UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica (triennale)

Elaborato Finale

Server Web orientati agli eventi: in direzione di una efficace interazione client server

Relatore:
Prof. Maurizio Marchese

Laureando:
Marco Albano

Correlatore:
Ing. Guido Brugnara

Anno Accademico 2005/2006

Tesi di Laurea

Marco Albano
<albano.marco@gmail.com>

12 febbraio 2007

Indice

Elenco delle tabelle

Elenco delle figure

Sommario

Questo lavoro di tesi nasce in seguito ad un periodo di stage presso l'azienda *Leader.IT*. Avendo come riferimento le problematiche di cui si é venuti a conoscenza durante il periodo di stage, in questa tesi si sono volute analizzare le caratteristiche delle soluzioni produttive per applicazioni web adottate dall'azienda. La *Leader.IT* si occupa della progettazione e dell'implementazione di applicazioni *ad hoc* e di attivitá sistemistica. In particolare sviluppa applicazioni *web based* con strumenti *open source* rilasciando a sua volta il proprio software con licenza *GPL* [5]. Durante il tirocinio si é stati coinvolti nello sviluppo di una parte di un'applicazione seguendo il modello concettuale proposto da *AJAX* [6], ma usando *JSON* [7] in vece del linguaggio di markup *XML* [8]. Ora che il browser sta diventando sempre piú postazione di lavoro oltre che "visualizzatore di pagine web"[1] puó risultare interessante usare gli strumenti che esso mette a disposizione per aumentare l'interazione *client-server*. A tal fine le tecnologie *AJAX* come *DOJO* [9] sono, lato *client*, il metodo migliore di operare. Il problema che l'azienda *Leader.it*, come tutte quelle altre che operano nel suo stesso campo, si trova a dover affrontare, riguarda il lato *server* e le tecnologie che questo deve usare. Per tale motivo la presente tesi sará improntata sull'analisi di diverse soluzioni di cui si descriveranno pregi e difetti fino a giungere ad una proposta di architettura.

[1] http://it.wikipedia.org/wiki/Browser

Capitolo 1

Introduzione

1.1 Descrizione del Problema

La diffusione sempre piú massiccia degli applicativi *web-based* anche per la gestione aziendale ha, nel tempo, portato i *web-developer* a sviluppare o pretendere strumenti sempre piú nuovi ed innovativi per semplificare il proprio lavoro o arricchirlo di funzionalitá e possibilitá. Da qui lo sviluppo di strumenti quali *Tomcat*[2] per *Apache*[1], *AJAX*, *DOJO* ed altri approfonditi in seguito, per rendere le pagine web dinamiche e fortemente interattive. Il passaggio fondamentale di questa evoluzione é la necessitá di arrivare ad una autonomia ed ad una gamma di possibilitá offerte fin'ora solo dal modello *MVC* [11]. Questo passo in avanti non puó essere percorso senza degli accorgimenti, oltre che sul lato *client*, anche sul lato *server*. Per questo, strumenti come il *web-server Apache* hanno bisogno di estensioni quali *Tomcat* per poter gestire pagine dinamiche. Con queste estensioni, comunque, non si puó cambiare il modello concettuale di base di questi progetti che é quello con il quale sono nati. Per le nuove sfide delle moderne Web Application é questo modello concettuale a dover cambiare, in quanto le nuove applicazioni gestionali, sono *web-based*. C'é necessitá di strumenti innovativi ed alternativi che possano supportare gli sviluppatori al meglio delle possibilitá che oggi la programmazione su web é in grado di offrire. Per questo sono da tempo disponibili soluzioni come *Struts*[3]. Questa estensione di Apache é adatta se si utilizza l'ambiente Java come strumento di lavoro. In altri ambienti di sviluppo occorre trovare soluzioni dimili, ma diverse.

1.2 Motivazioni

L'approccio *event-driven* é diverso dall'idea generale che sta alla base della maggior parte delle applicazioni web usate al giorno d'oggi. Le differenze tra uno schema *event-driven* ed uno *request-response* saranno illustrate piú esaurientemente nei prossimi capitoli. In questa sezione ci si limiterá a darne una veloce descrizione riferendosi ai punti principali e tralasciando di andare nei particolari.

1.2.1 Il modello *MVC*

MVC é l'acronimo di *Model View Controller*, modello-vista-controllore. É un *design pattern* che descrive tre livelli diversi per lo sviluppo di applicazioni che devono presentare una gran quantitá di dati agli utenti e Di poterli gestire senza cambiare l'interfaccia grafica oppure cambiare quest'ultima senza andare ad agire sui dati presentati.

Il Modello

É la rappresentazione dei dati su cui l'applicazione lavora. É, in generale, un *database*, od un *filesystem* o qualunque altro meccanismo di persistenza di dati. Il modello *MVC* non pone limitazioni in questo senso, in quanto é importante lo schema di fondo, non le tecnologie che lo implementano.

La Vista

É la rappresentazione grafica del modello, in una forma che sia possibile manipolare. Nel particolare, nel nostro caso, come verrá illustrato in seguito, sará un insieme di tecnologie quali *DOJO*, quindi *AJAX*, ad assolvere questo compito. In generale, in un'applicazione basata su web é *HTML* [12] o meglio *XHTML* [13] il linguaggio usato per descrivere la vista.

Il Controllore

Il controllore é un creatore e gestore d'eventi che hanno la possibilitá di cambiare lo stato sia del modello che della vista.

1.2.2 Il modello *request-response*

Request-response sta per richiesta-risposta. In questo schema ad una azione del *client* corrisponde una risposta dal *server* ed il tutto si svolge in maniera assolutamente sincrona e del tutto *stateless*. Nessuno dei due puó, per esempio, prendere iniziative che influenzino il comportamento dell'altro, o meglio: si puó, si fa giá adesso, ma é un'operazione che ricade nella categoria illustrata sopra e la maniera in cui si mette in atto é puramente un *escamotage* che sopperisce ad una base di strumenti non adatta.

