequest, DOMParser, etc.).
- Web Services (SOAP).
- Supporto all'aggiornamento automatico (ancora incompleto).
- Barra di ricerca nel documento.
- Cronologia.
- Supporto per l'accessibilità.
- Servizi IPC per la comunicazione tra applicazioni basate su Gecko (ancora incompleto).
- Interfacce storage/sqlite (per ora, non è attivo per default).
- Funzionalità dell'interfaccia utente.

I seguenti elementi dell'interfaccia utente sono forniti da XULRunner, ma possono essere sovrascritti da altri programmi quando necessario:

- Le API e l'interfaccia utente per installare, disinstallare e aggiornare le applicazioni XUL.
- Extension Manager.
- File picker (utilizza il filepicker nativo del sistema).
- Barra di ricerca.
- Finestre di dialogo e interfaccia utente dell'Help.
- Interfaccia utente per la sicurezza (SSL, etc).
- XULRunner 1.8.0.4 è una developer preview stabile, basata sullo stesso codice di base di Firefox 1.5.0.4.

XULRunner non fornisce:

- Interfaccia utente per i bookmark e la cronologia (deve essere creata con l'applicazione).
- XForms (sarà disponibile con un'estensione).

XULRunner è un piattaforma che permette l'installazione di applicazioni basate sulle tecnologie Web standard come XUL, XHTML, CSS, Javascript. Installando un'applicazione su XULRunner sfruttiamo il rendering di Gecko e possiamo lanciarla con tutti i privilegi e la possibilità di riutilizzare tutti i componenti Xpcom. Quindi abbiamo la possibilità di accedere a file locali sia in lettura che in scrittura, possiamo comunicare attraverso la rete sfruttando HTTP e importare componenti esterni sfruttando Xpcom.

3 Firefox Extension

Nel capitolo precedente è stata illustrata l'architettura di Firefox, con particolare evidenza al suo sistema di gestione e condivisione dei componenti. In questo capitolo invece, analizzeremo le Firefox Extension. Ne illustreremo l'architettura e le principali tecnologie coinvolte, in modo da capire come un'estensione possa interagire con il browser e in particolare usare i componenti Gecko.

Il motivo che ha spinto centinaia di sviluppatori a creare una Firefox Extension, è stato sicuramente il fatto di poter riusare con semplicità tutti i componenti Gecko e le sue interfacce. Sviluppare una Firefox Extension implica l'uso di principi simili a quelli necessari per sviluppare estensioni per altri applicativi basati su Gecko ed XPCOM, come Thunderbird, Seamonkey, and Flock [20].

Sviluppare un'estensione significa aggiungere nuova logica alle applicazioni Mozilla, a differenza dei plugin che visualizzano specifici contenuti multimediali. Le estensioni permettono di personalizzare l'applicazione, adattandola alle necessità di navigazione degli utenti, aggiungendo nuove

funzionalità o modificando quelle già esistenti. Ad esempio, è possibile modificare il sistema tradizionale di gestione del download o creare nuovi strumenti per sviluppatori Web.

3.1 Introduzione alle estensioni

Come detto in precedenza sviluppare un'estensione coinvolge più tecnologie. E' inoltre necessario rispettare una certa architettura per permettere all'estensione di interagire con l'applicazione.

Uno dei concetti chiave per sviluppare un'estensione Firefox è capire cosa sono i componenti Chrome. Per elementi Chrome ci si riferisce a una serie di elementi dell'interfaccia utente che non appartengono all'area dei contenuti, ma all'area riservata ai componenti del browser, quindi ad esempio toolbar, menubar, sidebar, cioè elementi propri del browser.

Nella documentazione ufficiale Mozilla viene utilizzato il termine "Supplier of chrome", riferendosi a qualunque componente fornisca elementi aggiuntivi all'interfaccia dell'applicazione[23]. Un supplier, a sua volta è composto da "providers" e crea nuovi elementi da aggiungere all'interfaccia.

Tipicamente, un'estensione è considerata un supplier e viene modellata suddividendola in diversi provider i quali si dividono i compiti. Ne avremo uno che si occupa della creazione dei componenti grafici e del loro comportamento, un altro fornisce lo stile dei componenti ed infine un terzo che si occupa di fornire i testi localizzati nella lingua impostata nel browser.

Più formalmente esistono tre tipi di "chrome providers" che sono:

- Content: contiene solitamente le dichiarazioni degli elementi e degli script che ne gestiscono il comportamento. Le dichiarazioni sono contenute in file XUL e Javascript. Partendo da questo provider è possibile creare anche altre sub-directory contenenti elementi e altri componenti dell'applicazione.
- Skin: contiene elementi per la formattazione dei contenuti. Contiene CSS e immagini che definiscono il look dell'applicazione. In alcuni casi

ci si riferisce a questo package con il termine “Theme”. Difatti, modificando il package Skin è possibile cambiare il tema dell’applicazione, lasciando invariata la parte dei contenuti. Il package Skin può essere suddiviso in differenti sub-directory supportando la presentazione per diversi componenti. Ad esempio, è possibile creare una sub-directory contenente i file che definiscono il tema per tutti i componenti del browser ed una directory differente per ogni componente che abbia un tema specifico.

- **Locale:** contiene file per la localizzazione dell’applicazione. All’interno della directory principale è presente una sub-directory per ogni lingua supportata nell’applicazione. Il nome della sub-directory deve indicare il codice della lingua. Ad esempio, una sub-directory en-US conterrà informazioni per la localizzazione in inglese US. Ogni sub-directory contiene quindi tutti i testi dell’applicazione in una specifica lingua. I testi sono contenuti all’interno di due file, un DTD e uno properties. Il primo definisce delle entità che possono essere incluse all’interno dei file XUL, e l’altro definisce stringhe che possono essere importate facilmente dagli script.

Possiamo dire che tipicamente un’estensione è costituita dai tre tipi di providers Chrome, che collaborano per la creazione di un nuovo elemento da aggiungere all’interfaccia globale. Questa è la proposta Mozilla che non vincola assolutamente gli sviluppatori, comunque liberi di modellare i componenti come ritengono più opportuno. Ovviamente rispettare l’architettura dei componenti Chrome offre molti vantaggi, come ad esempio la possibilità di localizzare con facilità la propria applicazione e di cambiare semplicemente il tema.

Oltre all’organizzazione dei providers è necessario segnalare al browser l’esistenza della nostra estensione e la sua organizzazione. Per farlo è sufficiente creare il file *chrome.manifest*, il quale verrà letto dall’applicazione Chrome registry e provvederà a caricare tutti i providers segnalati nel manifest.

L’organizzazione tipica dei file di un supplier è questa:

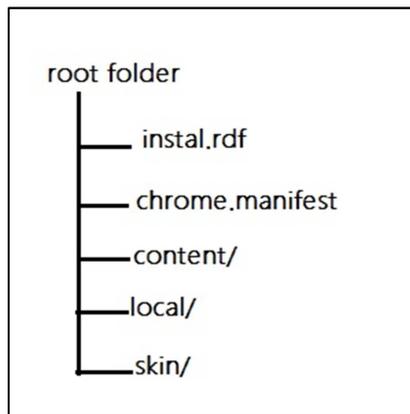


Figura 5: Struttura cartella root

In Figura 5, è presente anche il file *install.rdf*, utilizzato per fornire informazioni d’installazione e configurazione di un’estensione. La personalizzazione di questo file verrà trattata più approfonditamente in seguito.

Riassumendo, il sistema di organizzazione dei componenti viene chiamato chrome system. Definisce un sistema flessibile per l’organizzazione e la registrazione dei providers ma soprattutto è il sistema che permette alla nostra estensione di essere caricata all’interno dell’applicazione centrale e di interagire con essa.

3.2 Chrome

La parola Chrome all’interno dei sistemi Mozilla storicamente ha più significati e ci si riferisce a concetti differenti tra loro [22]. Con la parola “Browser Chrome” ci si riferisce all’intera interfaccia grafica di Firefox, cioè tutta l’area intorno alla zona di visualizzazione dei contenuti. Con la parola “Chrome URL” ci si riferisce ad un URL che utilizza uno schema Chrome. In questo paragrafo con la parola Chrome ci riferiremo al “sistema Chrome” inteso come sistema di registrazione dei providers e di organizzazione di un’estensione.

Come detto nel Capitolo 1, l’intera interfaccia grafica dei sistemi Mozilla è stata sviluppata in XUL, linguaggio interpretato da Gecko allo stesso modo di HTML, SVG e molti altri. Questo significa che è possibile caricare contenuti

remoti in uno di questi linguaggi e visualizzarne il rendering all'interno dell'area di visualizzazione.

Per ragioni di sicurezza, i contenuti caricati da remoto hanno permessi limitati e non possono accedere a tutti i componenti di Gecko. Per questo motivo, Mozilla fornisce la possibilità di installare localmente i contenuti e caricarli come sistemi Chrome, ottenendo tutti i privilegi. Così è permesso accedere a componenti e servizi XPCOM, file locali, cronologia e tutto ciò che viene condiviso nel sistema. Nel caso in cui si volessero concedere i privilegi ad un documento remoto, sarebbe necessario fornire un certificato e l'utente dovrebbe fare da garante per poter eseguire le operazioni [20].

Per ottenere tutti i permessi, non è sufficiente sviluppare un sistema come un sistema Chrome, ma bisogna fare in modo che il browser lo riconosca come tale. Per questo è necessario caricare i providers tramite URL Chrome.

Dato che, un URL Chrome può fare riferimento solo a elementi registrati, la registrazione dei singoli elementi diventa imprescindibile. Quindi possiamo affermare, che per ottenere i permessi è necessario registrare i providers della propria applicazione e successivamente invocarli con un URL Chrome.

La registrazione Chrome non è solo il sistema necessario per ottenere tutti i permessi, ma è anche il sistema con cui è possibile installare i contenuti localmente ed estendere Firefox. All'utente sembrerà che l'estensione ha modificato l'interfaccia Firefox, ma il codice in realtà è separato. Con XUL e Chrome Overlay è possibile identificare un punto preciso dell'interfaccia generale di Firefox ed estenderlo con nuovi contenuti.

Tutte le interfacce di Mozilla in realtà sono un insieme di pacchetti chrome contenenti XUL, Javascript e CSS. I file sono accessibili tramite URL Chrome ottenendo quindi tutti i privilegi. Naturalmente, i pacchetti che compongono il browser sono molto più e complessi rispetto ad una normale estensione. E' possibile esaminare questi pacchetti nella cartella di installazione di Firefox.[20]

3.2.1 Chrome URL

Per installare un'applicazione chrome e quindi godere conseguentemente di tutti i privilegi, è necessario referenziare i file e i providers con delle URL Chrome. Utilizzando le URL chrome godiamo anche di un altro vantaggio importante, i pacchetti local e skin verranno selezionati automaticamente in base alle preferenze impostate dall'utente nel browser. Ad esempio, possiamo fare riferimento ad un particolare file del tema senza sapere quale tema stiamo utilizzando. L'importante è che il file abbia lo stesso nome in tutti i temi disponibili. Questo significa che non è necessario sapere dove si trova per potervi accedere. Mozilla si occuperà di calcolare dove è situato esattamente il file e di recuperarlo. Scrivere applicazioni con un sistema di riferimenti chrome rende più semplice anche la localizzazione dell'applicazione. Infatti sarà Mozilla a selezionare di volta in volta la sub-directory contenente i testi dell'applicazione.

La sintassi base di una URL chrome è la seguente:

```
chrome://<package name>/<part>/<file.xul>
```

Il *package name* è il nome del package a cui si vuole fare riferimento , *part* è il nome del providers contenuto nel package ed infine il nome del file a cui riferirsi. Ad esempio:

```
chrome://reader/skin/img.jpg
```

Con questo URL ci si riferisce ad un file img.jpg situato nel provider skin del supplier reader.

Per riferirsi ad una sub-directory è sufficiente aggiungerla alla fine dell'URL.

```
chrome://reader/skin/sub/img.jpg
```

Nell'esempio precedente il file è situato nella sub-directory sub all'interno del providers skin.

Quando si apre un URL chrome, Mozilla controlla la lista di applicazioni chrome installate nel sistema, recupera il file di registrazione e con questo dispone di tutte le informazioni per poter calcolare la posizione esatta del file. Quindi, una URL chrome può essere utilizzata solo per localizzare file e providers di applicazioni chrome istallate nel sistema e registrate.

E' possibile utilizzare chrome URL nello stesso modo con cui si utilizzano le URL HTTP, inserendola direttamente nella URL Bar di Firefox o utilizzarli all'interno delle applicazioni.

Nel caso in cui si utilizzi un URL dove non è indicato il nome del file, verrà selezionato automaticamente il file appropriato. Segue un esempio:

```
chrome://reader/content/
```

Nell'esempio precedente sarà selezionato automaticamente il file XUL con lo stesso nome dell'applicazione cioè *reader.xul*. Nel caso in cui si fa riferimento ad un package skin verrebbe selezionato *reader.css* e nel caso di locale verrebbe selezionato il file *reader.dtd*.

In definitiva possiamo dire che le URL Chrome non fanno riferimento a locazioni precise sul disco, ma sono dei riferimenti a providers di applicazioni chrome. Gli URL possono essere risolti solo se l'applicazione è stata precedentemente installata nel sistema.

3.2.2 Registrazione Chrome

L'operazione di registrazione serve a segnalare l'esistenza di un'applicazione chrome e la localizzazione sul disco dei suoi providers.

Il servizio che si occupa della registrazione, si chiama Chrome registry ed è eseguito dal runtime di Gecko. E' configurabile e persistente. Offre la possibilità di installare differenti providers e selezionare in automatico skin e local preferiti. Per informare il Chrome Registry della presenza di un'applicazione chrome e della localizzazione dei suoi providers, si utilizza il file *chrome.manifest* posizionato nella cartella radice dell'estensione. Questo significa, che per installare una nuova estensione, dobbiamo inserire un file

manifest all'interno della chrome directory dell'utente o della directory chrome centrale.

Il testo contenuto nel file è basato su un sistema di linee di testo. Ogni linea, analizzata indipendentemente dalle altre, rappresenta un'informazione individuale per il chrome registry. Una linea del manifest dichiara un provider e ne mappa la locazione sul disco.

Il file manifest viene esaminato ogni volta che l'applicazione Gecko viene invocata, questo per verificare l'esistenza di nuovi pacchetti e applicazioni.

Una linea del file manifest ha la seguente sintassi:

```
<providername> <packagename> <filepath>
```

Providername è il nome che vogliamo assegnare al provider, seguito dal nome del package che in questo caso si riferisce all'applicazione ed infine il path per la localizzazione del provider sul disco.

All'interno dei file manifest è anche possibile dichiarare l'overlay con la parola chiave overlay. Segue un esempio:

```
overlay chromeURL chromeURL
```

Una dichiarazione di overlay permette di fare il merge tra due file XUL. E' il sistema utilizzato per estendere l'interfaccia Firefox, così possiamo dichiarare il file contenente gli elementi grafici da aggiungere all'interfaccia centrale. Si tratterà questo argomento in maniera più approfondita nei successivi paragrafi.

E' possibile usare anche URL Jar per indicare providers contenuti all'interno di archivi jar. Infatti, come vedremo in seguito nel paragrafo dedicato al packaging, una tecnica molto diffusa, è quella di distribuire applicazioni in cui i provider sono interamente contenuti all'interno di un archivio Jar. In questo caso, è necessario indicare lo schema Jar per definire il path all'interno di un archivio. Ad esempio:

```
jar:<jar name>!/<path_in_archive>
```

Come indicato nell'esempio è necessario indicare il nome del Jar e successivamente il path all'interno dell'archivio Jar.

In Javascript è possibile sovrascrivere funzioni proprietarie. Aggiungendo `xpcnativewrappers=yes` alla fine di una linea, si indica che un gruppo di script non può sovrascrivere le funzioni proprietarie. Questa opzione viene aggiunta per evitare problemi, in quanto alcune estensioni potrebbero sovrascrivere delle funzioni proprietarie, impedendo alle altre di lavorare correttamente.

Per i provider Skin e Locale è necessario specificare che tipo di provider stiamo dichiarando. Essi hanno una sintassi particolare e simile a quella generale degli altri provider. Ne segue un esempio

```
skin browser classic/1.0 jar:classic.jar!/skin/classic/browser/  
locale browser en-US jar:en-US.jar!/locale/browser/
```

Negli esempi sopra indicati vediamo che tra la dichiarazione nomeprovider e URL, c'è una parola chiave che indica il tipo particolare di skin o di locale. Questo permette a Mozilla di riconoscere la versione e lingua supportata.

3.2.3 Chrome Package

Ogni estensione è un pacchetto stand-alone con un chrome-URL separato e contiene i file che ne descrivono la sua interfaccia grafica, il suo comportamento e il suo stile grafico.

Per chrome package ci si riferisce alla directory d'installazione, dove sono contenuti tutti gli elementi della nostra estensione e il file manifest necessario per la registrazione.

Ogni provider normalmente è situato in una directory differente all'interno della cartella root, che normalmente viene chiamata chrome directory. I provider possono includere differenti file, ognuno di questi accessibile da un differente chrome URL.

Il sistema è estremamente flessibile e permette di includere tutte le parti di cui si ha bisogno, non necessariamente tre come nell'organizzazione tradizionale. Questo permette di modellare la propria applicazione come lo si ritiene più opportuno.

A differenza di come elencato precedentemente dove tutti i package sono contenuti nella cartella Chrome, è possibile aggiungere package situati in una qualunque locazione del file-system. L'importante è registrarlo correttamente nel file manifest e usare la directory chrome come proxy.

3.2.4 Chrome overlay

L'operazione di overlay permette di condividere elementi dell'interfaccia grafica e di dividere in più parti un unico file XUL. Le singole parti del documento XUL sono contenute in file separati e sono definiti file di overlay. Insieme definiscono un unico file XUL.

Attraverso il meccanismo di overlay possiamo definire elementi UI che verranno aggiunti all'interfaccia generale come risultato di un update, quindi può essere considerato come un meccanismo generale per:

- Aggiungere elementi all'interfaccia grafica.
- Sovrascrivere piccoli pezzi di XUL senza dover modificare l'interfaccia generale.
- Riusare particolari pezzi di interfaccia UI.

XUL file e overlay lavorano insieme per descrivere un unico singolo Master Document. Non ci sono restrizioni relative a cosa deve essere definito nel Master Document e cosa invece deve essere definito nei documenti di overlay, il sistema semplicemente prevede un merge tra i contenuti.

Quando plugins, estensioni o altre applicazioni forniscono nuovi elementi da aggiungere all'interfaccia generale di Firefox, devono essere definiti all'interno di un file di overlay [24].

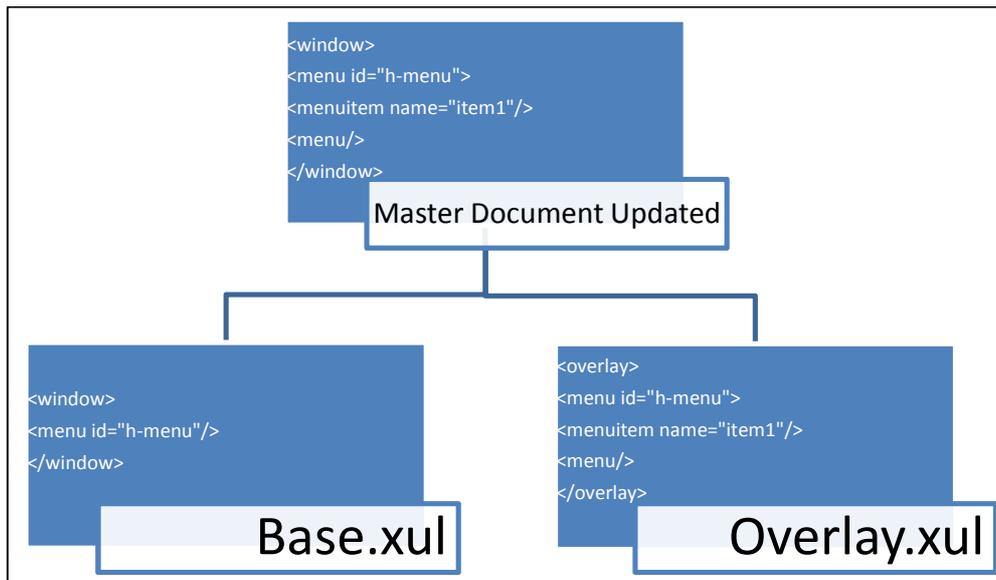


Figura 6: Overlay

L'update del Master document è il risultato del merge eseguito dal layout engine, tra il documento base e i file di overlay. Il punto in cui saranno inseriti gli elementi definiti nel file di overlay è determinato da un sistema di riferimenti di id.

Come possiamo vedere nel diagramma in Figura 6, se un attributo ID del documento overlay, corrisponde ad un ID del documento base, i figli dell'elemento del overlay saranno inseriti nella struttura del Master.

L'overlay dei documenti può essere eseguito anche in assenza di ID. In questo caso il documento di overlay viene semplicemente aggiunto alla fine del documento base. La vera forza del sistema di overlay viene dagli ID. Le regole per il merge basato su ID permettono di determinare con precisione il punto in cui si vogliono inserire i contenuti. Quando vengono identificati due ID uguali, l'operazione di merge eseguita dal layout engine segue i seguenti passi[25]:

1. Vengono identificati gli elementi del base e dell'overlay che hanno corrispondenza di ID
2. Il contenuto dell'elemento nell'overlay viene appeso in coda ai contenuti dell'elemento base

3. Tutti gli attributi dell'overlay vengono aggiunti al base. Nel caso in cui ci siano gli stessi attributi quelli del base vengono sovrascritti con quelli del overlay.

Il primo e il secondo punto aggiungono contenuti al documento base, mentre nel punto 3 può avvenire overriding di attributi come per esempio attributi di style.

Il sistema di overlay può essere utilizzato più volte partendo da un solo documento base. Quindi, potremo avere un documento Master, che è stato prodotto dal merge tra un base document e più overlay document. In questo caso il processo di merge non cambia, viene aggiunto un overlay per volta, ottenendo quindi Master document intermedi, fino ad arrivare all'esaurimento degli overlay document[25].

L'overlay può essere di due tipi, esplicito o dinamico. Questo determina il modo con cui vengono identificati i file di overlay da sovrapporre al Master Document.

L'overlay esplicito verrà eseguito ogni volta che il layout engine processerà il file base. Questa tecnica è molto utilizzata per dividere un singolo documento in più parti e poter riutilizzare tutte la parti anche in altri documenti.

Inserendo la seguente istruzione all'interno del documento base, il parser di Gecko caricherà il contenuto del file overlay ed eseguirà il merge.

```
<?xul-overlayhref="chrome://component/content/overlay.xul"?>
```

L'overlay dinamico è reso possibile dall'operazione di chrome registry. Mozilla consultando un RDF datasource, contenente informazioni sui componenti ed estensioni, decide quali sono i file da sottoporre a overlay. Questo sistema prevede che entrambi i file base e overlay siano installati nel sistema chrome [25].

L'overlay dinamico è stato introdotto per aggiungere contenuti all'interfaccia base di Mozilla da parte di un'estensione o di un tema[25]. Per utilizzare questa tecnica è necessario specificare i file di overlay nel file chrome.manifest aggiungendo la seguente istruzione.

```
overlay <chrome uri base > <chrome uri overlay>
```

In questo modo quando l'applicazione verrà installata, saranno archiviate le informazioni di overlay all'interno di un apposito database.

Ogni volta che l'applicazione verrà riavviata sarà consultato il database per recuperare i file da sottoporre ad overlay, nel caso in cui un file di overlay non venisse trovato, l'operazione verrebbe saltata.

3.5 Localizzazione

Come detto precedentemente Chrome fornisce un meccanismo per la localizzazione delle applicazioni. Il testo localizzato deve essere memorizzato in due file differenti, un DTD ed un properties.

I DTD contengono dichiarazioni di entità, così, sfruttando le proprietà fornite dai documenti XML, è possibile importare elementi esterni all'interno di file XUL. I file properties invece vengono usati per definire stringhe di testo utilizzabili da script.

Per ogni lingua presente nel sistema è necessario creare una nuova directory, contenente i file per la localizzazione. Quindi, all'interno del provider locale è presente una directory per ogni lingua in cui è possibile localizzare l'applicazione.

Le cartelle contenenti i file per la localizzazione possono essere chiamate in uno dei seguenti modi:

- Utilizzando l'identificatore per il linguaggio; IT per l'italiano
- Utilizzando l'identificatore di una lingua e il paese; it-IT per l'italiano o en-US per l'inglese degli Stati Uniti.

Se all'interno della cartella locale sono inserite più localizzazioni, chrome sceglierà automaticamente quella che più si avvicina alle preferenze dell'utente. Inoltre, è necessario segnalare la presenza dei pacchetti locale all'interno del file chrome.manifest, aggiungendo la dichiarazione di un nuovo locale come segue:

```
locale NAMEExtension it-IT locale/it-IT/
```

Per la localizzazione con DTD è sufficiente dichiarare il DOCTYPE con l'URL Chrome della cartella locale. Il sistema Chrome sceglierà automaticamente la localizzazione che si avvicina di più alle esigenze dell'utente. Così, all'interno del documento XUL sarà possibile riutilizzare le entità definite nel DTD selezionato da sistema Chrome.

Invece con la localizzazione dei file Javascript è necessario usare i file properties. I testi all'interno del file vengono definiti come segue:

```
nomeString : testo
```

Per utilizzare le stringhe da uno script è necessario usare un componente *string-bundleset*, che permette di importare automaticamente le stringhe contenute nel file properties. Di seguito si riposta un esempio della dichiarazione dell'elemento nel file:

```
<stringbundleset id="stringbundleset">
<stringbundleid="string-bundle"src="chrome://extension/locale/f
ile.properties">
</stringbundleset>
var stringsBundle = document.getElementById("string-bundle");
var errorString = stringsBundle.getString("error");
```

La dichiarazione dell'elemento *string-bundleset* deve essere definita nel file xul, di conseguenza nel file Javascript possiamo recuperare l'elemento *string-bundle* tramite ID e utilizzare il suo metodo `getString` per recuperare le stringhe.

3.6 Distribuzione

La distribuzione di un'estensione può avvenire tramite un installer che permette al browser di installare l'estensione automaticamente, oppure si può distribuire l'estensione fornendo tutti i file e procedere all'installazione

manuale.

Ovviamente, la modalità più comoda per la distribuzione di un'estensione è sicuramente quella che permette l'installazione automatica. Comunque ci sono casi dove è effettivamente vantaggioso installare l'applicazione manualmente, come ad esempio durante la fase di sviluppo dell'applicazione.

Per lanciare un'installazione automatica è necessario che l'estensione sia in formato XPI. Quando il browser riceve un'estensione di questo tipo, lancia automaticamente la procedura di installazione attraverso l'add-on manager. Se si vuole distribuire l'estensione tramite un Web server e spingere all'installazione automatica, è necessario impostare correttamente il Web server. E' importante settare correttamente il content-type come application/x-xpinstall, questo permette al browser di riconoscere l'installer e di lanciare l'installazione automatica.

In ambedue i casi è necessario che l'estensione includa il file install.rdf, esso contiene informazioni importantissime per questa fase, come ad esempio l'ID applicazione che stiamo installando, le versioni di compatibilità e il nome dell'applicazione.

3.6.1 XPI

Mozilla fornisce un meccanismo, per impacchettare tutti i file che compongono un'estensione in un unico singolo installer, chiamato XPIInstall (XPI)[20].

XPIinstaller contiene un pacchetto jar, al cui interno si inseriscono i file che compongono la nostra applicazione. In aggiunta un installer può contenere anche uno script per personalizzare il processo di installazione.

Xpi installer è lui stesso un file jar con estensione XPI, che permette di distinguerlo dagli altri archivi. I passi che vengono eseguiti per l'installazione automatica sono i seguenti [20]:

- Connessione a una pagina Web contenente l'estensione XPI.

- Apertura di un dialog box che comunica all'utente l'installazione dell'applicazione. In caso di più installer all'utente verrà presentata una lista. L'utente a questo punto può accettare o rifiutare.
 - Se l'utente ha deciso di continuare, il pacchetto verrà scaricato.
 - Verrà estratto dall'archivio il file install.js ed eseguito.
 - Una volta terminato lo script, l'applicazione può considerarsi installata
- Come indicato l'installazione viene lanciata automaticamente collegandosi ad una pagina Web. Per permettere che l'installazione sia lanciata automaticamente è necessario configurare la pagina in modo che :

- L'header content-type sia settato come application/x-xpinstall .
- La pagina Web contenga uno script che lanci l'installazione richiamando il metodo install() dell'oggetto globale Installer, indicando l'URL del file XPI.

L'archivio XPI deve contenere un file Javascript install.js, utilizzato per la fase di installazione. I file che devono essere installati, cioè quelli che compongono l'applicazione, sono normalmente contenuti in un jar, con la normale struttura delle directory Chrome.

Il file install.js può controllare il processo di installazione. Ad esempio, può essere usato per verificare la versione dei file ed eventualmente eseguire solo un update del software.

Lo script di installazione deve necessariamente eseguire le seguenti operazioni[20]:

1. Inizializzare l'installazione, specificando quale package si sta installando e la sue versione.
2. Usare la funzione install() per definire quali file e quali directory devono essere installate.
3. Avviare il processo di installazione installando i file necessari.

E' importante notare che durante il secondo step si possono indicare quali file devono essere installati e che i file non saranno copiati fino allo step successivo.

Mozilla mantiene un file registry, il quale contiene informazione su tutti i componenti attualmente installati nel sistema. Quando un componente viene

installato, il registro viene aggiornato automaticamente. All'interno del registry sono contenute anche informazioni sulla versione dei file attualmente installati. Questo può essere sfruttato per controllare facilmente la versione delle estensioni.

3.6.2 Install.rdf

Il file `install.rdf` è il file usato dall'add-on manager per determinare informazioni sull'estensione installata. In questo file sono contenute informazioni utili a determinare la versione dell'estensione, la compatibilità con le varie versioni di Firefox e informazioni sullo sviluppatore. Il formato utilizzato per gestire queste informazioni è RDF/XML.

Rdf (Resource Description Framework) è lo strumento di base proposto dal W3C per la codifica, lo scambio e il riutilizzo di metadati strutturati. Consente l'interoperabilità tra applicazioni che scambiano informazioni sul Web.

Il principio alla base di Rdf, è che qualunque cosa da esso descritta è una risorsa. Ogni risorsa è identificata da un URI, in questo caso utilizzate come identificatori univoci di risorse. Il modello è formato quindi da risorse e valori che sono legate tra di loro da delle relazioni semantiche.

L'unità base è lo Statment che viene utilizzato per dichiarare una tupla Soggetto-predicato-oggetto dove il soggetto è una risorsa, il predicato è una proprietà e l'oggetto è un valore.

Il file `install.rdf` quindi è scritto in sistassi RDF/XML e ne segue un esempio.

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:em="http://www.mozilla.org/2004/em-rdf#">
  <Description about="urn:mozilla:install-manifest">
    <!-- properties -->
  </Description>
</RDF>
```

Come possiamo vedere, vengono definiti due spazi dei nomi, uno per gli elementi RDF e uno EM per l'Extension Manager.

Con lo spazio dei nomi EM si definiscono le proprietà dell'estensione. Alcune di queste sono opzionali mentre altre sono obbligatorie. In assenza di queste ultime l'applicazione non sarà installata. Segue una lista delle principali proprietà:

- Id: l'identificativo dell'estensione può essere di due formati:
 - GUID, composto da 5 gruppi di cifre esadecimali `xxxxxxxx-xxxx-xxxx-xxxxxxxx` e può essere generato usando degli appositi tool.
 - Una stringa formattata come `extensionname@example.org` più semplice da imparare da generare e da manipolare.
- Version: una stringa che identifica la versione dell'estensione fornita. Il formato solitamente utilizzato è la versione Toolkit version format. In questo formato la stringa è costituita da una o più parti separate da un punto (`<em:version>2.0<em:version>`). Ogni parte della versione è formata da una sequenza di 4 sottoparti:

`<NumeroA>< StringB>< NumberC>< StringD>`

Ognuna di queste parti è opzionale ed i numeri sono interi in base 10. E' possibile aggiungere alcuni caratteri speciali per la compatibilità all'indietro e la leggibilità delle sigle come ad esempio:

- `1.5.0.*` significa infinito
- `1.0+` tutte le versioni successive alla 1.0

Le due stringhe vengono confrontate da sinistra verso destra. Se manca un'intera parte, viene considerata 0. Le parti 'a' e 'c' sono confrontate come numeri, mentre le parti 'b' e 'd' sono confrontate batwise essendo stringhe. Se una parte non esiste, è sempre minore di una parte che esiste.

- `1 == 1. == 1.0 == 1.0.0`
- `.1a < 1.1aa < 1.1ab < 1.1b < 1.1c`
- `1.1pre1a < 1.1pre1aa < 1.1pre1b < 1.1pre1`

Se l'estensione non usa un formato valido non verrà installata.

- **Type:** un valore intero che rappresenta il tipo di applicazione fornita:
 - 2= Extension
 - 4=Theme
 - 8=Locale
 - 32 =Multiple item package
- **Target Application:** serve a specificare la versione su cui installare l'estensione. Questo elemento conterrà gli elementi `id`, `minversion`, `maxversion`, definiti come tipi `version` precedentemente illustrati. Permette all'autore di specificare su quali versioni di Firefox è possibile installare la propria estensione. Ad esempio, se l'estensione è compatibile solo con le versioni più vecchie di Firefox 3, è possibile specificare l'elemento `maxversion=3.0.*`. E' possibile inserire anche più di un elemento `target application` in modo da poter definire più versioni di compatibilità. Si riporta un esempio di `target application`:

```
<em:targetApplication>
  <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>1.5</em:minVersion>
    <em:maxVersion>3.0.*</em:maxVersion>
  </Description>
</em:targetApplication>
```

- **Name:** permette di definire il nome dell'estensione, verrà utilizzato per essere visualizzato in diversi punti dell'add-on manager.

Questo elenco comprende solo gli elementi obbligatori, senza i quali l'installazione non verrebbe eseguita. Esiste una lunga serie di elementi EM opzionali che permettono di definire informazioni sull'autore, sulla descrizione dell'applicazione, URL dell'home page e molti altri dati non fondamentali per il processo di installazione.

4 Ambiente di sviluppo

Quando inizia un nuovo progetto software è necessario preparare correttamente l'ambiente di lavoro, in modo da avere a propria disposizione tutti gli strumenti utili per il processo di sviluppo. Solitamente per predisporre l'ambiente di sviluppo è sufficiente installare un IDE(Integrated Development Environment) che contiene tipicamente un editor, un tool per il debug, un compilatore e altri strumenti.

Attualmente non esiste un IDE contenente tutti gli strumenti necessari per lo sviluppo di una Firefox Extension, ma esistono molti strumenti che possono aiutare in alcune fasi del processo di sviluppo. Questi strumenti non compongono un ambiente di sviluppo completo, devono essere usati singolarmente, in maniera indipendente l'uno dall'altro. Molti di questi sono ereditati dalla programmazione Web.

In questo capitolo si illustreranno gli strumenti utilizzati per lo sviluppo di WebRecorder e sufficienti allo sviluppo di qualunque altra Firefox Extension. E' stato utilizzato Komodo, un editor di testo che permette di scrivere codice Javascript, XML, XUL, Rdf ed inoltre automatizza la fase di packaging di

un'estensione. Un altro strumento importante è il debugger Javascript in quanto tutta la logica dell'applicazione viene scritta in questo linguaggio. E' quindi fondamentale avere a disposizione uno strumento che permetta di controllare l'esecuzione del codice run-time.

4.1 Impostare Firefox come ambiente di sviluppo

Oltre ad installare un editor ed un debugger, è importante impostare Firefox correttamente e prepararlo per lo sviluppo ed il test dell'applicazione. Infatti, Firefox verrà utilizzato come ambiente di esecuzione e test, in quanto l'unico modo di eseguire una Firefox Extension è quello di mandarla in esecuzione insieme al browser.

Firefox può essere impostato in modo da dare più informazioni riguardo errori, eccezioni e attività svolte dall'applicazione. Per non rallentare l'esecuzione del browser e compromettere dati personali, è consigliato creare un profilo dedicato allo sviluppo e al test dell'applicazione.

E' possibile lanciare più di un'istanza di Firefox allo stesso tempo, usando il parametro `-no-remote`. Ad esempio il seguente comando apre Firefox con il profilo "dev".