1.3 Struttura della Tesi

- nel capitolo 2 verrá data un'introduzione al web ed alle web application, alle tecnologie usare che verranno esaminate una per una. Quindi si passerá ai web server ed alle tecnologie usate da questi ed a quelle che potrebbero essere utilizzate in vece delle precedenti.

- nel capitolo 3 verrá quindi proposta una nuova architettura appoggiata da motivazioni maturate sulle tecnologie e i prodotti oggi esistenti. Soluzione che verrá proposta in un prototipo di cui sará studiata l'architettura in aperta relazione con le tecnologie rivelatesi piú promettenti ed utili nel corso dello studio che ci apprestiamo ad intraprendere.

- nel capitolo 4 si dará ampio spazio allo stato dell'arte delle tecnologie esistenti ed dei web server event-driven presenti oggi esaminandoli in maniera critica per poterne evidenziare i pregi e gli eventuali difetti.

- nel capitolo 5 saranno tratte le conclusioni in relazione al lavoro svolto nel corso di questa tesi e nel precedente stage presso l'azienda *Leader.IT*.

Capitolo 2

Contesto

2.1 Web & Web Applications

2.1.1 Cenni storici

Il web come lo conosciamo noi é il risultato di un'evoluzione continuativa dal 1991[1], anno in cui Tim Berners-Lee portó a compimento la sua stessa idea avuta due anni prima assieme a Robert Cailliau[2]. Condividere documenti con altri ricercatori, mettendo in rete il primo server web sviluppando il linguaggio HTML con il protocollo *HTTP* [14], ora standard *W3C* [15]. In principio le pagine presentate erano solamente statiche, la semplicitá del linguaggio permise una rapida diffusione del suo uso, pagine con testo e collegamenti ad altri ipertesti, con semplici regole di formattazione prima in ambito accademico, quindi in quello commerciale. Fin da subito, per superare le limitazioni imposte dalla staticitá delle pagine vennero introdotti strumenti per generare pagine dinamiche, ad esempio estraendo delle informazioni da un database. Questo passo venne eseguito con le prime *CGI* [16] che permettevano ai browser di chiamare procedure esterne sul web server e di presentare il risultato sotto forma di pagine *HTML*. Il problema maggiore di questo procedimento é che le richieste fatte alle *CGI* vengono soddisfatte ad ogni invocazione con una chiamata diversa alla procedura atta a gestirla, nessuna ottimizzazione di sorta viene effettuata, né gestione della sessione. Questo limite ha spinto gli sviluppatori delle odierne tecnologie a potenziare gli strumenti su entrambi i fronti. Quello del browser e quello del server.

2.1.2 Il browser

Il primo browser fu sviluppato sempre da Tim Berners-Lee, si chiamava *Nexus* [17] (il primo vero nome era proprio WorldWideWeb, in seguito cambiato in Nexus per via del nome del sistema operativo su cui girava: Next). Le operazioni concesse erano "avanti", "indietro", "trova" e poche altre e supportava il protocollo HTTP e quello *FTP* [18]. Da allora l'evoluzione ha portato molti cambiamenti, ed oggi come oggi, il browser é una vera postazione di lavoro. I browser piú diffusi in questo momento sono *InternetExplorer* [19] e *Mozilla Firefox* [20] che oggi sono giunti alle versioni 7 e 2 rispettivamente. Entrambi hanno supporto per *JavaScript* [22][3] ed hanno *DOM* [21] come interfaccia.

Il Document Object Model

Il DOM é un'interfaccia per il documento che viene visualizzato nella pagina del browser. É una rappresentazione della pagina nelle diverse proprietá che

[1] http://en.wikipedia.org/wiki/World_Wide_Web

[2] all'epoca entrambi matematici del CERN, Conseil Européen pour la Recherche Nucléaire, organizzazione europea per la ricerca nucleare, http://www.cern.ch

[3] trattato in seguito nel capitolo riservato alle tecnologie

la caratterizzano. Per mezzo di questo strumento é possibile interagire con la pagina visualizzata e cambiarne l'aspetto ed il comportamento. Attraverso JavaScript, ad esempio, si possono animare le varie parti della pagina rendendole dinamiche. É ad esempio possibile cambiare l'immagine visualizzata allorché il mouse vi si posiziona all'interno e tornare alla precedente quando il mouse ne esce. In figura 2.1 si nota come il *DomInspector*[4] di Mozilla-firefox

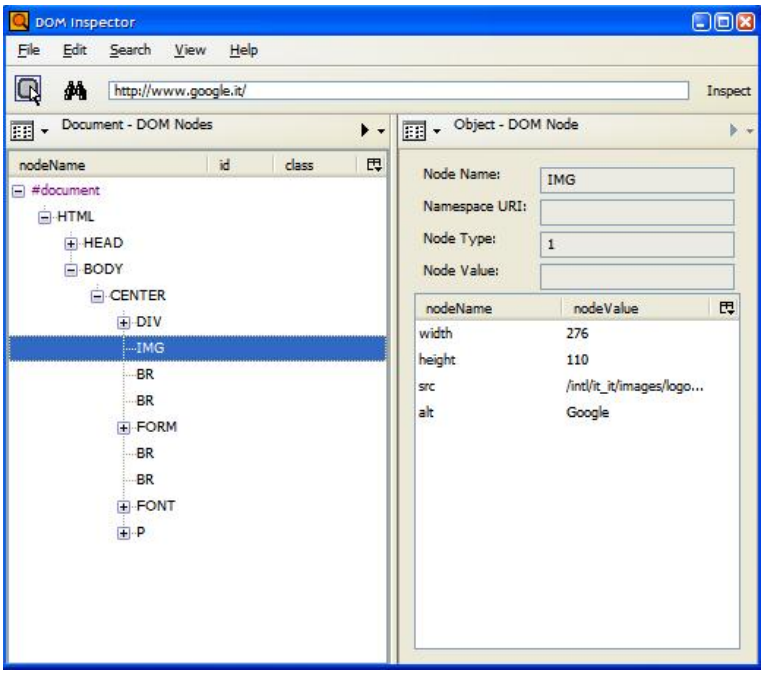

Figura 2.1: Modello DOM della pagina http://www.google.it.

permetta di avere visione della pagina secondo la sua struttura: il primo elemento é il documento che contiene codice HTML, nel riquadro a sinistra infatti viene visualizzato l'albero delle *tag* HTML per ordine di posizione nel codice. La tag *HEAD*, non espansa per problemi di spazio, contiene il titolo della pagina, come in una normale pagina HTML. All'interno della tag "center" che indica l'accentramento di ció che vi é contenuto, si trova un "div", un contenitore, che ha al suo interno l'immagine di Google Italia, due break-line ed il form del motore di ricerca. Nel riquadro a destra invece vengono gli visualizzati gli attributi dell'oggetto selezionato a sinistra che viene chiamato nodo. Ogni oggetto ha caratteristiche diverse e quindi attributi diversi, nel caso specifico l'oggetto rappresentante la tag "img" (image) ha come attributi la larghezza, l'altezza, la sorgente cioé l'URL[5] dell'immagine

[4]programma che permette di visualizzare il DOM della pagina attualmente visualizzata nel browser

[5]uniform resource locator

vera e propria e l'attributo "alternative" che rappresenta la visualizzazione in alcuni casi del "tooltip" o la stringa visualizzata al posto dell'immagine nel caso si usi un browser web testuale. Tutti questi attributi possono essere manipolati via DOM con il supporto di un linguaggio di scripting quale JavaScript. Ció di cui si parlava prima, ad esempio, il *rollover*, il cambio di immagine al passaggio del mouse, con JavaScript che si interfaccia con il browser tramite il DOM, risulta di facile implementazione.

```
<html><head><title>Prova di cambio immagine</title></head>

<body>
    <img id="immagine" alt="prova" src="immagine1.jpg">
</body>

<script language="javascript" defer>
    var img = document.getElementById("immagine");

    img.onmouseover=function(e){this.src="immagine2.jpg";};

    img.onmouseout=function(e){this.src="immagine1.jpg";};
</script>

</html>
```

Da questo esempio si vede come sia semplice eseguire l'operazione descritta. Esaminando i vari passaggi si puó vedere come l'immagine sia contraddistinta da un *id* con il quale, all'interno di uno script, si puó ricavare il nodo. Se si torna a guardare l'albero dei nodi in figura 2.1 si noterá che il primo elemento é *document* ed a questo difatti bisogna riferirsi ogniqualvolta si cerca per id un nodo dell'albero. Una volta trovato il nodo, con due semplici eventi JavaScript, gli si cambia l'attributo *src* del DOM che indica la posizione dell'immagine su disco da caricare e visualizzare. Questo tipo di programmazione, con lo script esterno alla dichiarazione dell'elemento, si usa quando piú oggetti devono avere lo stesso comportamento. Se, al contrario, si vuole che solo un nodo abbia certe caratteristiche l'evento é inseribile all'interno della dichiarazione del nodo stesso. In questo caso si limita lo script solo agli oggetti che lo dichiarano rendendo il codice HTML piú leggibile.

```
<html>
<head>

<title>Prova di cambio immagine</title>

</head>
```

```
<body>

    <img id="immagine" alt="prova" src="Photo-0011.jpg"

    onmouseover="this.src='Photo-0010.jpg';"

    onmouseout="this.src='Photo-0011.jpg';">

</body>

</html>
```

In questo caso, come si puó vedere, l'esecuzione delle righe di script é limitata all'immagine all'interno della cui dichiarazione si trovano. Nonostante si sia usato *this* in questo esempio si puó tranquillamente usare *document* che puó comunque essere invocato ed interrogato per avere il riferimento all'oggetto effettivo[6]. Il DOM, peró non si limita a rappresentare i nodi della pagina HTML. Puó anche rappresentare oggetti del browser. L'attributo *window*, ad esempio, si riferisce alla finestra del browser che puó essere ridimensionata o spostata. L'oggetto *window* ha tutte le proprietá che si possono sfruttare nella finestra di un browser, ad esempio la possibilitá di navigare l'*history*.

```
<html>
<head>
<title>Prova di navigazione con DOM</title>

<script language="javascript">

function browse(button_type){

    switch (button_type) {

        case 1: window.back();
        break;
        case 2: window.stop();
        break;
        case 3: window.forward();
        break;
        case 4: window.home();
        break;
```

[6]per questo procedimento si puó far assumere comportamenti particolari all'oggetto B agendo sull'oggetto A.

```
        default: alert("Wrong code! "+button_type);
    }
}
</script>

</head>

<body>

<input id="indietro" type="button" value="indietro"
    onclick="browse(1);">
<input id="stop" type="button" value="stop"
    onclick="browse(2);">
<input id="avanti" type="button" value="avanti"
    onclick="browse(3);">
<input id="home" type="button" value="home"
    onclick="browse(4);">
</body>

</html>
```

2.1.3 Le web-application

Come é giá stato accennato in precedenza le web-application non sono altro che applicazioni che hanno la particolaritá di avere come interfaccia grafica codice HTML visualizzabile nel browser. Questo é il loro punto di forza, il codice é tutto contenuto nel server e questo significa che non ci sono problemi inerenti l'installazione. L'utente non deve piú preoccuparsi che il software che sta comprando possa effettivamente girare sui computer della sua azienda o del suo ente e che magari si debba comprarne di nuovi e piú potenti. Potrebbe voler significare cambiare molti computer con le spese che questo comporta, o essere limitati a dover cambiare sistema operativo perche' l'unica applicazione che fa al proprio caso funziona su software proprietari e le licenze costano, o si é lavorato con altri strumenti e sarebbe dispendioso in termini di tempo e denaro dover aggiornare il personale. Altro punto di vantaggio sulla logica centralistica del server é quello per cui gli aggiornamenti non imporranno agli utenti di stare nemmeno un minuto senza poter lavorare, o di preoccuparsi di dover porre attenzione all'installazione della patch, o di doversi preoccupare di incompatibilitá tra le varie versioni. Gli utenti non dovranno nemmeno sapere che si sta risolvendo un baco trovato nel software dell'applicazione, non sará loro interesse dato che potrá essere fatto direttamente sul server senza distoglierli dal proprio lavoro. Con particolari accorgimenti nella programmazione della web-application sará altresí possibile evitare accidentali perdite dei dati per disattenzione dell'utente.