```
ProgramFiles\MozillaFirefox\firefox.exe -no-remote -P dev
```

Per la creazione di un nuovo profilo è necessario avviare Firefox con il parametro `-ProfileManager` e verrà aperta una dialog box per la gestione dei profili. Da qui è possibile creare un nuovo profilo e scegliere la directory d'installazione. Ricordiamo che questa sarà la directory che conterrà la nostra estensione, infatti è importante installarla in un posto di rapido accesso.

Oltre alla gestione dei profili, è importante impostare alcune preferenze di Firefox, in modo da avere più informazioni sulle attività della nostra estensione e di effettuare alcune attività basiche di debug.

Per entrare nella pagina d'impostazione delle preferenze è necessario digitare `about:config` nella barra degli indirizzi e premere invio. Nella pagina

visualizzata vengono elencate tutte le preferenze contenute nei file *pref.js* e *config.js*. Ogni preferenza è espressa da un variabile, che può essere intera, stringa o booleana [26].

Nella guida Firefox Developer viene consigliata la modifica delle seguenti preferenze:

- `javascript.options.showInConsole = true`: attiva il login degli errori nei file chrome nella console degli errori.
- `nglayout.debug.disable_xul_cache = true`: disabilita la cache XUL, quindi dopo le modifiche di file XUL non è necessario riavviare l'applicazione.
- `browser.dom.window.dump.enabled = true`: abilita l'uso della funzione `dump()` e permette di stampare nella console di default.
- `javascript.options.strict = true`: abilita warning nella console. Quando questa preferenza viene abilitata può succedere di visualizzare una lunga lista di warning prodotti da tutte le altre estensioni Firefox. E' possibile filtrare i warnign impostando adeguatamente i filtri nella console.
- `extensions.logging.enabled = true`: invierà informazioni più dettagliate sull'installazione alla console.
- `nglayout.debug.disable_xul_fastload = true`.
- `dom.report_all_js_exceptions = true`: permette di visualizzare in console tutte le eccezioni Javascript.

Una delle funzionalità più utili è quella che attiva l'uso della console, cioè quella che attiva la funzione `dump()`. Per visualizzare la console è necessario avviare Firefox con il parametro `-console`. Sviluppando un'estensione Firefox, abbiamo tutti i privilegi Chrome e ci viene permesso di utilizzare anche dei metodi più sofisticati per l'accesso alla console principale di Firefox. Possiamo utilizzare `Components.utils.reportError` e `nsIConsoleService` per loggare messaggi di errore e warning nella console principale che può essere aperta da Ctrl-Shift-J.

4.2 Komodo Edit

Komodo edit è l'editor utilizzato per lo sviluppo di WebRecorder. E' un editor open source e offre supporto per tutti i linguaggi di programmazione necessari allo sviluppo di un'estensione Firefox. Esiste anche una versione distribuita a pagamento, Komodo IDE, che include supporto per attività di debug, subversion, gestione database e altre funzionalità avanzate.

Una caratteristica importante di Komodo Edit è la gestione di progetti per Firefox Extension. Automatizza alcuni fasi del processo di sviluppo e offre una serie di funzionalità molto utili tra cui:

- Supporto di tutti i linguaggi programmazione necessari allo sviluppo dell'estensione.
- Creazione automatica delle directory Chrome necessaria alla creazione di un'applicazione Chrome.
- Creazione automatica e aggiornamento del file chrome.manifest.
- Interfacce grafiche per la configurazione del file install.rdf.
- Funzionalità di build automatico utile per la creazione del pacchetto XPI necessario alla distribuzione dell'estensione.

Nell'immagine in Figura 7 si riporta uno screenshot dell'applicazione. Nella sidebar sono presenti i file dell'applicazione disposti secondo l'organizzazione delle directory Chrome.

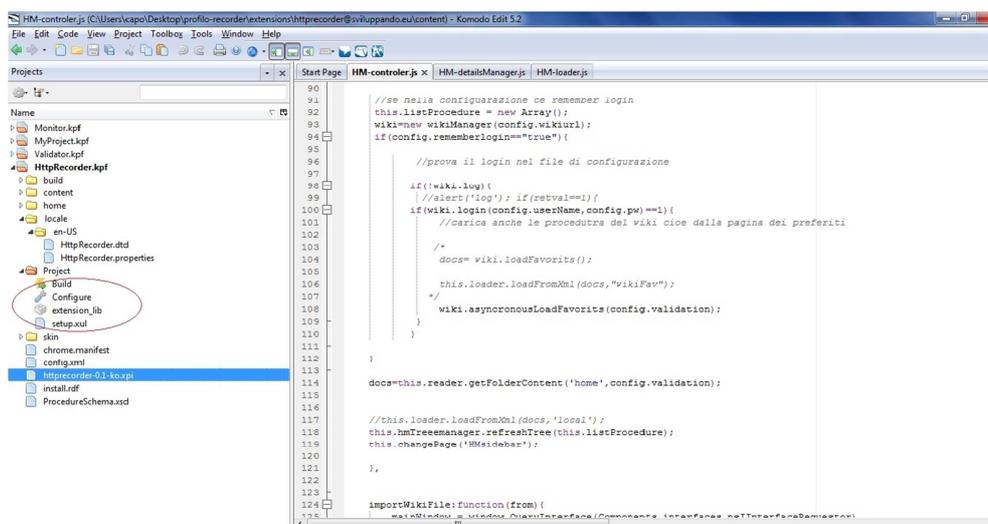


Figura 7: Screen Shot Komodo

All'interno di ogni progetto è presente una directory Project contenente i file che Komodo usa per gestire il nostro progetto.

Come possiamo vedere nell'immagine 1, nella directory Project sono presenti quattro comandi. Il primo, Build, serve a creare direttamente l'installabile XPI per la distribuzione dell'applicazione. Il secondo, Configure, permette di configurare il file install.rdf tramite una dialogbox mostrata in Figura 8.

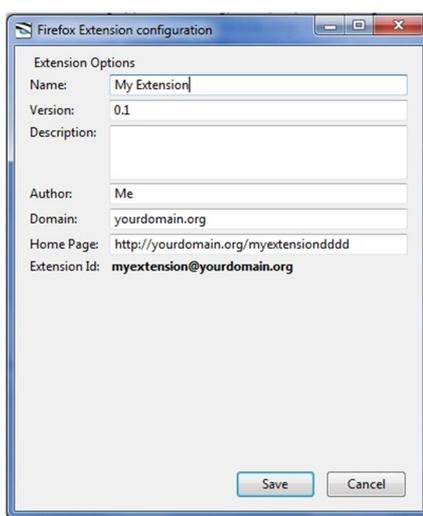


Figura 8: Configuration Extension

Extension Lib serve a gestire le librerie importate nell'estensione.

Come detto in precedenza, la funzione build di Komodo crea in automatico il pacchetto installabile XPI. Per questo i path all'interno del file chrome.manifest, mappano l'organizzazione del pacchetto XPI e non la reale disposizione delle cartelle nel progetto. Potrebbe essere comodo rifare il mapping del chrome.manifest secondo la distribuzione reale della directory di lavoro, ed installare il progetto direttamente nella directory Extension di Firefox. Così non sarà necessario eseguire il build ad ogni modifica dell'estensione, ma semplicemente salvare il file e ricaricare l'estensione all'interno del browser.

Questo è molto importante dato che la versione di base Komodo non offre strumenti per il debug di un'estensione. L'unico modo per eseguire il test di un'estensione è quello di eseguirla avviando il browser. Infatti, come vedremo meglio nel seguente paragrafo, utilizzeremo Firefox e alcune estensioni per il debug dell'applicazione.

4.3 Javascript Debugger

Javascript è conosciuto come uno dei linguaggi più difficili su cui eseguire il debug, a causa della sua natura dinamica [3]. Javascript è un linguaggio di script che viene interpretato dal browser senza essere compilato. Questo comporta l'eliminazione della fase di compilazione, dove normalmente vengono scoperti buona parte degli errori. Inoltre, è un linguaggio di programmazione non tipizzato, caratteristica utile quanto pericolosa, spesso causa di errori difficili da individuare.

Per questo è necessario utilizzare uno strumento efficiente per il debug che permetta di osservare il programma durante la sua esecuzione nel browser. Le normali tecniche di debug a disposizione di un programmatore Javascript, si limitano alla possibilità di eseguire stampe sulla console, di aprire alert durante l'esecuzione del codice e utilizzare il metodo `document.write` per eseguire delle stampe nel documento [3]. Tutte queste soluzioni sono utili ma non sufficienti ad eseguire il debug su un programma complesso composto da migliaia di linee di codice.

Esistono diversi strumenti per il debug Javascript, molti di questi possono essere installati nel browser come estensione e permettono di osservare l'esecuzione del codice. Un problema tipico del debug di un'estensione, è che non tutti i debugger permettono di osservare l'esecuzione del codice contenuto all'interno di una Firefox Extension, in quanto interpretato come parte integrante del browser. Uno strumento che evita questo problema è il debugger Javascript Venkman, dato che permette di lavorare anche sui file del browser stesso.

Nella Figura 9 viene riportato uno screenshot del debugger Venkman in esecuzione. L'interfaccia del debugger può essere configurata in diversi modi, scegliendo gli elementi da aggiungere all'interfaccia in modo da visualizzare gli strumenti più opportuni rispetto al tipo di attività che si sta svolgendo. Nella figura sono state contrassegnate le aree più importanti. Nella sezione A abbiamo la lista di script e finestre in questo momento aperte, che includono anche i file del browser. Nella sezione "b" vengono visualizzati i contenuti di un singolo file. Nella sezione "c" e "d" abbiamo rispettivamente visualizzazione delle

variabili e Call Stack delle funzioni. Infine nella sezione “e” la Interactive Session.

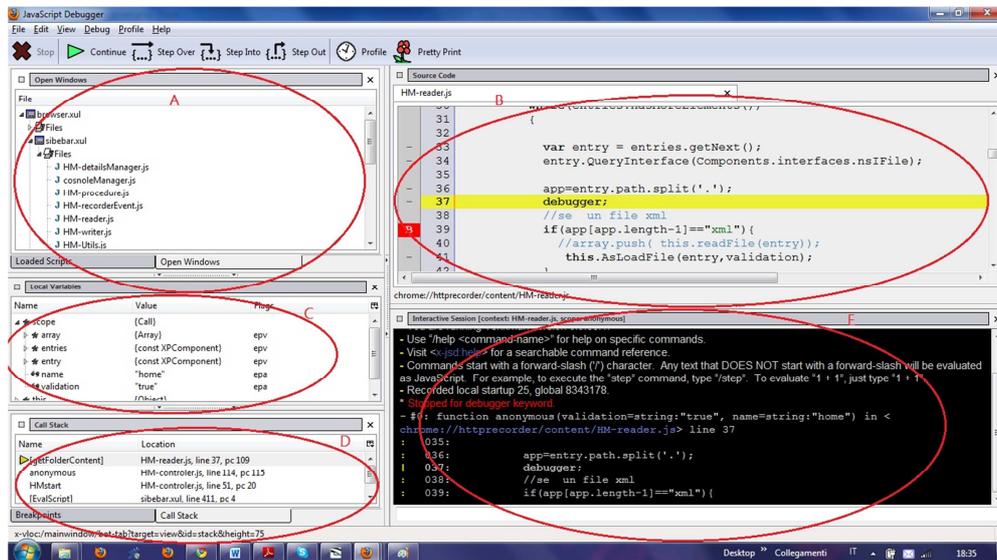


Figura 9: Venkman Debugger

Come mostrato in Figura 10 il debugger visualizza una lista con tutte le finestre aperte in questo momento.

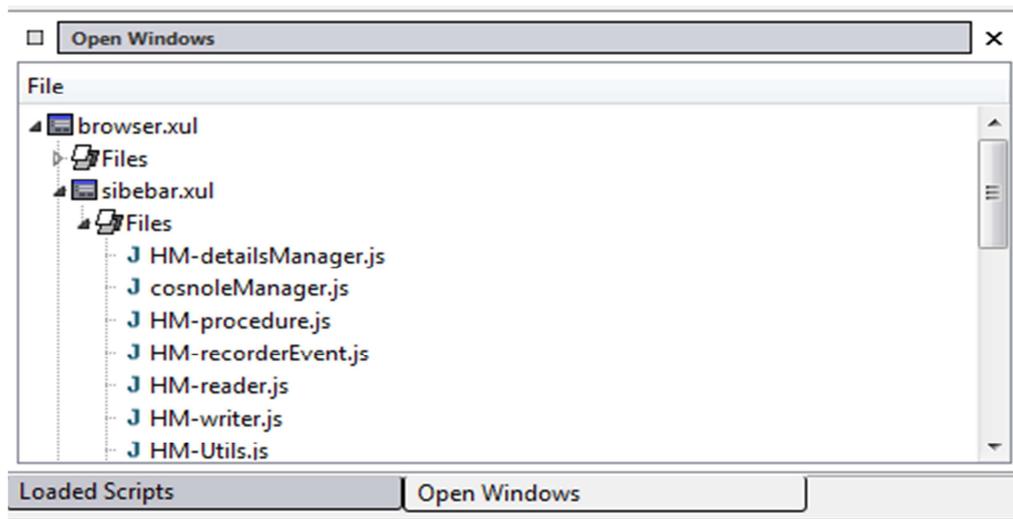


Figura 10: Zoom Loaded Script

Per eseguire il debug è necessario impostare una finestra come “Evaluation Object”. Se l’estensione è aperta, dovremo trovare la finestra che contiene l’estensione e impostarla come Evaluation Object. A questo punto, tutti gli strumenti sono impostati sull’osservazione dell’applicazione desiderata.

Tutto il meccanismo di debug si basa sull'uso dei breakpoint. Permettono di interrompere l'esecuzione del codice in un determinato punto ed osservare lo stato attuale di tutte le variabili e dello stack con le chiamate di funzione. Come mostrato nella Figura 11 i breakpoint possono essere impostati in due modi:

- Inserendo all'interno del codice sorgente la parola chiave `debugger`.
- Etichettando una linea di codice con "B" all'interno dell'area di visualizzazione del codice.

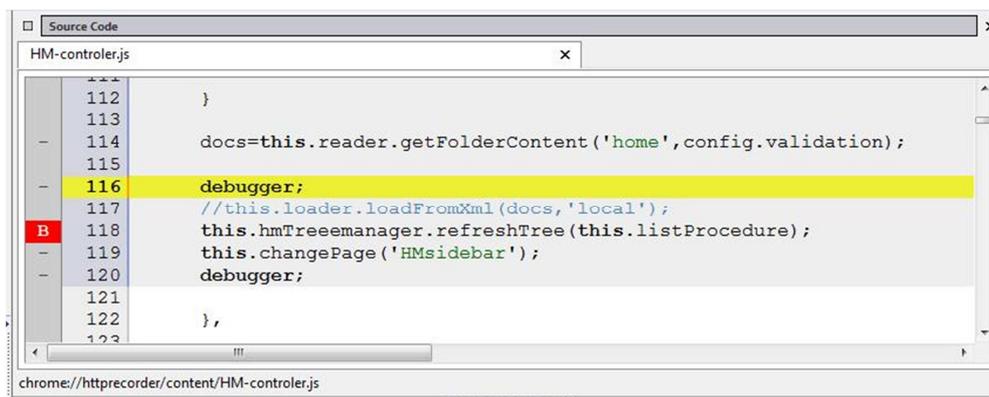


Figura 11: Breakpoint

Quando l'esecuzione del codice arriverà alla linea contrassegnata, il programma verrà interrotto e potremo usare gli strumenti disponibili per analizzare l'attuale situazione del sistema.

Nella Figura 12 è possibile visualizzare tutta la lista delle variabili suddivise in due gruppi:

- Variabili Scope: tutte le variabili che fanno parte dello scope della funzione in cui si è interrotta l'esecuzione del codice.
- Variabili This: tutti le proprietà che appartengono all'oggetto attualmente referenziabile con l'attributo `this`.

Per ogni variabile è possibile osservarne il tipo, il valore e il flag che indica ulteriori informazioni sulla variabile.

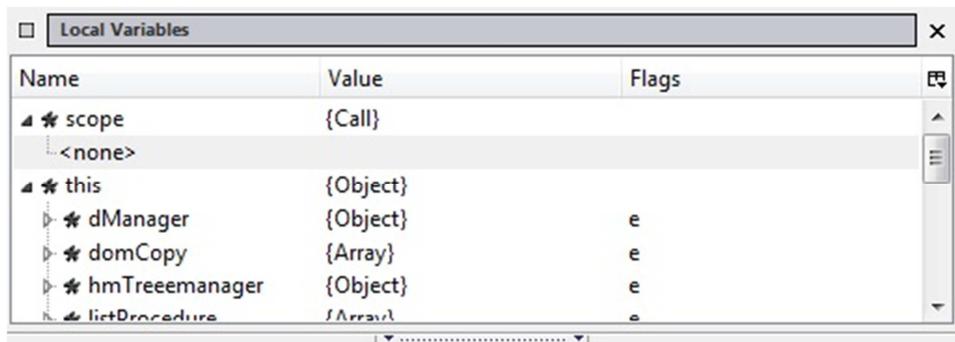


Figura 12: Zoom Lista Variabili

Nella Figura 13 viene mostrato il call Stack, che permette di osservare la situazione attuale dello Stack di chiamate, mostrando la lista di tutte le chiamate di funzione avvenute fino ad adesso.

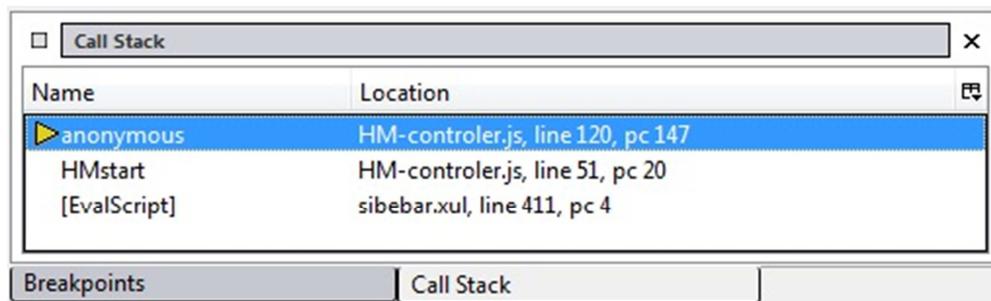


Figura 13 : Zoom Call Stack

Quando lo script è stato messo in pausa per il raggiungimento di un breakpoint, possiamo far ripartire l'esecuzione del programma in differenti modi. Utilizzando step-into nella barra degli strumenti l'esecuzione avanzerà di una linea e si fermerà di nuovo. Con la funzione step-over invece l'esecuzione del codice verrà interrotta alla fine dell'esecuzione della funzione attuale.

Con la Interactive Session mostrata in Figura 14, è possibile interagire con il programma, accedere a tutte le variabili e richiamare funzioni. Ad esempio è possibile inserire istruzioni Javascript e mandarle in esecuzione senza far riprendere l'esecuzione del programma principale.