Non sará piú possibile infatti perdere dati rilevanti per esempio perdendo o rompendo il computer portatile di un utente. I dati saranno tutti sul server e questo permetterá anche di averne piú controllo ed evitare perdite per disattenzioni o ad esempio virus. Inoltre, sempre sulla stessa linea di pensiero, non é necessario installare nulla sul client che non sia un browser, oramai diffusissimo ovunque. Un altro vantaggio, assolutamente non trascurabile, é che tutti i sistemi operativi hanno un browser, quindi, nonostante quelli OpenSource (Mozilla, Mozilla-Firefox ad esempio, come anche Opera) abbiano quasi sempre implementazioni per tutti i maggiori OS (Microsoft Windows, GNU/Linux e MacOS), ogni sistema operativo implementa il proprio (Explorer, Konqueror, Safari). Questo significa che non si sará nemmeno piú limitati sulla scelta del sistema operativo per essere sicuri che gli applicativi che ci funzionano sopra possano effettivamente girare. Si potranno fare in partenza altre scelte molto meno dispendiose e largamente piú sicure. Nemmeno lo stile é limitato oramai. Il susseguirsi dell'uscita di nuove tecnologie in combinazione con quelle giá esistenti, CSS [23], Flash e gli stessi JavaScript, hanno permesso significativi miglioramenti nella grafica rispetto al puro HTML. In fine, ma non meno importante, un ulteriore vantaggio delle applicazioni web-based é quello per il quale si potrá provare l'applicazione prima di comprarla senza bisogno di alcun appuntamento o intervento esterno. Sará infatti interesse dell'azienda mettere online una versione dell'applicazione a scopo dimostrativo sul proprio sito internet per permettere a chi fosse interessato di provarla senza impegno alcuno, magari con un form in una apposita sezione per fissare un appuntamento qualora si sia interessati all'acquisto.

2.2 Web Server e Tecnologie

Le web-application, come sono state esposte fin'ora, hanno bisogno di una base per funzionare. Un sito, statico o dinamico che sia, necessita di un server che gli permetta di essere accessibile da internet, o dall'intranet. Questo compito é svolto dai server web e dalle tecnologie illustrate in precedenza. Un server web non é altro che un'applicazione che rende accessibili dalla rete dei contenuti organizzati in pagine HTML. Un client effettua una chiamata al server con particolari parametri ed il server, in base a delle regole interne prestabilite, risponde alla chiamata inviando del codice HTML (o HTML corredato da codice JavaScript, CSS...). Quindi il client effettua un'ulteriore chiamata al server che completa la richiesta mandando immagini e quant'altro serva al client per completare il caricamento della pagina.

2.2.1 Web Servers tradizionali

IIS

IIS[24], Internet Information Server é il web server prorietario Microsoft ed ha una diffusione che si aggira attorno ad un quarto del mercato totale mondiale. Questo risultato, per nulla trascurabile é dovuto alla semplicitá di questo strumento che essendo integrato nel sistema operativo (ne é la sua estensione per internet) é di facile configurazione. Questo porta lo svantaggio di essere utilizzabile solo su un sistema operativo. L'ultima versione disponibile é la 6.0 e supporta le seguenti features:

Affidabilitá ,

- **Application Pools**, consiste nel dividere le web application in pool che vengono interrogate ad intervalli regolari con un ping. Quando una di esse termina inaspettatamente (non risponde al ping) non viene terminato l'intero IIS. Vengono invece isolate le richieste per quella web application che viene rilanciata ed a cui viene poi passata la coda insoluta. Per ogni web application inoltre viene monitorato il numero di crash in un dato periodo, se questo supera un numero prestabilito il servizio viene interrotto e viene notificata, ad ogni eventuale richiesta, una pagina che ne indica la non disponibilitá.

- **Process Orphaning**, questa feature permette di mantenere una web application nello stato in cui é terminata se questa non termina come previsto (i.e. crash dell'applicazione). Questo permette un debug dell'applicazione coadiuvato dall'effettivo stato in cui é terminata l'applicazione.

- **Uninterrupted TCP/IP connections**, si tratta di mantenere attiva e congelata una connessione dopo una terminazione inaspettata

dell'applicazione per ristabilirla al suo riavvio. Particolarmente utile per client che non implementano la logica di retry.

- **CPU throttling**, permette di dedicare piú tempo CPU ad un processo (web application che dir si voglia) piuttosto che ad un altro per favorirne uno che risponde ad esempio a piú chiamate rispetto ad un altro che stabilisce meno connessioni.

Gestibilitá ,

- **XML Configuration Data**, disponibile solo per Windows Server 2003, permette una totale configurazione del server tramite pagine XML. Introdotta nella versione 6.0 di IIS questa feature rende piú facile e veloce cambiare la configurazione al proprio server web senza dover usare script come nelle versioni precedenti. Inoltre questi cambiamenti alla configurazione posso essere effettuati mentre IIS é in funzione, senza dover riavviare l'applicazione.