```
Interactive Session [context: HM-controller.js, scope: [init]]
| 118:   this.hmTreeemanager.refreshTree(this.listProcedure);
: 119:   this.changePage('HMsidebar');
: 120:   debugger;
* Continuing from breakpoint.
* Stopped for debugger keyword.
- #0: function anonymous() in <chrome://htpcreorder/content/HM-controller.js> line 120
: 118:   this.hmTreeemanager.refreshTree(this.listProcedure);
: 119:   this.changePage('HMsidebar');
| 120:   debugger;
: 121:
: 122:   },
< 0001: wiki.log
> ${0} = [boolean] false
```

Figura 14:Interactive session

Quelle precedentemente elencate sono solo le funzioni più utili e più usate dal debugger, che contiene anche delle funzioni più avanzate come ad esempio:

- Funzioni avanzate per il breakpoint. E' possibile creare una funzione Javascript e associarla ad un breakpoint. Ogni volta che l'esecuzione arriverà al breakpoint, il debugger eseguirà anche la nuova funzione. Utile, ad esempio per contare il tipo o il numero di occorrenze e per fare dei controlli automatici.
- Funzioni di profilazione, che servono a misurare i tempi di esecuzione del programma.
- Funzione pretty print, servono alla formattazione del codice, utile ad esempio nel caso in cui una funzione generata tramite la funzione eval() sia di difficile lettura. Con la funzione pretty print possiamo formattare correttamente il codice.
- È possibile escludere il debug di alcune funzioni. Ad esempio in caso dell'utilizzo di un framework Javascript possiamo decidere di ignorare alcuni file
- Watching. Con questa funzione è possibile mettere sotto osservazione una determinata variabile e monitorarne i cambiamenti durante l'esecuzione del codice. Si può usare per monitorare l'andamento di variabili globali e vedere quale funzione ha modificato lo stato della variabile

Il debugger Venkman in definitiva offre strumenti per il debug di un'applicazione Javascript, permettendo di eseguire il codice step-by-step, con la possibilità di osservare lo stato del programma. Permette inoltre di separare le funzioni per il debug da quelle proprie dell'applicazione. Come abbiamo visto con le impostazioni avanzate dei breakpoint, con le funzioni di watching, e visualizzazione delle variabili, è possibile creare delle funzioni e visualizzarne i risultati senza modificare il codice sorgente dell'applicazione.

5 Descrizione del contesto applicativo

L'obiettivo principale di questo lavoro di tesi è sviluppare un'applicazione in grado di monitorare le attività svolte sul Web, registrarle e poterle riprodurre. L'idea è di sviluppare un'applicazione in grado di registrare delle operazioni svolte sul Web, come ad esempio l'acquisto di un biglietto aereo o la creazione di una nuova pagina su un wiki, e di riprodurle in maniera automatica, ripetendo tutti i passi fino al termine dell'operazione.

L'applicazione permetterebbe quindi, attraverso un comando di "registrazione", la memorizzazione di tutte le attività svolte e attraverso un comando di "riproduzione", la replicazione delle stesse.

Le motivazioni che hanno spinto a sviluppare quest'applicazione derivano dall'esigenza di avere un assistente alla navigazione che sia in grado di replicare operazioni di routine sul Web. L'applicazione agevola la navigazione Web, fornendo uno strumento evoluto che si aggiunge a quelli tradizionalmente disponibili.

In aggiunta una caratteristica importante e voluta fin dall'inizio, è quella che permette di identificare gli input delle operazioni registrate. Quindi,

un'applicazione che permettesse, non solo di replicare le operazioni registrate, ma anche di reimpostare tutti i campi di input prima della loro replicazione. Questo permette agli utenti di registrare delle routine, che al momento di essere replicate, possono essere personalizzate e riadattate in base alle esigenze del momento. L'idea è quindi di creare delle routine parametriche che possono essere personalizzate modificandone i parametri.

Il raggiungimento di questo obiettivo, ci ha spinto a creare una piattaforma per la condivisione delle procedure, dove gli utenti possono condividere le proprie routine e importarne altre. La condivisione di routine favorisce il riuso di best-practices. Principio ispirato dall'articolo SAC [5] dove viene proposto un framework per la condivisione e il riuso di best-practices. L'utilizzo di un wiki per la condivisione delle routine è senz'altro il miglior metodo per favorire la partecipazione e la discussione degli utenti.

La nostra idea è simile al concetto di routine e di "macrocomandi", che permettono la replicazione di operazioni abitudinarie. Questo è stato da sempre un problema di difficile soluzione. Uno dei primi approcci fu quello implementato da Microsoft in Windows 3.1 con Macro Recorder. Un sistema che permetteva di registrare una serie di mouse click e di replicarli in automatico. Questo sistema forniva un metodo per replicare semplici task [30]. Ovviamente fu un'applicazione con molti limiti, in quanto non teneva in considerazione il tipo di software che si stesse utilizzando o i tipi di elementi visualizzati nella pagina, ma ripeteva semplicemente degli eventi. I limiti di tale approccio risiedono nella poca flessibilità delle azioni registrate, dove solo cambiando la risoluzione del monitor producevano gravi errori.

Con il passare del tempo il concetto si è evoluto, arrivando allo sviluppo di sistemi per la creazione di macrocomandi specifici per ogni applicazione. Ad esempio le macro per applicativi office, permettono di memorizzare una sequenza di azioni, che devono essere ripetute in più documenti o più volte nello stesso documento. Tra le molte utilità, le macro permettono di riformattare più tabelle in un documento molto lungo o riorganizzare dati in un foglio di lavoro [31]. Oppure le azioni di Photoshop, che permettono di registrare azioni e di

replicarle ogni volta sia necessario. Con questo sistema è possibile avere una propria libreria di azioni da rieseguire su ogni immagine.

5.1 Lavori Correlati

Il concetto di registrazione delle azioni si è sviluppato ed è arrivato a coprire anche il campo della navigazione Web, dove si è vista la nascita di applicativi che permettono la registrazione e la replicazione di azioni. Esistono diverse applicazioni di questo tipo, tra le più famose troviamo iMacros e CoScripter. Entrambe le applicazioni sono simili, permettono di registrare azioni eseguite sul Web, di replicarle e di condividerle con gli altri utenti della piattaforma.

iMacros è una applicazione sviluppata da iOpus, con l'obiettivo di automatizzare azioni eseguite sul Web, testare applicazioni Web ed estrarre dati importanti dalle azioni degli utenti [33]. CoScripter invece è un'applicazione sviluppata dal gruppo di ricerca IBM guidato da Allen Cypher. Anche questa ha l'obiettivo di registrare azioni sul Web, replicarle e condividerle con gli altri utenti.

Uno degli aspetti più innovativi della piattaforma CoScripter, è che le azioni sono rappresentate in un linguaggio di script, leggibile e modificabile dall'utente. Quest'approccio deriva rivisitando le applicazioni Greasemonkey[35] e Chickenfoot [36] che offrono agli utenti la possibilità di modificare script Javascript. L'obiettivo di CoScripter è di creare un linguaggio il più vicino possibile a quello umano, in modo da avere degli script leggibili dall'utente e interpretabili dalla macchina. Il loro obiettivo è di permettere agli utenti di modificare facilmente le azioni contenute nello script con un semplice editor di testo. Ogni azione registrata da CoScripter viene rappresentata con un comando del loro linguaggio di script. Questo, permette di modificare le azioni registrate o addirittura crearne di nuove utilizzando il solo linguaggio di script come unico strumento[6]. Il laboratorio IBM continua a sviluppare il linguaggio, perseguendo degli obiettivi di accessibilità, cercando di creare un linguaggio ad alto livello che semplifichi la navigazione Web[34].

L'applicazione CoScripter ha un obiettivo diverso da quello di questo lavoro di tesi. Infatti, il nostro obiettivo non è quello di sviluppare delle routine che possano essere interamente modificate o create da un particolare linguaggio. Il nostro obiettivo è invece quello di creare delle procedure parametriche, dove si possano modificare in maniera semplice solo alcuni parametri. Non abbiamo lo scopo di permettere la creazione di routine utilizzando un linguaggio di script, ma l'unico modo è di registrarle durante la navigazione Web. Ovviamente, con il nostro approccio, l'utente ha meno possibilità di compiere errori dovuti alla sintassi e alla semantica del linguaggio.

6 WebRecorder

In questo capitolo andremo a descrivere WebRecorder, l'estensione Firefox sviluppata in questo lavoro di tesi, utile per la creazione di procedure Web. Inoltre, mostreremo lo studio che ci ha portato a sviluppare il nostro approccio alla creazione di procedure e che ha guidato tutto il processo di sviluppo dell'applicazione. Concluderemo discutendo l'architettura dell'applicazione e i componenti necessari per implementare tale approccio.

6.1 Introduzione ai concetti di “procedura” e “step”

Un concetto fondamentale all'interno dell'applicazione WebRecorder, è il concetto di “procedura” che ha guidato tutto il processo di sviluppo dell'applicazione. Una procedura Web è quella che precedentemente abbiamo chiamato macro, quindi una sequenza di azioni eseguite sul Web. La procedura è costituita da un insieme di attività svolte sul Web che possono comprendere attività di navigazione, compilazione di form, selezioni di elementi e tutto ciò si

possa fare in una pagina Web. Ovviamente una procedura può comprendere attività eseguite su diverse pagine Web. Il concetto di procedura vuole modellare il processo di navigazione Web, contemplando i concetti di caricamento nuove pagine e azioni eseguite sui singoli documenti.

Una procedura è modellata come un insieme di step. Uno “step” è l’unità atomica che identifica una singola azione compiuta su un documento. Ad esempio, uno step potrebbe essere un click all’interno di un documento, la modifica di un campo di input o la richiesta di una nuova pagina Web.

In definitiva, WebRecorder è un’applicazione in grado di monitorare azioni eseguite su un documento e riprodurre esattamente il processo di navigazione compiuto dall’utente.

6.2 Scelta tecnologica

L’approccio più consono allo sviluppo di un’applicazione di questo tipo consiste sicuramente nell’approccio client-side. L’idea è di monitorare le attività lato client, archiviare dati e in seguito forzare il client a ripetere tali operazioni. All’interno di questo di approccio sono possibili due tipi di soluzioni, una consiste nello sviluppare una browser extension e l’altra implica l’uso dei bookmarklets[5].

I bookmarklets sono dei programmi Javascript che possono essere inseriti in un URL. Utilizzando lo schema Javascript, si possono inserire delle istruzioni in un URL che saranno eseguite dal parser. Il browser identifica il protocollo Javascript, esegue il programma e ne visualizza i risultati nella pagina. E’ possibile utilizzarli anche senza valore di ritorno, eseguendo delle operazioni sul documento corrente e infine aprendo una nuova pagina. Possono essere utilizzati in qualunque posto in cui si userebbe un normale URL, all’interno di un documento o salvarli tra i preferiti[4].

Sarebbe possibile adottare i bookmarklets come soluzione implementativa della suddetta applicazione. Di fatto con questa tecnologia è possibile eseguire le operazioni fondamentali per la nostra applicazione, ma sarebbero comunque limitativi rispetto all’uso di una Firefox Extension.

In primo luogo non hanno la possibilità di essere installati come applicazione Chrome. Conseguentemente, si perderebbero tutti i vantaggi derivanti dal fatto di poter utilizzare i componenti Xpcom ed eseguire operazioni con i privilegi Chrome. In aggiunta, sarebbe di difficile gestione il sistema di archiviazione dei bookmarklets, i quali vengono gestiti come un URL. Questo comprometterebbe sicuramente l'usabilità del sistema. Infine, i bookmarklets ci limiterebbero anche dal punto di vista della modellazione del software, in quanto prevedono che tutto il programma sia contenuto in una sola linea di codice.

In definitiva, i bookmarklets sono una soluzione per lo sviluppo di piccoli programmi Javascript che non si adattano alla modellazione di un sistema complesso come WebRecorder. Per questo si è deciso di sviluppare una browser extension.

6.3 Descrizione dell'applicazione

L'applicazione WebRecorder è una Firefox Extension contenuta all'interno di una sidebar. L'applicazione può essere avviata dal menu "Strumenti" di Firefox, dove sono presenti tutte le estensioni installate nel sistema, oppure digitando la combinazione di tasti per la scelta rapida "CTRL+MAIUSC+A".

Come mostrato nella Figura 15, una volta avviata l'applicazione viene aperta la sidebar, all'interno della quale è possibile navigare tra le pagine dell'applicazione. WebRecorder è completamente contenuta in locale, quindi l'applicazione potrebbe anche essere eseguita anche off-line.

Come detto in precedenza, è possibile condividere con altri utenti le proprie procedure. Questo avviene tramite la pubblicazione delle procedure su un wiki, che ovviamente non risiede in locale ma viene utilizzato come servizio esterno. Il wiki viene utilizzato per l'archiviazione e la condivisione delle procedure pubbliche e per offrire agli utenti della piattaforma la possibilità di discutere e scambiare opinioni. All'interno dell'applicazione è stata predisposta una sezione che permettere di effettuare il login su un wiki esterno. Questo serve semplicemente a recuperare le informazioni e le procedure preferite disponibili

su tale wiki o a pubblicare le procedure in maniera non anonima. Il mancato login non compromette in nessun modo l'utilizzo dell'applicazione o la possibilità di pubblicare procedure.

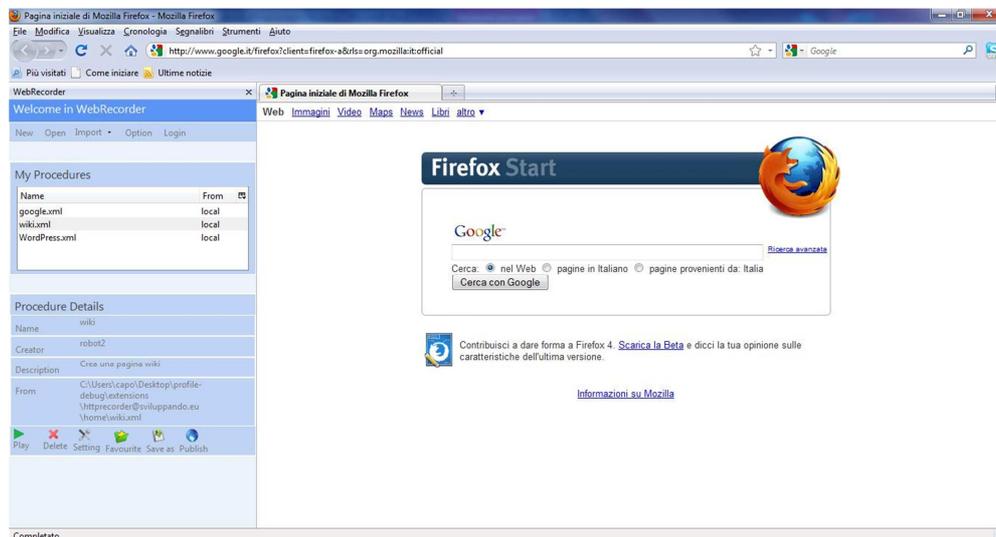


Figura 15: WebRecorder

WebRecorder permette di collegarsi ad un qualsiasi wiki correttamente predisposto quindi, come vedremo meglio in seguito, è possibile cambiare facilmente il wiki di riferimento. Per questo la registrazione deve essere eseguita sul wiki stesso e non viene gestita da WebRecorder, dove è possibile solo effettuare il login.

Le procedure vengono salvate in formato XML. Il sistema quindi prevede che tutte le procedure siano in questo formato, sia quelle salvate su file che quelle contenute in una pagina wiki. Per garantire la validità delle procedure importate è stato predisposto un modulo per la loro validazione che può essere attivato opzionalmente.

6.3.1 Home page

La home page, mostrata in Figura 16, è la pagina principale dell'applicazione. In questa pagina troviamo tutte le procedure caricate nel sistema e tutti i comandi principali.

Nella toolbar, situata nella parte superiore della pagina, sono presenti una serie di pulsanti per l'attivazione delle principali funzioni del programma. Con il pulsante "New" si accede alla pagina che permette di registrare una nuova procedura. I pulsanti "Open" e "Import" permettono di caricare nuove procedure all'interno del programma. Open, apre una dialog box per la selezione del file sul disco, mentre Import importa una procedura contenuta all'interno di una pagina Web permettendoci di sceglierne l'URL. Infine, Option apre un dialog box per la modifica delle impostazioni generali.

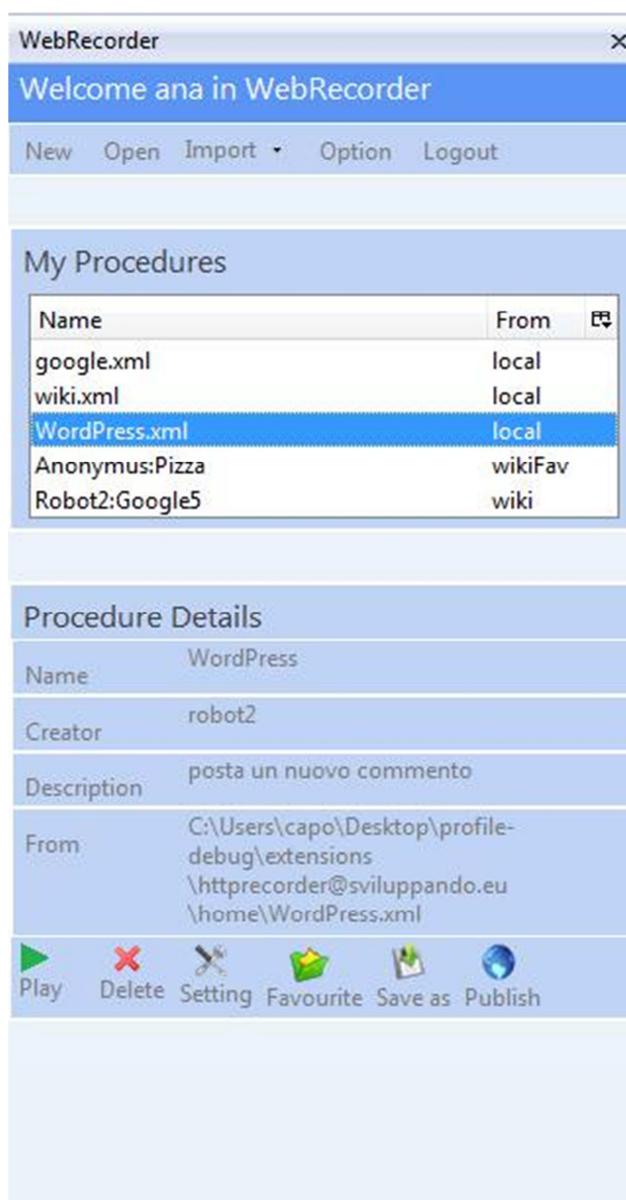


Figura 16: Home Page

Nella sezione “My Procedure”, nella parte centrale della pagina, troviamo l’elenco delle procedure attualmente caricate nell’applicazione. Ogni linea dell’elenco contiene informazioni su una singola procedura. Per ogni procedura sono presenti due campi: “Name” e “From”. Nel campo Name si trova il nome del file contenente la procedura. Nel caso la procedura fosse importata dal wiki, il nome corrisponderebbe a quello della pagina in cui è contenuta.

Il campo From contiene un “flag” che indica la provenienza della procedura. I possibili valori del campo From sono i seguenti:

- local: la procedura è stata caricata dal file system locale.
- wiki: la procedura è stata importata da una pagina wiki.
- wikiFav: la procedura è stata importata dal wiki perché presente tra i propri preferiti del wiki.
- new: la procedura è stata registrata in questa sessione ma non è ancora stata salvata.

Selezionando una procedura dell’elenco si aggiornano automaticamente i campi della sezione “Procedure Details”, dove sono forniti degli ulteriori dettagli sulla procedura selezionata:

- Name: contiene il nome.
- Creator: contiene il nome dell’autore. Potrebbe essere un qualunque utente, dato che è possibile importare procedure dal wiki.
- Description: una breve descrizione.
- From: indica la locazione. Contiene il path o l’URL a seconda che la procedura venga caricata dal file system locale o dal wiki.

Nella parte inferiore della sezione Procedure Details, è presente una toolbar contenente i pulsanti per attivare funzioni sulla procedura selezionata. Da qui è possibile attivare il comando Play, necessario a mandare in esecuzione una procedura. Attivando questa funzione saremo reindirizzati alla pagina di Play. Il comando Delete, permette di eliminare una procedura dalla lista di procedure attualmente caricate nel sistema e opzionalmente di rimuoverla dal disco locale o dai preferiti del wiki. E’ importante notare che non è possibile usare questa

funzione per eliminare una pagina wiki, per questa operazione è necessario aprire la pagina, effettuare il login e cancellarla direttamente dal wiki.

Una delle funzioni più importanti del sistema, è quella che ci permette di modificare le impostazioni della procedura selezionata, attivabile dal pulsante “Setting”.

Infine abbiamo “Save as” e “Publish” che permettono di esportare la procedura attualmente selezionata. Con Save as possiamo salvarla in locale, mentre Publish crea una nuova pagina contenente la procedura sul wiki attualmente selezionato. Per la sua pubblicazione saremo reindirizzati in una dialog box per la verifica dell’esistenza e selezione del nome della pagina wiki.

6.3.2 Rec page

The screenshot shows the WebRecorder application interface within a Mozilla Firefox browser window. The browser's address bar displays the URL: `http://www.google.it/#hl=it&source=hp&biw=1043&bih=585&q=WebRec`. The page title is "WebRecorder - Cerca con Google".

The interface is divided into several sections:

- Navigation Bar:** Includes "Record Your Procedure", "Logout" (with a red stop button), "Stop", and "Box step".
- Box step:** A list of events: "load", "mousedown", "change", "mouseup", "click".
- Procedure Details:** A table with the following data:

| Procedure Details | |
|-------------------|--------------|
| Name | NewProcedura |
| Creator | ana |
| Description | |
| From | new |
- Search Results:** A Google search for "WebRecorder" showing approximately 82,000 results. Suggested search terms include "WebRecorder A", "Internet Macros V", "Web recorder lx", and "WebRecorder C".
- Sidebar:** Contains navigation links for "Tutto", "Immagini", "Video", "Notizie", "Bologna", and "Nel Web".

Figura 17: Rec page

La pagina di Rec è la pagina predisposta per la registrazione di una procedura.

Una volta entrati nella pagina è possibile iniziare la registrazione della procedura. La registrazione si avvia premendo il tasto Rec nella Barra degli strumenti in alto nella pagina. Durante questa fase, come mostrato in Figura 17, si attiva un segnale rosso lampeggiante, che contraddistingue la fase di registrazione di una procedura.

Nella parte centrale della pagina è presente un elenco che tiene traccia di tutti gli step registrati durante la navigazione. Vengono registrati step relativi a tutti i documenti aperti, questo significa che è in grado di registrare attività in uno qualunque dei Tab. Quindi, WebRecorder registra anche attività del browser come apertura o chiusura di un Tab.

Nella parte inferiore della pagina troviamo la maschera con i dettagli della procedura e la toolbar per l'attivazione delle funzioni su una procedura. Nella Figura 17 la toolbar è assente, in quanto durante la fase di registrazione non è possibile attivare funzioni che modifichino una procedura, quindi la toolbar in questo momento non è visibile.

Quando terminiamo di registrare una procedura, verrà inserita automaticamente tra le procedure caricate nel sistema con lo stato di "New" e con un nome di default. Per cambiare queste impostazioni è necessario entrare nella sezione di edit, dove si possono modificare le caratteristiche di una procedura e dei singoli step. Per questo nella maschera di riepilogo dati della Figura 17, troviamo la procedura con il nome di "New Procedure". Il nome del Creator in questo momento è impostato su "Ana", dato che al momento della registrazione era stato effettuato il login dall'utente Ana.

6.3.3 Edit page