- **Configuration Versioning Rollback, Import & Export**, viene mantenuto uno storico dei cambiamenti nella configurazione in maniera tale che possa essere effettuato un rollback per tornare ad una versione precedente del file e poter annullare i cambiamenti. Inoltre é possibile importare od esportare la configurazione per riportarla altrove e trovare il server configurato come lo si é lasciato.

- **GUI**, l'amministrazione del server é possibile anche usando una Graphical User Interface semplificando di molto l'uso di IIS anche a chi non sia esperto di web o computer.

- **Automated Patch**, IIS permette, come parte del sistema di patch di Windows Server 2003, di aggiornarsi automaticamente scegliendo tre modalitá di aggiornamento che ricalcano quelle implementate in Windows XP:

  - notificare quando una nuova patch é disponibile;
  - scaricare la patch e notificare la sua disponibilitá;
  - programmare l'installazione e scaricare la patch.

Scalabilitá ,

- **Process Isolation**, ció che si diceva sul Pool di processi vale anche come caratteristica di scalabilitá. Organizzare le web-application in gruppi permette di mantenere piú siti di applicazioni differenti senza confonderli tra loro.

- **TCP filtering**, per evitare attacchi contro porte aperte per sbaglio IIS permette di limitare quali debbano essere aperte.

Naturalmente IIS fornisce supporto ad ogni strumento proprietario Microsoft come ad esempio FrontPage ed é giá operativo per l'*IPv6*[25].

Architettura ,

Per questo prodotto sono state introdotte molte innovazioni soprattutto nel passaggio dalla release 5.0 alla 6.0, l'attuale che si sta illustrando. L'architettura non é da meno, l'introduzione dell'application pools l'ha significativamente rivoluzionata nella parte relativa appunto alle web-application.

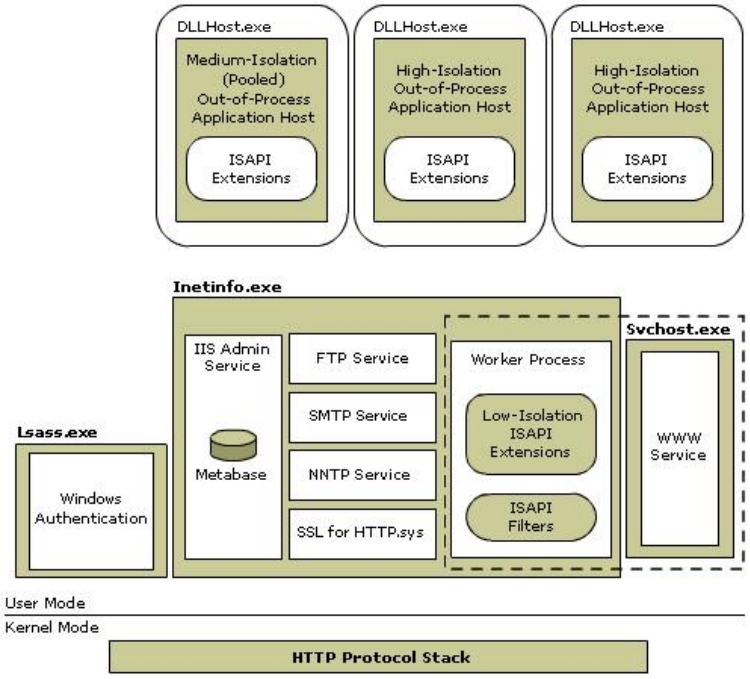

Figura 2.2: La struttura del server web IIS.[27]

Come si puó notare dalla figura 2.2 nonostante la divisione in moduli questi sono di competenze vaste e non sono di piccole dimensioni. Questo comporta una poca modularitá e la necessitá di rivedere tutta quanta l'applicazione per introdurre dei cambiamenti, anche di poco conto.

Apache

Sviluppato dalla Apache Software Foundation[28], Apache deve il suo nome al sistema di patch con cui veniva aggiornato il server NCSA HTTPd[29]. Quando si decise di sviluppare il tutto per lanciare un nuovo server che

conservasse lo scheletro di quello sul quale si era lavorato fino ad allora ci si accorse che la versione era null'altro che un'ulteriore patch. Il nome non poteva che essere *a patchy server*. Il pregio di questo progetto che lo porta ad essere la soluzione scelta da piú del 65% dei siti mondiali é la sua semplicitá per piccole realtá con poche pretese e la sua completezza in situazioni piú particolari, dove si ricercano prestazioni ed affidabilitá unite alla possibilitá d'accesso e manipolazione della quasi totalitá dei particolari. Se si unisce questo risultato ad un prodotto opensource capirne l'enorme successo non é difficile. Per necessitá di chiarezza le features di Apache saranno illustrate di pari passo con l'architettura in quanto, a differenza di IIS illustrato nel capitolo precedente, Apache non é un insieme di servizi piú o meno utili. É piuttosto una serie di livelli ognuno con le proprie caratteristiche.

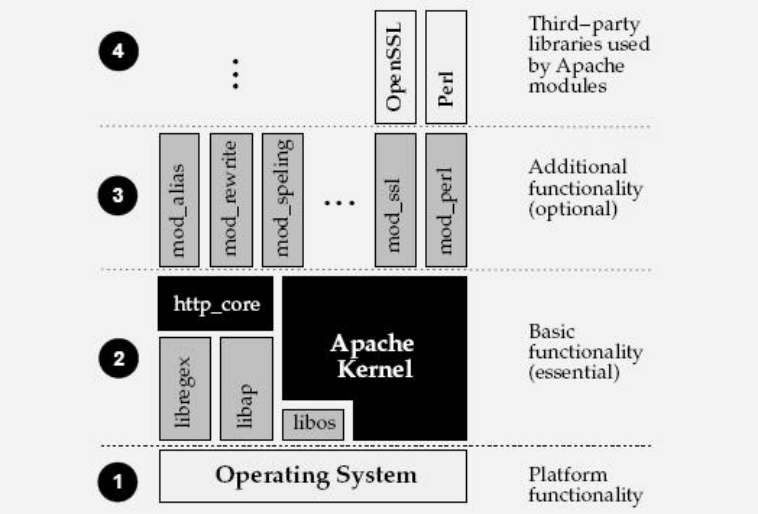

Figura 2.3: La struttura del server web Apache.[30]

Operating System , Il primo livello riguarda il sistema operativo. Apache é stato scritto prevalentemente per sistemi UNIX o cosiddetti UNIX-like, ma ció non toglie la possibilitá di usarlo in Micrsoft Windows o OS/2. Questo era precedentemente possibile grazie all'emulazione thread delle librerie *POSIX* [26]. Successivamente con l'uscita della versione Apache 2 gli strati di MPMs[7] e ARP[8] sono stati implementati con le API[9] native del sistema operativo supportato.