La pagina di edit è una delle sezioni più importanti dell'applicazione, qui è possibile modificare una procedura e tutti i suoi step. Si può accedere alla pagina di edit da qualunque sezione dell'applicazione. Significa che possiamo modificare una procedura dopo averla creata, importata da un wiki o da una

pagina Web. Dopo averla modificata, è possibile salvarla in locale o pubblicarla di nuovo .

All'interno di questa pagina sono riportati tutti i dati di una procedura e dei suoi step. Come vediamo in Figura 18, dove viene riportata una schermata della finestra di edit, le informazioni sono suddivise in quattro sezioni.

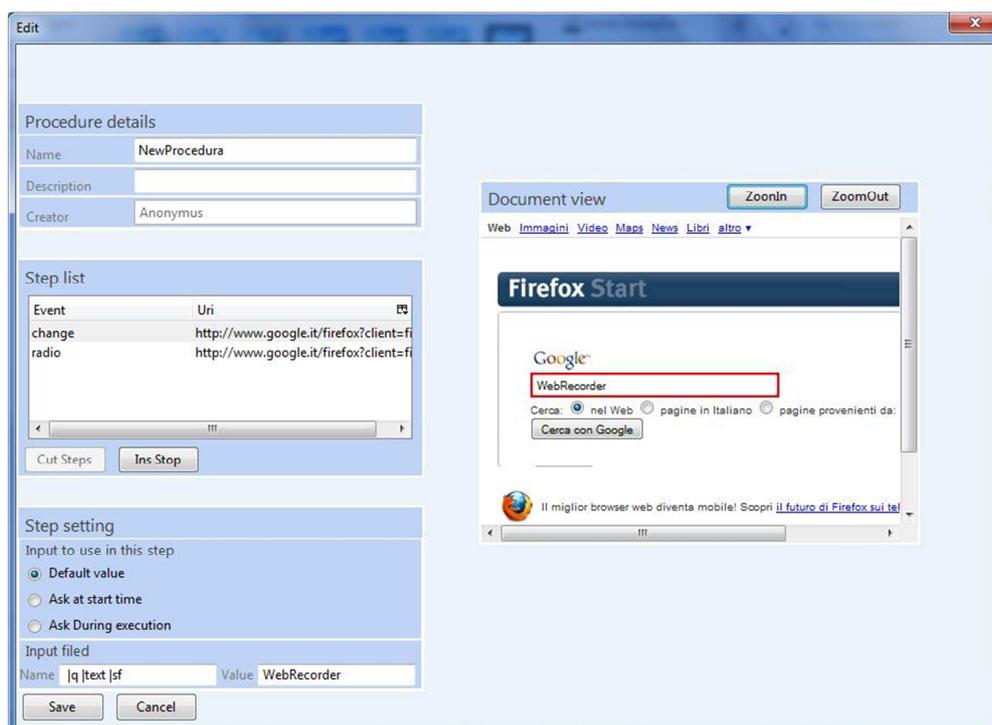


Figura 18: Edit Page

Nella sezione di “Procedure Details” sono presenti i dati relativi alla procedura, ovvero:

- Name: nome.
- Description : una breve descrizione.
- Creator: nome del creatore. Questo parametro non può essere modificato, riporta il nome dell’utente identificato al momento della registrazione.

Nella sezione “Step list” viene riportato l’elenco di tutti gli step che compongono la procedura. Per ogni step è presente il tipo di azione eseguita e l’URL del documento su cui si è verificata. Nell’esempio della Figura 18 gli step sono due, un change che determina il cambio di valore di una text box, ed uno radio che rappresenta il cambio di valore di un radio button. E’ possibile

modificare la lista degli step utilizzando i pulsanti di “Cut Step” e “Ins Stop”. Il primo, Cut Step, permette di cancellare parte della procedura fino allo step selezionato, che diventerà il primo step della procedura. E’ possibile effettuare questa operazione solo su eventi di tipo load. Ins Stop permette di inserire uno step di pausa. Quando l’esecuzione della procedura arriverà a questo step, si fermerà e attenderà un comando dell’utente per riprendere l’esecuzione.

Selezionando uno step vengono aggiornate automaticamente le sezioni di “Step setting” di “Document view”.

Nella sezione di Document view è presente la riproduzione del documento su cui si è stato prodotto lo step. Il documento viene riprodotto esattamente nello stato in cui si trovava al momento del verificarsi dell’evento. Come possiamo vedere nella Figura 18, selezionando lo step change, viene riprodotto il documento su cui si è verificato l’evento e viene evidenziata la zona su cui si è verificato colorandola di rosso. Come mostrato nell’esempio, è evidenziata la input box dell’home page di Google. In aggiunta nella parte superiore ci sono due pulsanti che permettono lo zoom sul documento, necessari nel caso in cui non sia possibile leggere i testi. Questa sezione serve ad aiutare gli utenti durante la fase di configurazione di una procedura, avendo sempre la corrispondenza tra step e zona dal documento su cui si è verificato.

Nella sezione di Step setting è possibile modificare le impostazioni di un singolo step. Non tutti gli step possono essere personalizzati ma solo quelli riguardanti la modifica di un campo di input. Ad esempio, selezionando uno step di tipo “Mouse Down”, la sezione Step setting non è attivata e non possono essere apportate modifiche allo step in questione. Invece, selezionando uno step di modifica di un campo di input, automaticamente la sezione di Step setting si aggiorna con le impostazioni relative a quello selezionato.

In Figura 19 è riportata la pagina maschera di Step setting aggiornata con i valori di un evento change. Nella prima parte sotto la voce “Input to Use in this Step”, possiamo selezionare la modalità di utilizzo dell’input durante la fase di replicazione. E’ possibile scegliere una delle seguenti tre opzioni:

- Default value: in fase di riproduzione non viene chiesto all’utente di impostare il valore di questo campo di input.

- Ask at Start time: in fase di riproduzione verrà chiesto all'utente di impostare il valore prima dell'esecuzione.
- Ask during execution: in fase di riproduzione l'esecuzione verrà interrotta per chiedere di impostare il campo di input.

Figura 19: Step setting

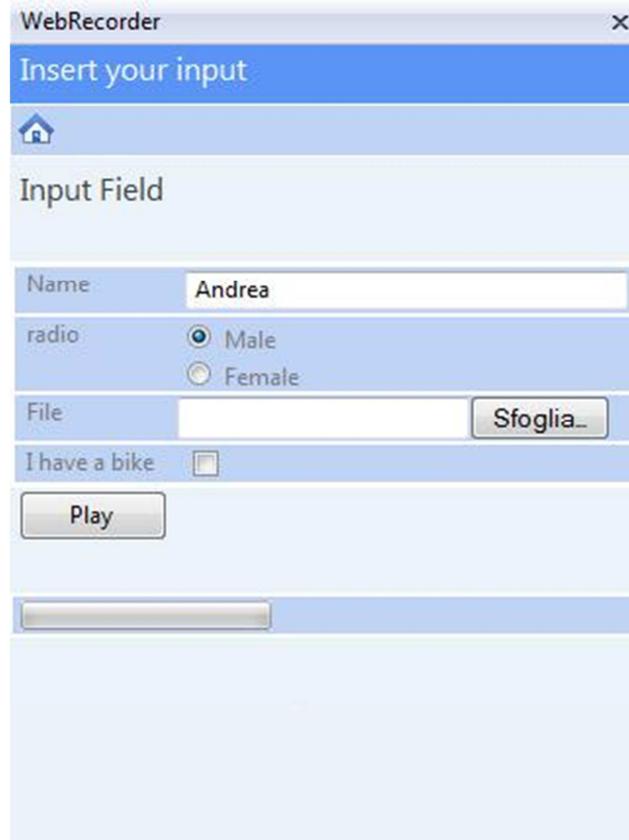
Nella parte inferiore sotto la voce “Input field” è possibile impostare un'etichetta ed un valore. Quindi, nel campo Name possiamo impostare il nome del campo di input, che sarà lo stesso ad essere presentato all'utente in fase di replicazione. Inoltre, è possibile definire un valore di default inserendolo nel campo “Value”. Nell'esempio in Figura 19, vediamo che i due campi Name e Value hanno già un valore. Sono i valori di default selezionati dall'applicazione in fase di registrazione.

Nella sezione Step setting possono essere riprodotti tutti i tipi di input. Quindi, nel caso in cui avessimo un input di tipo radio dovuto alla modifica di un radio button, verrebbe riprodotta tutta la sequenza di radio button presenti nel documento come mostrato in figura Figura 20.

Figura 20: Setting radio step

6.3.4 Play page

La pagina di Play è la sezione utilizzata per controllare l'esecuzione delle procedure. All'interno di questa pagina potrebbe essere chiesto di reimpostare gli input, come vediamo nella Figura 21.



The screenshot shows a web browser window titled "WebRecorder" with a close button (X) in the top right corner. The page content is as follows:

- A blue header bar with the text "Insert your input" and a home icon on the left.
- A section titled "Input Field" with a light blue background.
- A "Name" input field containing the text "Andrea".
- A "radio" input field with two options: "Male" (selected with a blue dot) and "Female" (unselected with a grey dot).
- A "File" input field with an empty text box and a "Sfoggia..." button to its right.
- An "I have a bike" input field with an unchecked checkbox.
- A "Play" button.
- A horizontal progress bar below the "Play" button.

Figura 21: Play page

In questa pagina viene proposto un form, dove possono essere reimpostati tutti i campi di input modificabili per la procedura. Sono visualizzati il nome del campo di input e il suo valore di default. Nell'esempio contenuto nella Figura 21 abbiamo 4 input di tipo differente:

- Input type "text": il cui nome del campo di input è "Name" e il valore di default è "Andrea".
- Input type "radio": il cui nome del campo è "Select" e vengono proposti due possibili radio button, "Male" e "Female" il valore di default è Male.

- Input type “checkbox”: il cui nome è “I have a bike” e il valore di default del campo è “Selected”.
- Input type “file”: il cui nome è “File” e non ha nessun valore di default. Con il pulsante select viene aperta una dialog box che ci permette di selezionare dal disco locale.

Una volta impostati i campi di input è possibile mandare in esecuzione la procedura e i nuovi valori verranno utilizzati durante la sua esecuzione. E' possibile che alcuni campi di input vengano richiesti durante l'esecuzione della procedura. In questo caso l'esecuzione della procedura sarebbe interrotta e saranno proposti nuovi campi di input da compilare. Il pulsante “Play”, in questo caso, sarebbe sostituito dal pulsante “Continue” per far ripartire l'esecuzione della procedura. In fondo alla pagina troviamo una progress bar che indica lo stato di avanzamento di una procedura durante la sua esecuzione.

In definitiva, questa è la pagina che ci permette di impostare gli input di una procedura e di controllarla durante la sua esecuzione.

6.3.5 Option page

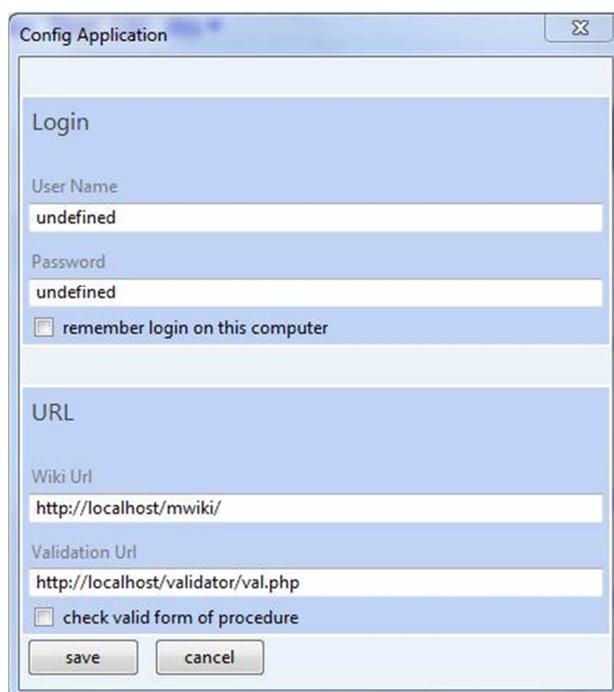


Figura 22: Option page

La pagina di option, mostrata in Figura 22, permette di modificare le impostazioni generali dell'applicazione. Possono essere modificate impostazioni relative al login e URL di riferimento.

Si possono modificare gli URL del wiki di riferimento e l'URL del server di validazione. In aggiunta per attivare la validazione è necessario spuntare la checkbox con "check valid form", così ogni procedura sarà importata solo se avrà superato la validazione, altrimenti sarà automaticamente ignorata.

6.4 Case test

Prima di iniziare con lo sviluppo dell'applicazione sono stati selezionati dei case test. In questo specifico progetto i case test sono siti Web su cui testare l'applicazione. Sono stati scelti deliberatamente eterogenei, cercando di fornire un ampio ventaglio che andasse a coprire tipi differenti di documenti Web, garantendo maggiore compatibilità all'applicazione.

L'applicazione è stata testata più volte durante tutto il processo di sviluppo, influenzando profondamente le scelte progettuali. Si è arrivati alla versione attuale in seguito a molteplici prove e test che hanno causato la modifica degli algoritmi e delle tecniche utilizzate. I siti Web selezionati per il case test sono i seguenti:

- Google Documents.
- Mediawiki.
- WordPress.
- PizzaBo.
- JQueryui.

Ognuno dei siti Web elencati ha delle caratteristiche che ne hanno determinato l'inclusione all'interno del test case. Google Documents è stato scelto per la complessità della sua interfaccia e le note caratteristiche Ajax. Il wiki Mediawiki e il blog WordPress sono stati selezionati come applicazioni Web di base, senza funzionalità Ajax particolarmente evolute e considerate tra le applicazioni Web più diffuse. Infine PizzaBo e JQueryui sono stati selezionati

durante il processo di sviluppo, per l'inclusione nel case test di funzionalità particolari. Nella sezione UI di JQuery è possibile utilizzare widget drag-and-drop, mentre PizzaBo è un'applicazione Web 2.0 che ha la caratteristica di aggiornare dinamicamente le pagine aggiungendo di volta in volta nuovi elementi grafici, comportamento particolarmente interessante da studiare per la nostra applicazione.

6.5 DOM event vs HTTP activity

Il primo punto critico dell'applicazione sta sicuramente nel metodo scelto per implementare i concetti di procedure e step. Ci riferiamo al metodo utilizzato per monitorare le attività utente e per replicarle. Questo punto influenza tutta l'applicazione che sarà basata su questo concetto. Infatti, decidere cosa monitorare implica conoscere a priori i dati di cui abbiamo bisogno per replicare una procedura. Quindi, il problema da risolvere è capire quali sono le informazioni sufficienti a replicare una procedura, e in seguito sarà possibile progettare un sistema per monitorarle e archiviarle.

Un primo approccio è basato sulla replicazione del traffico HTTP. L'idea è di monitorare le attività HTTP eseguite durante una sessione e archiviare i dati relativi a tutto il traffico. Per fase di riproduzione della procedura quindi, sarebbe necessario sviluppare un algoritmo che analizzando i dati, sia in grado di replicare la procedura riproducendo il traffico HTTP.

Un approccio alternativo a quello appena illustrato, consiste nel monitorare gli eventi DOM. Qui l'idea è di monitorare tutti gli eventi DOM generati su un determinato documento e archiviare dati su di essi. Durante la fase di replicazione di una procedura è necessario riprodurre artificialmente gli eventi DOM riproducendo la navigazione dell'utente.

Ambedue gli approcci hanno vantaggi e svantaggi. Ad esempio l'approccio basato su eventi DOM, ha lo svantaggio di generare degli step dipendenti dalla struttura del documento. Il problema sta nel fatto che con il passare del tempo, la struttura del documento potrebbe cambiare, rendendo difficile individuare di nuovo l'elemento su cui riprodurre l'evento.

D'altro canto, anche l'approccio basato su attività HTTP ha degli svantaggi, situati nella difficoltà d'interpretazione del traffico. Riuscire a interpretare in maniera sistematica il traffico HTTP, potrebbe avere delle difficoltà, poiché le varie applicazioni Web non rispettano sempre i comuni pattern di comunicazione. Lo sviluppo di applicazioni di Ajax e Web 2.0 ha portato ad un allontanamento dai pattern di comunicazione standard. Ad esempio analizzando il traffico HTTP potrebbe essere difficile identificare i valori di input inseriti dall'utente. Di seguito si riporta un classico esempio di comunicazione HTTP:

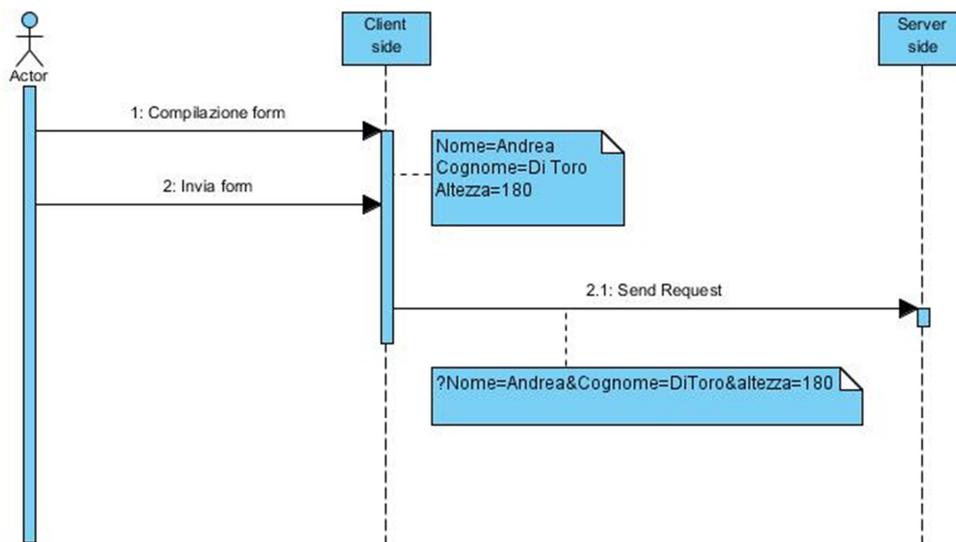


Figura 23: Diagramma Interazione http

La Figura 23 mostra un'interazione classica, l'utente compila i dati e invia il form. Il client codifica i dati in una query string che viene inviata al server con una richiesta GET o POST. Nell'esempio appena riportato, intercettando la comunicazione, sarebbe semplice capire quali sono i dati del form e quindi capire quali sono le variabili contenenti i dati di input. Non è detto che tutte le applicazioni Web rispettino questo pattern di comunicazione. I dati potrebbero essere inviati con una richiesta di tipo POST, in formati non sempre facili da interpretare come XML, JSON o altri. I nomi delle variabili non sempre sono quelli attesi. In oltre i contenuti stessi delle variabili, spesso sono dei codici

identificativi di elementi selezionati nella pagina, difficilmente interpretabili modificabili dall'utente.

Un altro problema di più difficile soluzione, è emerso durante l'implementazione di un prototipo basato su questo approccio. In alcuni casi gli URL vengono generati dinamicamente e contengono codici di sessione non replicabili. Di seguito riportiamo un esempio di comunicazione di Google Documents.

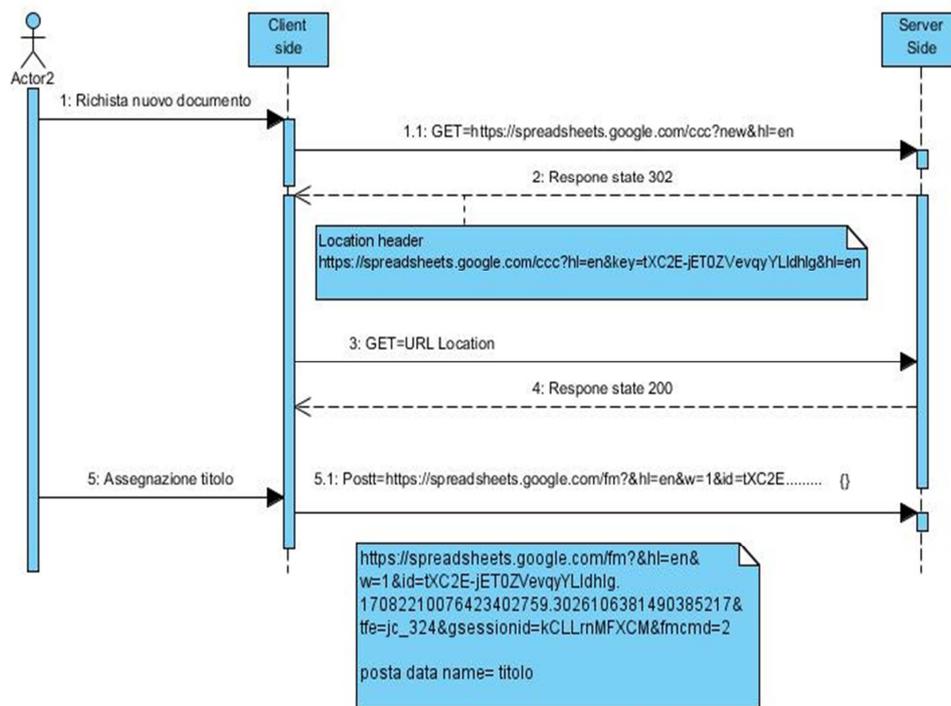


Figura 24: Diagramma interazione HTTP Google

Nel diagramma d'interazione della Figura 24, vediamo come a ogni azione dell'utente corrisponde una richiesta. In seguito alla richiesta di creazione di un nuovo documento, il client invia una richiesta GET a un URL. Il server risponde con un response avente status-code 302 e con un header location contenente l'URL del nuovo documento creato. Questo pattern di comunicazione è comune ed è spesso utilizzato anche negli altri case test. Il problema arriva dalla richiesta effettuata in seguito all'assegnazione di un titolo al documento. Come vediamo nella nota sottostante la richiesta 5.1, l'URL utilizzata include diverse variabili. Le variabili contengono dati di sessione relativi al login dell'utente e alla sessione instaurata da questa comunicazione.

Quindi, quest'URL cambia in base alla sessione, all'id del documento e al login dell'utente.

Le difficoltà di replicazione di questa richiesta sono molto alte, dovremo disporre dei dati di sessione presenti nel cookie di sessione, del codice identificativo del documento presente nell'URL della precedente richiesta e dovremo capire come ricostruire l'URL ad ogni richiesta.

Per questi motivi abbiamo deciso di scegliere l'approccio basato su DOM event, anche se non privo di problemi sicuramente ci permette di generare un'applicazione più standard e che funzioni sulla maggior parte di applicazioni Web. Nei paragrafi successivi si riporta una trattazione approfondita dell'implementazione del sistema basata sull'approccio DOM event.

6.6 Procedure basate su eventi DOM

Una volta scelto l'approccio basato su DOM event, bisogna stabilire quali sono i dati importanti da monitorare per generare step e procedure. Per poterlo capire, è necessario individuare i dati di cui abbiamo bisogno per replicare una procedura. Il recorder e il player sono stati sviluppati in parallelo e sono frutto di molteplici test, così siamo arrivati ad avere un set di dati sufficiente a riprodurre una procedura. La difficoltà sta appunto nell'individuare i dati necessari a modellare il processo di navigazione dell'utente e per cercare di replicare tutte le sue azioni.

Il player di DOM event è il componente incaricato di riprodurre una procedura analizzando i dati prodotti dal recorder. La riproduzione avviene generando artificialmente gli eventi DOM all'interno dei documenti e riproducendo la sequenza di azioni eseguite dall'utente durante la navigazione. Nei seguenti paragrafi, andremo a descrivere quali sono i dati necessari al player per riprodurre una procedura e di conseguenza quelli che il recorder deve analizzare e archiviare durante la fase di registrazione.

Per arrivare ad avere questo set di dati, è necessario analizzare una serie di problemi che si presentano al momento di modellare il processo di navigazione dell'utente. Tutti i dati elencati in seguito sono quelli sufficienti a

descrivere una procedura e sono ricavati dal documento e dagli eventi DOM generati sul documento. Queste rappresentano le uniche fonti da cui attingeremo i dati per una creare una procedura.

Gli oggetti event contengono una grande quantità di dati riguardanti l'evento generato e possono essere di diverso tipo, a seconda dell'evento che rappresentano. La categoria di appartenenza di un evento definisce le proprietà dell'oggetto e quindi il tipo d'informazioni contenute nell'evento stesso. Alcune proprietà invece sono in comune tra tutti gli oggetti, come ad esempio la proprietà target che permette di accedere all'elemento DOM su cui si è generato l'evento.

L'altra fonte importante di dati per le procedure è l'oggetto document. Da esso possiamo accedere all'URL, al referer e ci permette di recuperare altri dati importanti per la modellazione della procedura.

6.6.1 Identificazione del documento

Il primo problema sta sicuramente nell'individuazione del documento su cui si genereranno gli eventi. Il primo elemento necessario è l'URL del documento che lo identifica in maniera univoca.

In aggiunta c'è un altro problema da risolvere relativo all'individuazione del documento. E' l'identificazione del Tab su cui si è verificato l'evento. Questo è necessario dato che vogliamo gestire procedure, che coinvolgono più Tab. Alcuni di questi potrebbero essere aperti fin dall'inizio, altri verranno aperti o chiusi durante l'avanzamento della procedura. Ad esempio, una procedura potrebbe coinvolgere due volte lo stesso documento aperto in Tab differenti o un documento potrebbe essere chiuso e riaperto più volte.

Durante la fase di sviluppo dell'applicazione si è arrivati alla decisione di inserire un ID per identificare i Tab. Il problema nasce nell'identificare il Tab in cui è necessario riprodurre un evento "load", il quale è generato dal caricamento di un nuovo documento. Inizialmente si pensava di risolvere il problema analizzando l'attributo referer dei nuovi documenti caricati, tenendo traccia così del'URL del precedente documento.

L'idea era di registrare l'attributo referer del nuovo documento caricato, e nella fase di replicazione della procedura, recuperare il documento con l'URL referer e spingerlo al load del nuovo documento. Il problema si presenta quando, durante la navigazione, il load del documento è causato dalla digitazione dell'URL nella barra degli indirizzi. In questo caso il documento ha referer nullo e nella fase di replicazione della procedura non sapremo in che Tab dobbiamo forzare il load del nuovo documento.

Per risolvere i problemi elencati è stato necessario aggiungere un ID al Tab, in modo da identificare sempre quello su cui sta lavorando l'utente. Elimineremo così ambiguità dovute a più documenti aperti con lo stesso URL e da eventi load generati digitando l'URL nella barra degli indirizzi. Così ogni step conterrà l'URL del documento su cui deve essere riprodotto e l'id del Tab contenente il documento.

6.6.2 Identificazione dell'elemento DOM

Un altro dato importante per una procedura DOM è quello che permette di individuare l'elemento su cui si è verificato l'evento. Per individuarlo in maniera univoca all'interno di un documento, è necessario identificare la sua posizione all'interno dell'albero DOM. Il metodo utilizzato per individuare l'elemento è fondamentale per questo tipo di approccio. Come detto in precedenza uno degli svantaggi di lavorare con l'approccio DOM, sta proprio nel rischio di generare procedure dipendenti dalla struttura del documento.

Ad esempio, se il documento sul quale abbiamo registrato una procedura fosse modificato, durante la replicazione potrebbero verificarsi degli errori dovuti all'errata identificazione dell'elemento. E' necessario identificare l'elemento con un metodo che conferisca robustezza all'applicazione.

Il modo più semplice per identificare un elemento DOM è di utilizzare gli attributi ID. Questo renderebbe la nostra applicazione resistente alle modifiche del documento in quanto aggiungendo o eliminando elementi riusciremmo comunque a recuperare l'elemento esatto. Come noto, non tutti gli elementi hanno un ID. Anzi, gli elementi con ID all'interno di un documento sono sicuramente la minor parte.

Un'altra soluzione a questo problema è di navigare l'albero DOM a ritroso, dall'elemento su cui si è verificato l'evento fino all'elemento radice. Archiviando dati su tutti i passi necessari per raggiungere il nodo radice è possibile costruire un "path", che ci permetterà di recuperare di nuovo l'elemento. Così facendo, in fase di replicazione della procedura, potremmo navigare l'albero dall'alto verso il basso, fino a raggiungere l'elemento interessato. Ad esempio nel grafico in Figura 25, dove è riportato un esempio di albero DOM, dovendo identificare la posizione dell'elemento su cui si è verificato l'evento click, dovremmo navigare l'albero fino all'elemento radice. Annotando la posizione di ogni elemento rispetto al padre, produrremo un path che, ci permetterà di localizzare l'elemento.

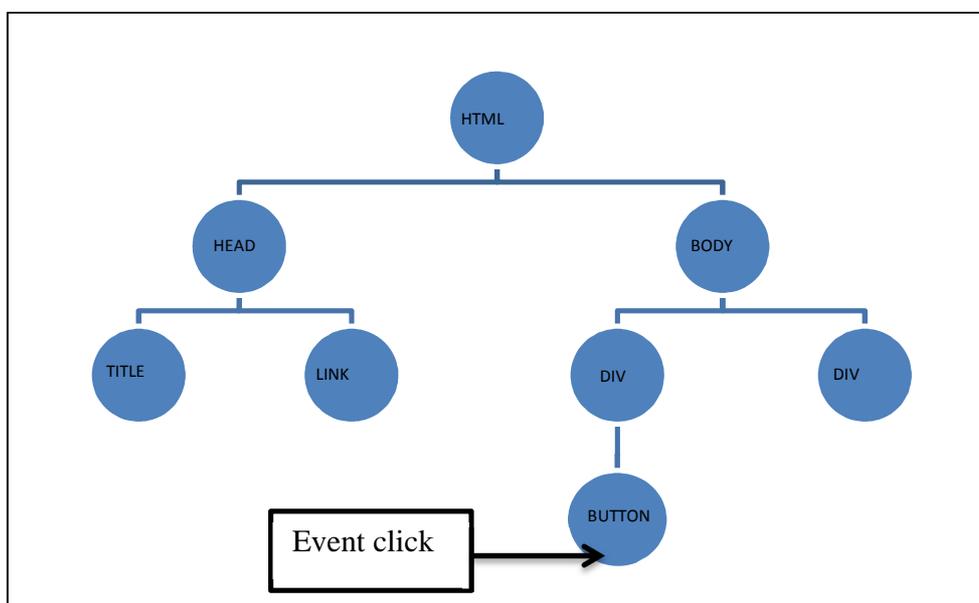


Figura 25: Albero DOM

Nell'esempio riportato in Figura 25 il path sarebbe HTML/1/0/0. Questo perché l'elemento "button" è il figlio numero 0 di "div", che a sua volta è il figlio numero 0 di "body" ed è il figlio numero 1 di "HTML".

Con quest'approccio riusciremmo sicuramente a identificare in maniera univoca gli elementi. Resta però il problema della dipendenza dalla struttura del documento. Ad esempio, nel grafo precedente, basterebbe aggiungere un elemento div come primo figlio di "body" per modificare il path, e rendere inutilizzabile la procedura registrata sul vecchio documento.

Il metodo che minimizza i problemi deriva da una fusione tra il metodo dei path e quello degli ID. La soluzione prevede il calcolo del path fino al raggiungimento del primo elemento con un ID. Così la replicazione prevedrà come passo iniziale, il recupero dell'elemento con un certo ID e poi la navigazione fino ad arrivare all'elemento in questione.

Con l'ultimo metodo illustrato non abbiamo eliminato totalmente i problemi ma li abbiamo ridotti notevolmente. Osservando i documenti HTML del test case, ci si accorge che la maggior parte di essi, usano gli ID per identificare elementi contenitori di sezioni e sottosezioni. Se così fosse saremmo sensibili solo all'aggiunta o rimozione di elementi contenuti nella sotto-sezione dell'elemento in questione.

Osserviamo l'albero DOM riportato in Figura 26, rappresentante l'albero DOM generato dalla home page del wiki mediawiki.

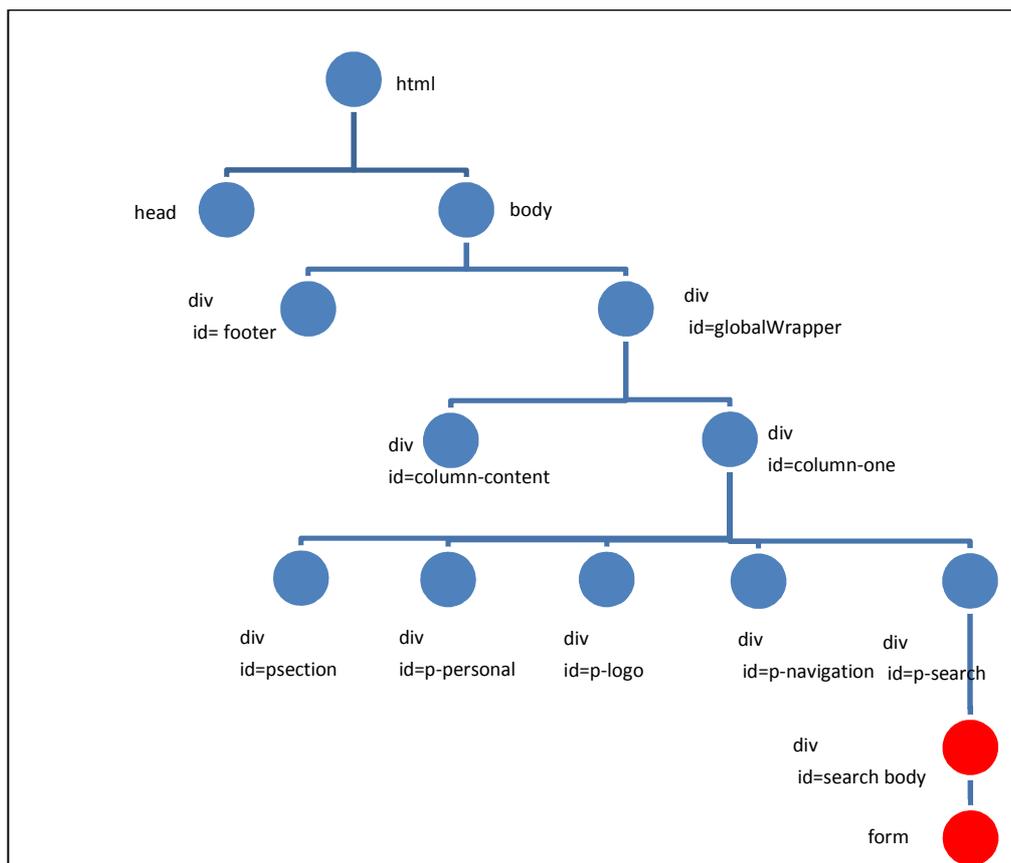


Figura 26 : DOM mediawiki

Nell'albero in questione sono riportati solo gli elementi con ID. Come vediamo gli ID sono utilizzati per identificare le varie sezioni della pagina. Gli

elementi rossi rappresentano un div con id=search-body, questo è il div contenitore degli elementi per la ricerca d'informazioni e al suo interno abbiamo un elemento form senza ID.

Supponiamo di voler identificare l'elemento form utilizzato per la ricerca d'informazioni all'interno del wiki. Dovendo calcolare l'intero path, che va dall'elemento form fino al nodo radice HTML, produrremmo un path sensibile alle modifiche di molte sezioni del sito. Ad esempio saremmo sensibili all'aggiunta di nuovi elementi all'interno del div column-one, dove basterebbe inserire un elemento contenitore di un banner pubblicitario per rendere inutilizzabile la procedura registrata.

Invece se calcoliamo il path fino a incontrare il primo elemento con ID, il path prodotto avrebbe come elemento radice il div id= search-body e sarebbe più resistente alle modifiche. Infatti le uniche modifiche che potrebbero invalidarlo sono quelle degli elementi contenuti nel div radice, quindi modifiche alla sottosezione dedicata al form di ricerca.

Oltre all'identificazione dell'elemento all'interno dell'albero DOM sono necessari anche altri dati. Durante la fase d'implementazione si sono riscontrati dei problemi nel recuperare l'elemento. I problemi erano dovuti al fatto che gli elementi molto spesso non sono presenti all'interno dell'albero DOM ma vengono aggiunti o modificati dinamicamente. Questo accade con alcuni elementi dell'interfaccia dei case test Google Document e in particolare PizzaBo. Infatti, questa è una tecnica che sta prendendo piede con il diffondersi delle applicazioni Web2.0. Tradizionalmente ad ogni azione dell'utente corrisponde la richiesta per una nuova pagina. Invece, nelle applicazioni Web 2.0, la pagina viene modificata, probabilmente in seguito ad un richiesta asincrona per recuperare i dati archiviati in un server remoto. In questo scenario, per la corretta replicazione di una procedura, non è sufficiente identificare un elemento all'interno dell'albero DOM, perché questo potrebbe non essere ancora nello stato corretto o potrebbe addirittura non essere ancora presente. Così si causa la generazione di un evento su un elemento non ancora nello stato corretto o addirittura non presente causando l'interruzione dell'esecuzione della procedura.

Le difficoltà derivano dal fatto di voler modellare il processo di navigazione dell'utente. Di fronte ad una applicazione Web di questo tipo, l'utente attende finché non compare l'elemento in questione o ha dei messaggi visivi che gli suggeriscono di attendere finché l'applicazione carichi gli elementi necessari per l'operazione successiva. La nostra applicazione in fase di replicazione non può accorgersi di questa situazione. Sarebbe inutile anche tentare un approccio basato sui tempi di attesa, ad esempio cercando di calcolare il tempo tra un evento e quello successivo otterremmo dei dati non significativi, in quanto i tempi dipendono dallo stato della rete, del server e dalla macchina su cui è in esecuzione l'applicazione.

La soluzione è quella di archiviare lo stato degli elementi, ed in fase di replicazione della procedura, nel caso in cui gli elementi non abbiano lo stato desiderato o non siano presenti, mettere l'applicazione in una fase di polling. Questo significa costringere l'applicazione a fare verifiche sullo stato dell'elemento a intervalli regolari di tempo, fino ad arrivare ad un limite di tempo massimo. Nel caso in cui si raggiunga questo limite e l'elemento non fosse ancora stato recuperato o non abbia ancora raggiunto lo stato corretto, la procedura non sarebbe andata a buon fine e l'esecuzione del programma verrebbe interrotta. Quindi oltre all'IDpath vengono archiviati anche dati che rappresentano lo stato degli elementi.

6.6.3 Identificazione dell'evento DOM

Ogni volta che nel documento viene generato un evento DOM questo viene intercettato ed analizzato dall'applicazione. Per replicare una procedura basata su eventi DOM abbiamo bisogno solo di una piccola parte di tutti gli eventi che possono essere generati in un documento. Come detto in precedenza siamo interessati a modellare il processo di navigazione, quindi intercetteremo solo eventi che ci permettano raggiungere tale scopo. Un evento DOM potrebbe essere visto come uno step di una procedura. E' importante per questo selezionare i giusti eventi e memorizzare gli attributi importanti per la replicazione dello step.

Alla generazione di un evento corrisponde la creazione di un oggetto event che viene passato in ingresso alla funzione eventHandler addetta alla gestione dell'evento. Ogni oggetto event ha una grande quantità di dati che descrivono molti aspetti dell'evento a cui noi non siamo interessati. Per questo dobbiamo riuscire ad archiviare soltanto i dati utili a poterlo replicare. In aggiunta gli oggetti event hanno degli attributi differenti che dipendono dalla categoria di appartenenza dell'evento. Esistono diverse categorie di eventi definite in DOM level 2 Events [28] che sono:

- UI Event: generati durante l'interazione con il documento.
- Mouse Event: generati quando il mouse viene usato per interagire con il documento.
- KeyBoard Event: generati quando si usa la tastiera per interagire con il documento.
- HTML Event: generati quando avviene un cambiamento nella finestra.
- Mutation Event: generati quando avviene una modifica alla struttura DOM.

Il primo evento utile è sicuramente l'evento "load", che ci permette di intercettare il caricamento di un nuovo documento. Ogni volta che l'utente richiede un nuovo documento seguendo un link o andando a digitare direttamente l'URL nella barra degli indirizzi, questo viene caricato. L'evento load è generato nel momento in cui sono state caricate tutte le risorse esterne contenute nel documento come ad esempio css, immagini, ecc. Con questo evento possiamo ricostruire il percorso di navigazione dell'utente.

Quando viene generato un evento load, per prima cosa deve essere analizzato. Infatti non siamo interessati a tutti gli eventi load di un documento ma solo a quelli relativi al frame contenitore. Ad esempio, un documento costituito da più frame o iframe genera più eventi load, esattamente uno per ogni documento caricato. Quindi se il documento in questione contiene un iframe, questo genererà due eventi load, il primo relativo al documento contenuto nell'iframe ed un altro per se stesso. Per il nostro scopo siamo interessati solo al

load del documento contenitore, in quanto il load del documento interno verrà eseguito in automatico ogni volta si caricherà il documento principale.

Per fare questo controllo e ignorare i load a cui non siamo interessati, recuperiamo l'oggetto `document` attraverso l'attributo `target` dell'evento. Il `target` di un evento `load` sarà sempre un elemento `document`. Una volta in possesso del `document`, possiamo accedere alla finestra contenitore attraverso l'attributo `defaultView`, da qui possiamo controllare se si tratta di un `frame` o di un contenitore attraverso la proprietà `frameElement`. Se supera il controllo, andiamo ad archiviare il `referrer` contenuto nel `document`, che indica il documento da dove è partita la richiesta. Questo è l'elemento di cui si necessita per poter ricostruire la navigazione.

Gli eventi necessari per replicare la maggior parte delle azioni dell'utente compiute su un documento, sono quelli della categoria `MouseEvent`. Le azioni che siamo intenzionati a replicare con questi eventi sono quelle relative all'attivazione di elementi e quelle relative al `drag and drop`, cosa necessaria visto il diffondersi di applicazioni Web 2.0. Gli eventi a cui siamo interessati sono `mousedown`, `mouseup`, `click` e `mousemove`. Questi eventi, come tutti quelli della categoria `MouseEvent`, hanno attributi contenenti le coordinate che identificano in maniera esatta il luogo di generazione dell'evento.

Gli eventi `MouseEvent` hanno due tipi di attributi per l'archiviazione delle coordinate che sono:

- `clientX`, `clientY`: identificano le coordinate all'interno della parte di finestra visualizzata nel browser.
- `screenX`, `screenY`: identificano le coordinate all'interno dell'oggetto `screen` che contiene la pagina.

Nel riprodurre questi eventi utilizzando le coordinate registrate, potrebbero verificarsi dei problemi. Ci si riferisce al caso in cui una procedura venga replicata in una macchina differente da quella di registrazione. In questo caso potrebbero verificarsi problemi dovuti alla differente risoluzione dei monitor, o ai differenti livelli di zoom impostati nel browser.

Eseguendo delle prove su una pagina contenente un "image map", si verificano problemi solo se il mapping dell'immagine viene effettuato server-side. Infatti, nel caso in cui l'immagine sia mappata client-side e le

coordinate per la replicazione del click siano errate, l'evento viene comunque riprodotto sull'esatto elemento AREA, recuperato nel DOM grazie all'uso dei path. Attivando l'esatto elemento AREA viene innescata correttamente l'azione a esso associata, anche se le coordinate effettivamente non corrispondevano a questo elemento. Invece, il problema si pone nel caso d'immagine mappata server-side, dove il server potrebbe ricevere delle coordinate non corrette.

Altre prove sono state effettuate per la replicazione di eventi associati al drag-and-drop. Il case test scelto è la pagina con gli esempi JQuery della sezione UI. Registrando una procedura, si è notato che essa viene replicata correttamente anche su macchine con risoluzione e livello di zoom totalmente differente da quello impostato nella macchina originale. Questo è dovuto a una particolare gestione del drag-and-drop di JQuery. Infatti, JQuery, calcola il livello di spostamento dell'elemento in base alla differenza di posizione degli eventi mousemove, che vengono ricalcolati di volta in volta. Questo significa che uno spostamento dal punto 5,5 al punto 20,20 determina uno spostamento di 15 punti in entrambe le posizioni, e la nuova posizione dell'elemento verrà ricalcolata aggiungendo 15 punti ad entrambe le coordinate dell'elemento. Con questo sistema le coordinate non vengono utilizzate come punti fissi ma solo per calcolare il livello degli spostamenti. Questo determina che anche procedure registrate con diverse risoluzioni possano essere replicate correttamente.

Ovviamente il caso particolare di JQuery non esclude che occorran problemi anche in altri sistemi di drag-and-drop.

Per ovviare a questi problemi si è deciso di gestire tutti gli eventi di tipo click utilizzando sempre le coordinate 0,0. Invece vengono utilizzate le coordinate reali per gli eventi mousemove, mouseup, mousedown, di solito utilizzati per le funzioni drag and drop.

Questa decisione deriva dall'importanza dell'evento click, il quale di solito viene utilizzato per l'attivazione di elementi, operazione molto più frequente e importante rispetto alle operazioni di drag and drop. L'azione di attivazione degli elementi viene considerata fondamentale per la replicazione di una procedura, e spesso inutile per il drag-and-drop. Per questo la soluzione che minimizza i problemi è quella di lasciare le coordinate 0,0 all'evento click. In questo modo non genereremo nessun problema per l'attivazione degli elementi

e lasceremo le coordinate originali per gli eventi di solito utilizzati nel drag-and-drop, dove è fondamentale lavorare con le coordinate .

Ovviamente questa soluzione garantisce sempre la corretta replicazione per eventi di attivazione ma allo stesso tempo non garantisce sempre la corretta esecuzione di azione legate al drag and drop.

Un altro evento importante da gestire è l'evento change, generato quando avviene la modifica di un qualunque campo di input. Siamo particolarmente interessati a quest'evento, in quanto ci permette di identificare gli elementi di input che vengono modificati in un documento. Grazie ad esso saremo in grado di selezionare chiaramente gli step che poi saranno modificabili dall'utente. L'elemento input HTML viene utilizzato appunto per permettere all'utente di inserire input nel documento, ogni modifica di questi elementi genererà un evento di tipo change.

Non è necessario monitorare i cambiamenti di tutti i tipi di input, perché alcuni di questi in realtà vengono utilizzati per diversi scopi, come ad esempio un campo di input di tipo "hidden", viene utilizzato normalmente per creare delle variabili nascoste all'interno della pagina. I campi di input a cui siamo interessati sono quelli di tipo:

- text
- password
- checkbox
- radio
- file

Gli eventi change su altri tipi di input possono essere tranquillamente ignorati. Quindi, ogni volta che viene intercettato un evento change viene analizzato, osservando se il tipo dell'elemento target corrisponde ad uno dei tipi a cui siamo interessati.

I dati che ci interessa archiviare sono quelli relativi ai nuovi valori dei campi di input. Per ottenere questi dati bisogna accedere ad attributi differenti a seconda del tipo di campo di input. Text, password e file conservano il loro valore all'interno dei campi value, mentre gli altri , cioè quelli cliccabili, hanno il valore archiviato nell'attributo checked.

Un discorso differente è necessario per gli input di tipo radio. Infatti, non è sufficiente archiviare lo stato dell'elemento radio per poter replicare una procedura, ma è necessario quello di tutti gli elementi radio presenti nel documento. Infatti, al momento di replicare una procedura l'utente potrebbe avere la necessità di modificare la selezione del radio button. Nel caso di tale input, è necessario presentare all'utente tutte le possibili scelte, quindi tutti i radio button con lo stesso nome contenuti nel documento. Per questo al verificarsi di un evento change su un elemento radio, è necessario archiviare informazioni su tutti i campi di input radio.

Oltre a quelli elencati, è necessario archiviare anche altre informazioni sui campi di input, come ad esempio il nome o la label contenenti la loro descrizione, in modo da avere ulteriori informazioni. Insieme al campo di input vengono archiviate informazioni contenute nell'attributo name. Anche questa soluzione però non è priva di problemi. Infatti, il campo name contiene il nome della variabile che sarà inviata al server e non il nome del campo di input visualizzato dall'utente nella pagina. Il valore contenuto nel campo name potrebbe non essere rappresentativo o potrebbe non cogliere il significato del input in questione. Per questo, si cerca di prendere maggiori informazioni memorizzando anche eventuali testi, prima o dopo del campo di input.

6.6.4 Identificare eventi in dialogBox

Oltre a dover identificare tutti gli eventi contenuti all'interno dei normali documenti, è necessario identificare anche eventi DOM che avvengono all'interno di dialog box. Infatti, in molte applicazioni Web, sono utilizzate dialog box durante l'interazione, ad esempio per accettare particolari condizioni di conferma o semplicemente per mostrare messaggi di errore.

Il problema nasce dal fatto che stiamo sviluppando una tecnica, per modellare il processo di navigazione che avviene all'interno di una finestra con differenti Tab. Mentre le dialog box sono di fatto nuove finestre su cui noi non abbiamo il controllo. Durante il processo di sviluppo si sono verificati problemi nel tentare di replicare una procedura per la creazione di un documento su

Google Documenti, in quanto questa operazione coinvolge l'uso di una dialog box per l'inserimento del titolo.

Il problema deriva dal fatto che al momento d'inizio registrazione, possiamo monitorare eventi che si generano nella finestra del browser attualmente aperta e non in quelle che si apriranno. In oltre le dialog box comuni, come ad esempio quelle aperte in seguito a comandi Javascript, come alert(), confirm(), prompt() sono dialog box preimpostate e salvate in locale su dei file XUL. Non abbiamo eventi che ci indichino in maniera diretta quando avviene l'apertura di una nuova dialog box.

La soluzione a questo problema sta nel mettersi in ascolto sull'evento "blur", generato ogni qual volta la finestra principale dell'applicazione passa in secondo piano e perde il focus. Questo è il momento in cui potrebbe aprirsi una nuova dialog box. Quindi ogni volta che si verifichi l'evento blur è necessario andare ad osservare se ci sono nuove finestre aperte, in caso affermativo registreremo i normali eventi DOM all'interno della nuova finestra.

6.6.5 Dagli eventi DOM a una procedura

Dal monitoraggio degli eventi sopra citati siamo in grado di ricavare tutti i dati necessari alla creazione degli step che compongono una procedura.

Nella tabella riportata in Figura 27 sono riassunti tutti gli eventi necessari a replicare una procedura. Per ogni evento viene riportato il controllo da effettuare, e sarà utile alla creazione della procedura solo se supererà il controllo. In aggiunta sono riportati i dati da archiviare in caso di superamento del controllo ed il target, cioè il tipo di elemento che genera l'evento in questione.

Come possiamo vedere dalla tabella, tutti gli eventi di tipo MouseEvent vengono ignorati nel caso in cui si siano verificati su elementi di tipo input. Questo, per evitare che in fase di replicazione della procedura i campi di input vengano selezionati due volte, prima con un change input e poi con un MouseEvent, determinando in alcuni casi un erroneo settaggio dell'input.

| Evento | Controllo | Dati da archiviare | Target |
|-----------|--|---|--------------------|
| Load | Verificare che il load sia di un documento contenitore attraverso i frame del window | URL Document.referer | Document |
| Mousedown | Verificare che il target non sia un elemento di input | URL document Id tab Stato elemento Position Coordinate x,y | Tutti elementi DOM |
| Mouseup | Verificare che il target non sia un elemento di input | URL document Id tab Position Coordinate x,y Stato elemento | Tutti elementi DOM |
| Click | Verificare che il target non sia un elemento di input | URL document Id tab Position Stato elemento | Tutti elementi DOM |
| Mousemove | Verificare che il target non sia un elemento di input | URL document Id tab Position Coordinate x,y Stato elemento | Tutti elementi DOM |
| Change | Verificare che sia un elemento di input | URL document Id tab Position Stato elemento Nuovo valore del campo di input Nel caso di uno step radio è necessario riportare tutti gli elementi radio | Elementi input |
| Blur | Verificare che ci sia una dialog box aperta | | Document |
| TabOpen | | Id Tab | Tab Browser |
| TabMove | | Id Tab | Tab Browser |
| TabClose | | Id Tab | Tab Browser |

Figura 27: Tabella degli eventi

Partendo da questi dati è possibile definire un documento XML, che modelli la procedura e gli step. Nella Figura 28 è riportato un esempio di documento XML che descrive una procedura.

L'elemento radice è "procedure" e al suo interno sono annidati 4 elementi che sono :

- description, una descrizione della procedura.
- creator, nome del creatore di una procedura.
- initSteps. Contiene gli step iniziali.
- steps. Contiene tutti gli step di un procedura.

Gli initStep, sono degli step speciali che vengono eseguiti prima della replicazione della procedura, servono a predisporre il browser all'esecuzione

della procedura. L'elemento `initStep` contiene una serie di step i quali hanno solo il nome dell'URL e l'ID del Tab da aprire.