[7] Multi-Processing Modules
[8] Apache Portable Runtime
[9] Advanced Programmable Interface

Core Level & Apache Kernel , É il server web vero e proprio, contiene il kernel dell'applicazione che assieme al modulo *http_core* implementa le funzionalitá HTTP base e le API per *module layer*. Il livello di funzionalitá base contiene anche la libreria generica *libap*, una piccola libreria di astrazione del sistema operativo (*libos*) e la libreria per le espressioni regolari *libregex* che dalla versione 2.0 é implementata con la sintassi per le espressioni regolari di Perl[39] 5 (*PCRE*, Perl Compatible Regular Expression Library).

Module Layer , É un livello di funzionalitá facoltative che rappresenta la vera forza di Apache e che lo contraddistingue dagli altri web server presenti sul mercato. Nessun modulo é necessario per permettere ad Apache di svolgere le funzionalitá di base, cioé servire pagine statiche da un'area predefinita. In veritá é necessario almeno *mod_mime*[10].

Third-Party Libraries , Questo livello é sostanzialmente dipendente dai moduli inseriti nel Module Layer. In questa parte infatti vengono caricate le librerie necessarie per le interfacce Apache come *mod_perl*[11] o *mod_ssl*[12].

I Moduli , Questa sezione sará dedicata alle maggiori estensioni di Apache, quelle che assolvono il compito di aggiungere funzionalitá al web server base.

- **mod_action**, serve per catturare le richieste su file di specifico contenuto *mime* o su particolari richieste HTTP. Serve per creare contenuto dinamico nelle pagine HTML.
- **mod_negotiation**, é usato per servire al meglio una risorsa quando questa é disponibile in diverse rappresentazioni basando la scelta sulle preferenze espresse dal client. Anche in presenza di preferenze mancanti, attraverso un'euristica permette di ipotizzare la migliore configurazione per il client.
- **mod_unique_id**, permette la creazione di una stringa casuale di cui é garantita l'unicitá addirittura su un cluster di server, naturalmente se questi sono configurati propriamente.
- **mod_dir**, serve per il rendirizzamento d'una richiesta su una *URL* incompleta o scritta male, per esempio in mancanza del carattere '/',

[10] Multipurpose Internet Mail Extension é uno standard di internet che permette la classificazione del contenuto di un file dipendentemente dalla sua estensione e quindi ne definisce i meccanismi atti a spedirlo. Queste direttive basate sullo standard possono ugualmente essere sovrascritte fornendo al core-layer di Apache delle direttive quali *ForceType*, *SetHandler*, *SetInputFilter* e *SetOutputFilter* che una volta dichiarate e configurate propriamente acquisiscono prioritá sulla base fornita dal modulo mod_mime.

[11] modulo necessario, ad esempio, per interfacciare Mason, che gestisce direttive Perl nelle pagine HTML.

[12] modulo per il Socket Secure Layer.

o del riferimento esplicito alla pagina di default (index.html) che viene servita automaticamente; é comune a praticamente tutti i web server presenti sul mercato.

- **mod_cgi**, é l'interfaccia piú classica per generare contenuto dinamico all'interno di una pagina.

- **mod_mime**, fornisce la configurazione degli *Handler* per i vari file di diverso contenuto. Si basa principalmente sulle estensioni dei file che passano dal filtro, se per quella particolare estensione c'é un handler assegnato allora questo solo viene chiamato per tradurre la pagina in HTML o per sapere cosa farne.

- **mod_mime_magic**, rispetto al precedente modulo non si basa sull'estensione del file, ma sui primi byte del suo contenuto, cercando di indovinare il *content-type* del file per chiamare l'handler giusto.

- **mod_expires**, questo modulo di preoccupa di passare al vaglia il campo *Expires* dell'header HTTP e, basandosi sulla data dell'ultima richiesta del client o del documento da servire[13], informare il client ed i proxy nel mezzo della non validitá della pagina in maniera tale che si possa effettuare un caching consistente.

- **mod_asis**, permette di definire tipi di file da mandare senza aggiungere gli header HTTP. É usato per mandare qualsiasi file dal server, o anche per le redirezioni, senza dover ricorrere a direttive CGI.

- **mod_perl**, questo modulo consente di integrare il linguaggio Perl all'interno di Apache permettendo di scrivere estensioni nel linguaggio di scripting piuttosto che usare le CGI. Questo porta un vantaggio non indifferente che deve essere ricercato nel fatto che perl, essendo un linguaggio di script, é tradotto in *byte-code*[14] e dunque interpretato, non compilato. Questo permette di velocizzare non poco il tempo di risposta.

- **mod_ssl**, é il collante tra il toolkit OpenSSL ed il web server Apache, fornisce la crittografia necessaria a poter instanziare connessioni HTTPS sul protocollo TCP/IP.

[13] si puó scegliere una delle due alternative e programmarne il comportamento

[14] il byte-code é un linguaggio a metá tra quelli di programmazione ed il linguaggio macchina. Si chiama cosí perché generalmente ogni operazione é tradotta in un singolo byte, poi trasformato effettivamente in linguaggio macchina dall'interprete, conosciuto meglio come *macchina virtuale*. Tutto questo permette di astrarre dall'hardware e di dover fare una traduzione piú 'leggera' ad ogni esecuzione, alleggerendo il carico CPU e velocizzando tutto il processo. Anche Java[31], nonostante non sia un linguaggio di scripting é comunque interpretato.