```
<procedure id="NewProcedura">
  <description/>
  <creator>robot2</creator>
  <initSteps>
    <initStep>
      <uri>http://www.repubblica.it/</uri>
      <index selected="true">HMTab-0</index>
    </initStep>
  </initSteps>
  <steps>
    <step type="event">
      <uri>https://docs.google.com/#all</uri>
      <index>HMTab-0</index>
      <typeEvent>mousedown</typeEvent>
      <idposition>ID:m/^/0/^/0/^/0/^/1/^/1/^/</idposition>
      <absposition>HTML/^/1/^/11/^/0/^/0/^/0/^/0/^/</absposition>
      <label/>
      <time>22661875</time>
      <extra>380/225/84/94</extra>
      <state>undefined/undefined/undefined/visible/inline-block</state>
    </step>
  </steps>
</procedure>
```

Figura 28: Procedura XML

Nell'elemento `steps`, sono contenuti tutti gli step che compongono la procedura. Nell'esempio riportato in Figura 28, `steps` contiene un unico elemento `step`. Un elemento `step` contiene tutti i dati ripostati nella tabella di Figura 27, più alcuni elementi necessari per replicare una procedura.

Gli elementi `step` hanno un attributo `type` che identificano il tipo di step. Quest'attributo, può avere due possibili valori: `event` e `DialogEvent`. Servono a capire se l'evento si è verificato in una dialog box o meno. Gli elementi contenuti nello `step` sono i seguenti:

- `URL`, contiene l'URL del documento su cui riprodurre lo step.
- `index`, contiene l'ID del Tab su cui si è verificato l'evento.
- `idposition`, contiene il path calcolato secondo il metodo degli ID.
- `absposition`, contiene il path della posizione calcolato senza il metodo degli ID.
- `typeEvent`, il tipo di evento da riprodurre in questo step.

- label, nel caso in cui sia un evento change, contiene la label dell'elemento input, o altre label assegnate dall'utente. Per altri tipi di eventi questo elemento è vuoto.
- time, il momento in cui si è verificato l'evento utile per riordinare gli eventi in ordine temporale.
- extra, informazioni utili per riprodurre l'evento che cambiano in base al tipo di evento. Ad esempio, in un MouseEvent conterrà le coordinate e in un input invece il nuovo valore del campo di input.

Per quanto riguarda gli eventi di tipo radio è stato necessario creare una struttura differente. In Figura 29 è riportato uno step di tipo radio, come possiamo vedere ha una struttura differente.

```

<step type="radio">
  <uri>file:///C:/Users/capo/Desktop/init.html</uri>
  <index>HMTab-0</index>
  <typeEvent>radio</typeEvent>
  <label>Select</label>
  <name>sex</name>
  <time>2</time>
  <selected>1</selected>
  <elements>
    <element>
      <label>Male</label>
      <idPath>HTML/1/9/7/</idPath>
      <absPath>HTML/1/9/7/</absPath>
      <state>>false/undefined/undefined/visible/inline</state>
    </element>
    <element>
      <label>Female</label>
      <idPath>HTML/1/9/11/</idPath>
      <absPath>HTML/1/9/11/</absPath>
      <state>>false/undefined/undefined/visible/inline</state>
    </element>
  </elements>
</step>

```

Figura 29: Radio step

La creazione di uno step di tipo differente per il settaggio di un elemento radio, è dovuta al fatto che in questo specifico caso, è necessario archiviare

informazioni non solo sull'elemento su cui si è verificato l'evento, ma anche sugli altri elementi di tipo radio presenti nel documento. A differenza degli step normali, uno step radio contiene l'elemento "elements", con la lista degli altri elementi radio, la loro label, il loro path e il loro stato. In oltre uno step radio contiene l'elemento selected che indica qual è l'elemento selezionato.

Quella appena descritta è la struttura del documento XML, che descrive una procedura con tutti i suoi step. Questo è il documento che viene prodotto dalla fase di monitoraggio ed è anche il documento che viene preso in input dal DOM player per la replicazione di una procedura. Contiene tutte le informazioni di una procedura, può essere salvato su disco o inserito in una pagina wiki, il che permette di esportare le procedure fuori dall'applicazione e di condividerle.

6.7 Monitorare Eventi DOM

Durante la fase di registrazione di una procedura, vengono monitorati gli eventi DOM che si verificano in ogni documento aperto nel browser. Monitorare le attività serve ad archiviare i dati necessari per replicare le procedure. L'obiettivo del componente che si occupa di monitorare gli eventi è quello di produrre l'xml presentato nel precedente paragrafo.

Per avere disporre di tutti i dati utili a creare una procedura è necessario avere accesso ai documenti aperti all'interno del browser e ad alcuni componenti XPCOM, che permettono di monitorare eventi sui Tab e di accedere a tutte le dialog box attualmente aperte.

Firefox permette di accedere a tutti i documenti aperti nel browser, solo se stiamo eseguendo operazioni da una finestra che ha i privilegi Chrome. Visto che stiamo sviluppando una estensione e siamo un possesso di tutti i privilegi, possiamo accedere alla variabile globale gBrowser che ci da accesso a tutti i Tab aperti in Firefox.

Gli eventi DOM sono monitorati aggiungendo degli eventListner su tutti i documenti aperti nel browser. DOM Level 2 [28] definisce dei metodi per l'aggiunta e la rimozione di eventHandlers che permettono di assegnare più di un eventHandler sullo stesso tipo di evento e sullo stesso elemento. Questo ci

permette di mantenere inalterati gli eventHandler già presenti nel documento e di aggiungerne di nuovi per monitorare gli eventi.

È necessario inserire un eventHandler per ogni evento da monitorare e posizionarli sull'elemento con livello gerarchico superiore, in modo da poter intercettare tutti gli eventi di un certo tipo, con un unico eventHandler. Questo ci è permesso grazie alla propagazione degli eventi all'interno dell'albero DOM. Generando un evento, esso attraverserà tutto l'albero fino a arrivare all'elemento target. Per questo motivo mettendoci in ascolto sull'elemento gerarchicamente superiore intercetteremo gli eventi su tutti gli elementi dell'albero.

Gli eventHandler vengono posizionati in ascolto sulla la fase di capturing di un evento, in modo da essere sicuri di poter catturare un evento. Infatti, durante la fase di bubbling l'elemento target potrebbe fermare la propagazione dell'evento.

Per riuscire a produrre il documento XML della procedura abbiamo bisogno di monitorare tutti gli eventi indicati nel paragrafo 6.6.4. Come mostrato nella tabella in Figura 30 per ogni evento viene posizionato un eventHandler che si occupa della gestione di un evento su tutto il documento. Ogni eventHandler gestisce l'evento passandolo alla funzione addetta alla gestione di tale evento.

| Evento | Posizione | Funzione |
|-----------|--------------|------------------|
| Load | TabBrowser | ManagerDomEvent |
| Mousedown | TabBrowser | ManagerDomEvent |
| Mouseup | TabBrowser | ManagerDomEvent |
| Mousemove | TabBrowser | ManagerDomEvent |
| Click | TabBrowser | ManagerDomEvent |
| Change | TabBrowser | ManagerDomEvent |
| Blur | Document | WindowEnumerator |
| Tab Open | TabContainer | ManagerTabEvent |
| Tab Move | TabContainer | ManagerTabEvent |
| Tab Close | TabContainer | ManagerTabEvent |

Figura 30: Tabella Eventi

Gli eventHandler che si occupano della gestione degli eventi sul documento vengono piazzati ad un livello gerarchico superiore rispetto a

quello del document, in modo tale da non doverli sostituire ad ogni load di un nuovo documento. Invece, gli eventHandler relativi alla gestione dei Tab, sono posizionati sull'elemento del browser TabContainer, addetto alla gestione dei Tab e in grado di generare eventi su ogni azione compiuta come apertura e chiusura di Tab.

L'unico eventHandler che deve essere necessariamente posizionato nel document è quello relativo all'evento blur, in quanto nel caso di apertura di una dialog box sarà il documento a perdere il focus e non il Tab.

6.8 Riprodurre eventi DOM

Per la riproduzione di eventi è necessario sviluppare un componente che data la descrizione XML della procedura sia in grado di replicarla.

Il primo problema nella replicazione di una procedura è riuscire a identificare gli eventi espliciti e impliciti. Con step espliciti, ci riferiamo a eventi che devono essere generati dal player, invece con step impliciti ci riferiamo ad eventi che verranno generati dal documento, come conseguenza di un evento esplicito. Quindi gli eventi espliciti devono essere generati direttamente dal player mentre gli eventi impliciti vengono generati dal documento.

Ricordiamo che replicare procedure con l'approccio DOM, significa forzare l'interfaccia a compiere certe azioni, emulando il comportamento dell'utente. A ogni azione dell'utente corrisponde una reazione dell'interfaccia, quindi quando generiamo una serie di eventi su un'interfaccia, dobbiamo aspettarci una reazione.

Attendere l'esecuzione di eventi impliciti è fondamentale per portar a termine la replicazione di una procedura. Un evento implicito potrebbe essere il caricamento di un nuovo documento o la modifica del documento con il quale stiamo interagendo. E' fondamentale attendere la corretta esecuzione di questi eventi per procedere con la replicazione di una procedura, altrimenti potremmo trovarci nella situazione di replicare eventi su un'interfaccia non aggiornata.

Per fare questo è necessario selezionare gli step che possono essere eseguiti in successione e quelli dove è necessario fermare il player per attendere

il verificarsi dell'evento. Questo implica che il player sia in grado, sia di riprodurre che di monitorare gli eventi.

La prima operazione eseguita dal player, è un'analisi della procedura da riprodurre. In questa fase vengono analizzati gli step e una procedura viene divisa in varie sotto-procedure, in modo da creare gruppi i cui step che possano essere eseguiti in successione. L'idea è di creare gruppi di step, dove sia necessario fermare il player alla fine dell'esecuzione del gruppo e successivamente mettersi in ascolto per attendere il verificarsi di un evento prima di passare al gruppo successivo.

Una procedura viene "tagliata" in presenza dei seguenti step con eventi impliciti:

- Load.
- Stop.
- DialogOpen.

In presenza di questi eventi viene creato un nuovo gruppo dove gli eventi impliciti sono l'ultimo step del gruppo. In questo modo se l'ultimo step è un load, il player si ferma attendendo il verificarsi di un load prima di eseguire il gruppo successivo.

In alcuni casi il player stesso, durante l'esecuzione degli step, è in grado di riconoscere alcune situazioni, dove è necessario fermare l'esecuzione e mettersi in attesa. Ad esempio, in presenza di step che implicano l'impostazione di input durante l'esecuzione della procedura. In questi casi il player si ferma, crea un nuovo gruppo di step con i restanti del vecchio gruppo e lo mette in testa alla lista di gruppi da eseguire.

Nei casi in cui il player è in fase di attesa a causa di step di tipo stop o input, sarà l'utente con una sua azione a far ripartire l'esecuzione del player. Mentre, in caso di stop dovuto all'attesa di un evento di tipo load, è necessario attendere l'esecuzione dell'evento ed eseguire controlli per verificare se effettivamente è il load del documento di cui siamo interessati.

A ogni load dobbiamo verificare che sia di un frame contenitore e che l'URL del documento caricato corrisponda all'URL del documento atteso.

Infatti, potrebbero verificarsi dei casi in cui il caricamento corrisponda ad altri documenti di cui non siamo in attesa.

Effettuare il confronto tra gli URL è una operazione che non può essere effettuata confrontando interamente le due stringhe. In alcuni casi il load del documento con l'URL atteso, non si verificherà mai. Sono i casi dove l'URL del documento caricato, cambia sempre a ogni ripetizione della procedura. Ad esempio, replicando la procedura che crea una nuova pagina wiki, l'URL della nuova pagina creata sarà sempre differente. Inoltre ci sono casi dove la procedura implica l'uso di un motore di ricerca, dove la pagina dei risultati probabilmente ha un URL che dipende dai campi di input inseriti nella ricerca.

Per questo è necessario eseguire un confronto parziale sugli URL e confrontare solo la prima parte dell'URL eliminando la query string e una parte del path. In questo modo riusciamo ad essere più flessibili e ad accettare anche URL simili eliminando la possibilità di confondersi con URL indesiderati.

In alcuni casi il load non è conseguenza di un evento esplicito contenuto nella procedura, ma di un evento esterno alla procedura. Sono i casi in cui un documento viene caricato in seguito alla digitazione dell'URL nella barra degli indirizzi. Il player si accorge di questa situazione dalla mancanza dell'attributo referer nello step load. In questo procederà ad effettuare la richiesta del nuovo documento prima di mettersi in attesa.

Nello stesso modo, ci sono casi in cui l'apertura di un nuovo Tab è un evento esplicito e non implicito. Il player si accorge di questa situazione dal fatto che l'evento TabOpen non è seguito dal TabMove, in questo caso provvede manualmente all'apertura del nuovo Tab e alla richiesta di un nuovo documento. Mentre invece se l'evento TabOpen è seguito direttamente da un load si mette in attesa degli eventi.

Il player una volta avviato, esegue uno ad uno tutti gli step del gruppo. Quando esegue uno step la prima cosa da fare è recuperare l'elemento e successivamente verificarne lo stato. Una volta superato il controllo dello stato è possibile replicare l'evento sull'elemento in questione.

La prima cosa da verificare è che l'elemento sia presente nel documento e la seconda è che abbia lo stato corretto. Entrambi i test non possono essere eseguiti una sola volta, in quanto i test potrebbero dare esito negativo solo

perché sono stati eseguiti nel momento sbagliato. Ad esempio, gli elementi possono essere aggiunti dinamicamente al documento e bisogna attendere che gli script del documento terminino la loro esecuzione prima di poter dire che il test non è superato.

Come detto nei paragrafi precedenti, non è possibile in nessun modo stimare i tempi di attesa, perché dipendono da molti fattori su cui non abbiamo il controllo.

La routine implementata per fare queste verifiche è quella riportata nel diagramma di attività di Figura 31. I controlli vengono eseguiti più di una volta ad intervalli di tempo regolari. Questo significa che se uno dei due controlli non è andato a buon fine, il player viene fermato e viene impostato un timer per rieseguire il controllo. Prima di poter dire che un elemento non è presente nel documento o che non ha lo stato corretto i test vengono ripetuti un numero determinato di volte.

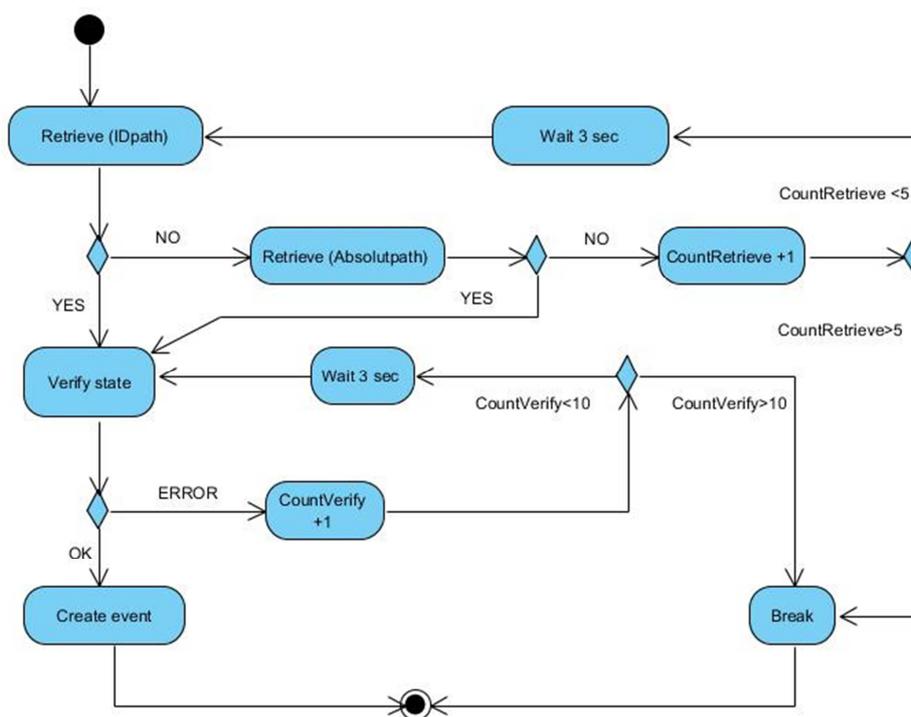


Figura 31: Verifiche sugli elementi

Come mostrato nel diagramma, per prima cosa si esegue un tentativo con l'Idpath, nel caso in cui non sia presente l'elemento, si esegue un altro tentativo

con il path assoluto. Questo perché in alcuni casi potrebbe cambiare anche l'ID degli elementi. Una volta recuperato l'elemento si procede al test dello stato.

Quando l'elemento viene recuperato ed è nello stato corretto è possibile creare l'oggetto event e applicarlo all'elemento o nel caso di eventi change impostare nuovamente il campo di input.

La replicazione di una procedura finisce nel momento in cui la lista con i gruppi di step da riprodurre è vuota. In alcuni casi potrebbero verificarsi degli errori durante la replicazione della procedura, questo è dovuto al cambiamento del documento dalla fase di registrazione. Come detto in precedenza lavorando con l'approccio DOM uno degli svantaggi è quello di essere dipendenti dalla struttura dei documenti. Per motivi di sicurezza, l'esecuzione della procedura viene interrotta nel caso in cui si siano verificati problemi durante la replicazione di uno step.

6.9 Architettura WebRecorder

WebRecorder è stato sviluppato prendendo come riferimento il pattern architetturale MVC (Model View Controller).

Nella Figura 32 è riportato il diagramma delle classi dell'applicazione, ovviamente sono riportate solo le classi principali del sistema. La view è rappresentata dalla classe Sidebar. Essa contiene tutte le pagine del sistema ed è il componente incaricato di catturare le azioni dell'utente e comunicarli al controller che si occuperà di gestirli. Il controller è la parte centrale del sistema, qui si concentra tutta la logica dell'applicazione. Il controller ha accesso a tutti i componenti ed è incaricato di istanziarli e di aggiornare l'interfaccia in base allo stato dell'applicazione. Il controller attiva e disattiva i due componenti principali dell'applicazione che sono il Recorder e il Player. Questi componenti implementano gli algoritmi e le operazioni descritte nei precedenti paragrafi.

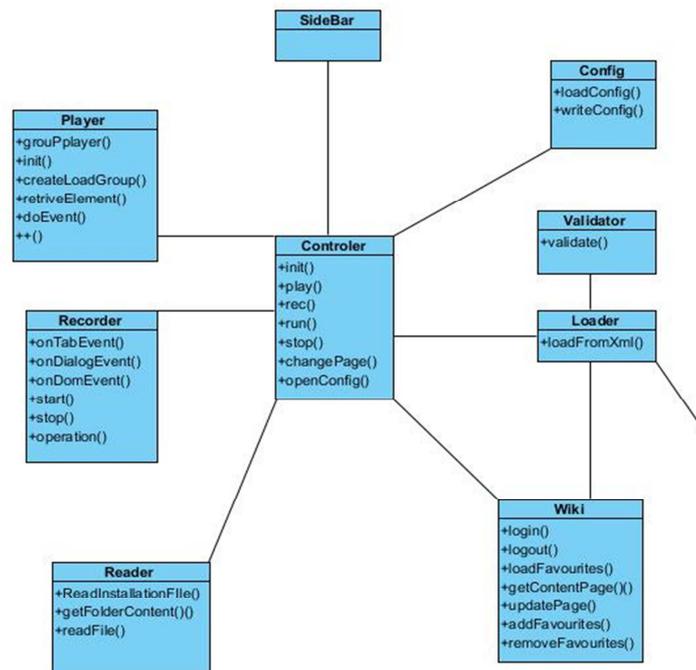


Figura 32: Diagramma delle classi

Il componente config, contiene le informazioni sulla configurazione del sistema. Contiene informazioni importanti per tutti gli altri componenti come ad esempio gli URL del wiki di riferimento e quello del server di validazione. In aggiunta contiene informazioni sul login dell'utente. All'avvio il config si occuperà di recuperare tutte le informazioni dal file config.xml, contenuto nella cartella radice dell'applicazione. Inoltre, si occupa di tenere aggiornato il file di configurazione sovrascrivendo le informazioni ogni volta che esse cambiano.

Il componente WikiGateway è incaricato della comunicazione con il wiki di riferimento. Si occupa della comunicazione con un wiki, implementando metodi per la comunicazione con gli API Mediawiki. Tutta la comunicazione viene generata e ricevuta da questo componente. Si occupa quindi di effettuare il login, l'upload, il recupero di procedure ed inoltre gestisce la sezione personale di ogni utente sul wiki. Un wiki compatibile con WebRecorder, infatti dovrebbe avere una pagina dedicata ad ogni utente, dove sono contenute delle informazioni relative all'utente come le procedure preferite contenute sul wiki. La classe WikiGateway si occupa di gestire queste informazioni aggiungendo o eliminando pagine dai preferiti.

Il reader invece si occupa della gestione dei file sul disco, l'applicazione ha una cartella di default nella quale salva i file delle procedure. Il reader esegue la scansione di questa cartella e carica il contenuto dei file.

Una volta caricati i file delle procedure all'interno dell'applicazione il reader e il WikiGateway li passano al loader, che si occupa di caricarli nel sistema e di aggiornare la lista della applicazioni disponibili. Il loader quindi è il componente incaricato della creazione di oggetti Procedure e degli Step. Una volta istanziata una procedura, il loader si occupa di aggiornare la lista delle procedure caricare nel sistema e l'interfaccia grafica. Prima di aggiornare la lista delle procedure esegue un controllo per verificare se nella configurazione è presente l'opzione di validazione. In questo caso la procedura verrebbe passata al Validator che si occupa di validare la procedura.

La validazione dei componenti non può essere effettuata client-side, perché Gecko non offre gli strumenti per compiere l'operazione. E' stato necessario compiere l'operazione di validazione con l'ausilio di un servizio esterno creato ad-hoc. Ovviamente il servizio di validazione è opzionale e l'URL del servizio può essere modificata dall'utente. Si è scelto di renderlo opzionale, perché in alcuni casi la validazione tramite servizio esterno potrebbe rallentare molto le operazioni di loading di una procedura.

Come mostrato nel diagramma in Figura 33, i servi esterni utilizzati da WebRecorder sono due, uno per la validazione delle procedura e uno per la condivisione delle procedure.

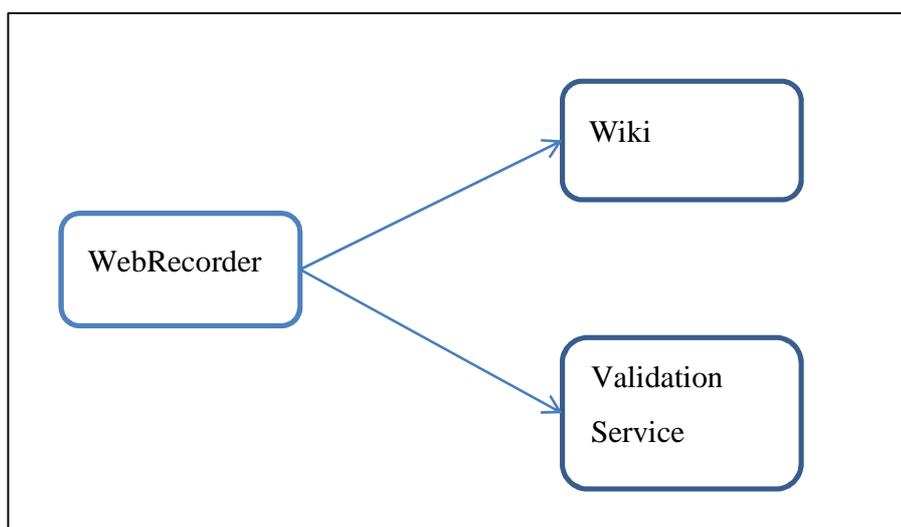


Figura 33: Servizi Esterni

Entrambi i servizi sono opzionali, quindi l'utente può decidere di utilizzare l'applicazione senza utilizzare i servizi del wiki e senza validare le procedure.

7 Conclusioni

In questo lavoro di tesi è stata sviluppata un'estensione Firefox chiamata WebRecorder, per la registrazione e la replicazione di procedure eseguite sul Web attraverso una piattaforma wiki. Le procedure possono essere esportate dall'applicazione e condivise tra gli utenti. L'obiettivo raggiunto con WebRecorder è di creare delle procedure parametriche che possono essere personalizzate, reimpostandone i parametri prima di essere replicate. I parametri corrispondono ai campi di input che possono essere modificati nelle pagine Web coinvolte nella registrazione della procedura. Questo ci permette di avere delle procedure personalizzabili con i propri dati, in modo da poter riadattare una procedura alle esigenze dei vari utenti.

Prima di procedere con lo sviluppo dell'applicazione, sono stati valutati due approcci, uno basato su HTTP ed uno basato su eventi DOM. I due approcci si differenziano tra loro nel modo utilizzato per monitorare le attività dell'utente e nelle modalità di replicazione di una procedura. L'approccio scelto per sviluppare WebRecorder è quello basato su DOM. Questa scelta è il risultato di valutazioni effettuate su prototipi, dove l'approccio DOM è quello che meglio ci permetteva di modellare il concetto di procedura e presentava meno problemi per la replicazione.

Ovviamente anche l'approccio DOM non è privo di problemi, derivati in parte dall'uso non standard degli elementi HTML. Il problema risiede nella possibilità di eseguire operazioni in modi diversi, tutti interpretati correttamente dai browser. Altri problemi derivano invece dalle limitate informazioni presenti negli eventi DOM, a volte insufficienti per i nostri scopi.

Ad esempio, è difficile creare un algoritmo standard che permette di recuperare la label degli elementi di input. Alcune volte, la label è presente all'interno dello stesso elemento, a volte lo precede e altre invece è contenuta in una tabella, nella cella adiacente o superiore al campo di input. Questo esempio ci fa capire come ci siano molti modi per modellare un documento HTML e ottenere lo stesso risultato.

L'introduzione del Web 2.0 e le tecniche Ajax hanno contribuito ulteriormente all'allontanamento dalle linee guida, spingendo verso una direzione meno standard, con nuovi modi per modellare un documento e nuovi pattern di comunicazione .

Con l'introduzione delle tecnologie Ajax, di fatto i documenti cambiano struttura dinamicamente, caricando dati da server remoti senza generare alcun evento di tipo specifico, rendendo difficile capire quando inizia e quando termina la fase di modifica del documento.

In aggiunta si verificano problemi in presenza di tecnologie Web non standard come Flash, Flex e SilverLight, dove non vengono prodotti eventi DOM e i vari elementi vengono trattati come oggetti a scatola chiusa.

In definitiva l'approccio DOM, utilizzato per lo sviluppo dell'applicazione, pone dei forti limiti. I limiti derivano sia dall'uso di tecnologie non standard ma anche dall'uso improprio di tecnologie standard. Per realizzare un'applicazione, che sia in grado di registrare e replicare procedure che coinvolgano qualsiasi applicazione Web, bisognerebbe sviluppare più approcci e implementarli tutti nella stessa soluzione.

8 Indice delle figure

| | |
|--|----|
| Figura 1: Gecko data Flow..... | 6 |
| Figura 2: Fasi del frame constructor | 7 |
| Figura 3: Reflow | 8 |
| Figura 4: XPCOM..... | 13 |
| Figura 5: Struttura cartella root | 30 |
| Figura 6: Overlay | 37 |
| Figura 7: Screen Shot Komodo | 50 |
| Figura 8: Configuration Extension..... | 51 |
| Figura 9: Venkman Debugger | 53 |
| Figura 10: Zoom Loaded Script | 53 |
| Figura 11: Breakpoint | 54 |
| Figura 12: Zoom Lista Variabili | 55 |
| Figura 13 : Zoom Call Stack | 55 |
| Figura 14:Interactive session..... | 56 |
| Figura 15: WebRecorder | 66 |
| Figura 16: Home Page..... | 67 |
| Figura 17: Rec page | 69 |
| Figura 18: Edit Page..... | 71 |

| | |
|---|-----|
| Figura 19: Step setting | 73 |
| Figura 20: Setting radio step | 73 |
| Figura 21: Play page..... | 74 |
| Figura 22: Option page..... | 75 |
| Figura 23: Diagramma Interazione http | 78 |
| Figura 24: Diagramma interazione HTTP Google..... | 79 |
| Figura 25: Albero DOM..... | 83 |
| Figura 26 : DOM mediawiki | 84 |
| Figura 27: Tabella degli eventi | 93 |
| Figura 28: Procedura XML..... | 94 |
| Figura 29: Radio step | 95 |
| Figura 30: Tabella Eventi..... | 97 |
| Figura 31: Verifiche sugli elementi..... | 101 |
| Figura 32: Diagramma delle classi..... | 103 |
| Figura 33: Servizi Esterni | 104 |

9 Bibliografia

- [1] Feldt K., *Programming Firefox: Building Application in the browser*, O'Reilly, 2007
- [2] Bosswel D., King B., Oeschger I., Collins P., Murphi E., *Creating Application whit Mozilla*, O'Reilly, 2002
- [3] Zakas N., *JavaScript for Web developer*, 2 ed., Wrox, 2009
- [4] FlanaganD., *JavaScript, The Definitive Guide*, 5 ed., O'reilly, 2006
- [5] Di Iorio A., Rossi D., Vitali F., Zacchiroli S., *Where are your manners? Sharing Best Community Practices in the Web 2.0*, ACM, 2009
- [6] Little G.,Lau T., Cypher A., Lin J., Haber E.,Kandogan E., *Koala Capture, Share, Automate, Personalize, Business Processes on the Web*, IBM Almaden Research Center, 2007.
- [7] <https://developer.mozilla.org/en/Gecko>
- [8] <http://www.mozilla.org/newlayout/doc/layout-2006-12-14/master.xhtml>
- [9] <http://en.wikipedia.org/wiki/Gecko>
- [10] <http://www.mozilla.org/newlayout/doc/gecko-overview.htm>

- [11] https://developer.mozilla.org/en/Introduction_to_Layout_in_Mozilla
- [12] https://developer.mozilla.org/en/Gecko_Embedding_Basics
- [13] <http://www.mozilla.org/projects/embedding/PublicAPIs.html>
- [14] <http://www.mozilla.org/projects/embedding/PublicAPIs.html>
- [15] https://developer.mozilla.org/en/Creating_XPCOM_Components
- [16] <http://www.ibm.com/developerworks/WebServices/library/co-xpcom.html>
- [17] <http://books.mozdev.org/chapters/ch08.html>
- [18] https://developer.mozilla.org/en/Creating_XPCOM_Components/Usin_XPCOM_Components
- [19] https://developer.mozilla.org/en/how_to_build_an_xpcom_component_in_javascript
- [20] https://developer.mozilla.org/en/XUL_Tutorial/
- [21] https://developer.mozilla.org/en/Building_an_Extension
- [22] <https://developer.mozilla.org/en/Chrome>
- [23] https://developer.mozilla.org/en/Chrome_Registration
- [24] https://developer.mozilla.org/en/XUL_Overlays
- [25] <http://mb.eschew.org/12>
- [26] <http://kb.mozillazine.org/About:config>
- [27] https://developer.mozilla.org/en/Setting_up_extension_development_environment
- [28] <http://www.w3.org/TR/DOM-Level-2-Events/>
- [29] <http://www.w3.org/DOM/>
- [30] http://oreilly.com/pub/a/windows/2005/04/12/automate_xp.html
- [31] <http://office.microsoft.com/it-it/help/definizione-e-utilizzo-delle-macro-HA010007210.aspx>
- [32] http://help.adobe.com/it_IT/photoshop/cs/
- [33] <http://www.iopus.com/imacros/firefox/?ref=fxhome>
- [34] <https://mozillalabs.com/blog/2007/09/coscripiter/>
- [35] <http://greasemonkey.softonic.it/>
- [36] <http://groups.csail.mit.edu/uid/chickenfoot/>