Mason

Mason[41] é un software che permette di creare siti con supporto per pagine dal contenuto dinamico tramite Perl[15]. É un'applicazione che funziona con l'estensione *mod_perl* di Apache interfacciandosi ad esso con il modulo Perl HTML::Mason::ApacheHandler. Funziona con il principio delle scatole cinesi. Nelle scatole cinesi la scatola che si vede quando sono montate non é che la piú grande che contiene le altre. Aprendole una alla volta si arriverá ad avere la piú piccola che é l'elemento minimo che si puó ottenere. Cosí funziona Mason fornendo un'archiettura di tipo *object-like* in cui basandosi sulla posizione sul disco di una risorsa richiesta questa seguirá un iter prima di essere servita. Quando a Mason si richiede una pagina che non contiene questa direttiva

```
<%flags>
    inherit=>'DaQualche/Parte/parent'
</%flags>
```

Mason si preoccupa di richiamare il file *autohandler* nella directory dov'é presente la pagina che si sta considerando. Se quel file non c'é nella directory allora estende la ricerca alla directory padre e cosí via fino alla radice[16] fino a quando non ne trova uno. Se non dovesse trovarne allora richiama un altro file, il *dhandler*.

```
http://mioServer/mieApplicazioni/Applicazione1/index.html
```

questa richiesta potrebbe seguire questo iter:

```
/mieApp/App1/autohandler   <= non trovato
/mieApp/autohandler        <= trovato, ricerca finita
/autohandler
/dhandler
```

In questa maniera, dividendo il sito da servire con questa logica, si possono annidare i componenti per richiamarli poi a piacere. Ad esempio l'autohandler di livello piú basso puó essere un componente che comprime la pagina da servire prima di mandarla al client. L'autohandler di secondo livello puó essere ad esempio quello che invece inserisce nella pagina i tag principali dell'HTML. Separato rispetto alla compressione dei dati perché potrebbe essere necessario anche non mandare esclusivamente pagine HTML. Se invece la flag *inherit* é dichiarata verrá chiamato il componente passato come parametro. La sintassi di Mason é piuttosto banale.

[15] trattato in seguito in una sezione apposita

[16] naturalmente per radice si intende la directory radice di Mason, non quella di sistema

- % indica una singola direttiva Mason;
- <% inica l'inizio di un blocco di direttive Mason che terminerá cosí %>;
- <%init> indica un insieme di direttive che devono essere eseguite come prima cosa ad ogni chiamata sul componente (ad esempio l'inserimento dei tag principali HTML) </%init>;
- <%shared> indica un insieme di direttive che devono essere eseguite come prima cosa ed essere disponibili in differenti scopes (ad esempio l'apertura di una connessione con il database) </%shared>;
- <%once> indica un insieme di direttive che devono essere eseguite una sola volta al primo caricamento (ad esempio dichiarazione di variabili globali molto grandi o dichiarazione di subroutine ed in generale prime inizializzazioni) </%once>;
- <%filter> indica un insieme di direttive che devono essere eseguite come ultima cosa prima di mandare la pagina (ad esempio comprimerla con ZIP per i browser che lo supportano) </%filter>;
- <%perl> indica un insieme di direttive che devono essere eseguite nella posizione in cui si trova il blocco </%perl>.

Naturalmente tutti i blocchi illustrati fin'ora contengono prevalentemente codice Perl con direttive Mason. Il codice HTML al di fuori di direttive Mason é lasciato com'é. Infatti Mason non traduce tutto il file. Prende solo i blocchi che lo riguardano, di cui quelli sopra sono solo una piccola parte, ed esegue direttive e codice Perl al loro interno producendo HTML come output.

2.2.2 Tecnologie applicate ai Web Servers tradizionali

JavaScript

É un linguaggio di scripting orientato agli oggetti sviluppato da Brendan Eich della Netscape Corporation e standardizzato tra il 1996 ed il 1999 dalla ECMA[32] con il nome di ECMAScript. Ora, lo standard ECMA-262 Edition 3 della ECMA trova corrispondenza con JavaScript 1.5 che é anche uno standard ISO[33]. Il nome JavaScript é stato assegnato per la vicinanza della sintassi a quella di Java, ma il linguaggio é molto piú simile al linguaggio C[17]. É solitamente usato all'interno di pagine HTML e dialoga con il browser tramite l'interfaccia fornita dal DOM. Ció che é piú interessante di questo linguaggio é che ha la possibilitá di intercettare gli eventi

[17] probabilmente il nome é stato scelto per il successo che Java della Sun[36] stava riscuotendo in quel periodo

del browser tramite l'interfaccia DOM. Questo permette, come mostrato nella parte dedicata al Document Object Model di modificare le pagine senza doverle ricaricare. Il rovescio della medaglia é che le implementazioni di JavaScript all'interno dei browser web non sono standard e questo crea non pochi problemi a chi voglia sviluppare applicazioni web con un buon grado di supportabilitá. Mozilla, ad esempio, sfrutta un'implementazione non standard di JavaScript, ma che gli é comunque molto vicina. Microsoft invece ne sfrutta un'altra molto piú scostata dallo standard rispetto alla precedente. Inoltre le difficoltá non terminano qui. C'é differenza infatti anche nelle interfacce DOM fornite dai browser ed anche questa é una difficoltá che si deve affrontare quando si vuole scrivere del codice supportato dal maggior numero di browser. Quello che si fa, in generale, é di scrivere un modulo, all'interno dell'applicazione, che rileva tramite interrogazioni particolari[18], che tipo di browser stia usando il client. Come si diceva la sintassi di JavaScript é molto piú vicina a quella del C rispetto che a quella di Java. Le variabili sono tipate dinamicamente a run-time e lo scope é limitato all'ambito di dichiarazione (in generale una variabile dichiarata all'interno di una funzione sará visibile solo in quella funzione, mentre una dichiarata all'esterno avrá visibilitá globale). JavaScript supporta gli oggetti, ma non le classi. Questo gli consente di realizzare dei prototipi aggiungendo le funzioni o gli attributi desiderati all'oggetto.

```
function Esempio(Attributo1, Attributo2, Attributo3)
{
    this.Attributo1 = Attributo1;
    this.Attributo2 = Attributo2;
    this.Attributo3 = Attributo3;

    this.StampaEsempio = function()
    {
        document.write(this.Attributo1);
        document.write(this.Attributo2);
        document.write(this.Attributo3);
    }
}
```

Questo pezzo di codice permette di definire un oggetto a cui vengono aggiunti tre attributi passati come parametro nella dichiarazione della funzione. A 'this', cioé all'oggetto Esempio, vengono quindi aggiunti i tre parametri passati come argomenti nella creazione dell'oggetto, nel suo istanziamento. La funzione di stampa prende il riferimento dell'oggetto su

[18] in alcuni casi puó addirittura essere necessario fare delle chiamate a cui solo un particolare browser puó rispondere per capire effettivamente con chi si sta dialogando

cui é definita e ne stampa gli attributi sul documento con la stessa sintassi di Java per le classi e di C per le struct.

```
var esempio = new Esempio(‘‘attr1’’, ‘‘attr2’’, ‘‘attr3’’);

esempio.StampaEsempio();
```

Quest'ultimo stralcio di codice permette di istanziare l'oggetto JavaScript con gli attributi passati come parametri e di stamparne poi sulla pagina il contenuto con la chiamata alla funzione *StampaEsempio*. In realtá gli oggetti sono immessi in memoria sotto forma di array associativi, hashtables, e quindi sono possibili due tipi di notazioni per l'accesso ad un attributo.

- la notazione classica della programmazione ad oggetti

  ```
  alert(esempio.Attributo1);
  ```

- la notazione classica dell'array associativo

  ```
  alert(esempio[‘‘Attributo1’’]);
  ```

Oltre agli oggetti JavaScript dichiarabili dal programmatore questo linguaggio ne fornisce alcuni giá implementati con funzioni molto simili a quelle che si possono trovare nelle librerie delle classi di Java

- **Array**,

  ```
  var nomi = new Array(2);
  var nomi = new Array("nome1","nome2");
  ```

  con i piú importanti attributi e funzioni correlati

  - Array.length, attributo che fornisce il numero degli elementi contenuto in un array;
  - Array.concat(), concatena la stringa su cui é chiamato con quella passata per parametro;
  - Array.pop(), restituisce l'ultimo elemento dell'array rimuovendolo;
  - Array.push(), aggiunge uno o piú elementi all'array restituendone la nuova lunghezza;
  - Array.reverse(), inverte l'ordine degli elementi dell'array su cui é chiamato;

- **Boolean**,

```
var vero = new Boolean();
var vero = new Boolean(false);
var vero = new Boolean(0);
var vero = new Boolean(null);
```

creano un oggetto Boolean allo stato *false*,

```
var vero = new Boolean(true);
//come anche una qualsiasi stringa non vuota...
var vero = new Boolean(‘‘false’’);
```

mentre questi ultimi creano un oggetto Boolean di valore *true*;

- **Date**,

  ```
  var nowDate = new Date();
  ```

  usato per lavorare sulle date e sui tempi, ha come funzioni piú importanti le corrispettive della classe *Date* in Java;

- **Function**, serve per trattare le funzioni come oggetti a se stanti, con parametri e valori e non come pure referenze ad una funzione;

- **Math**, serve per le operazioni matematiche, ha come attributi le costanti piú importanti

  - Math.E, la base del logaritmo naturale;
  - Math.LN2, logaritmo naturale di 2;
  - Math.PI, il p greco;

  anche qui i metodi trovano sostanziale corrispondenza con l'omonima classe Java e la maggior parte delle librerie matematiche degli altri linguaggi di programmazione;

- **Number**, é un oggetto che funge da wrapper per i numeri;

- **Object**, é, come in Java, l'oggetto radice con attributi e funzioni comuni a tutti i suoi discendenti, si vuole porre l'attenzione soprattutto sugli attributi;

  - Object.constructor, specifica una funzione per creare un attributo di un oggetto ed é ereditato da tutti gli oggetti di un particolare prototipo;
  - Object.prototype, aggiunge attributi e funzioni al prototipo dell'oggetto su cui é chiamato;

```
MioOggetto.prototype.attributo_mancante = null;

istanza_di_MioOggetto.attributo_mancante = ‘‘valore’’;
```

questa maniera di agire é molto importante quando si vuole aggiungere alla struttura di un oggetto un particolare attributo sapendo di volerne tener traccia anche su oggetti giá istanziati ed usati correntemente;

- **RegExp**, come l'omonima classe Java serve per le espressioni regolari;
- **String**,

```
var stringa = new String(2);
var stringa = new String("stringa");
```

anche su questo oggetto c'é la corrispondenza con l'omonimo in Java e ne implementa i maggiori metodi

  - String.length, attributo che fornisce il numero di caratteri all'interno della stringa;
  - String.concat(), concatena la stringa su cui é chiamato con quelle passate per parametro;
  - String.bold(), stampa la stringa come se fosse contenuta nei tag dell'HTML, si ha anche String.italics() e altre funzioni dello stesso tipo;
  - Array.link(), aggiunge un link nel documento, dalla stringa all'URL passato come parametro alla funzione;

AJAX

L'Asynchronous JavaScript and XML é una tecnica per creare web application interattive. Permette di scambiare piccoli pacchetti con il server, non solo a seguito di una esplicita richiesta del client, anche per richiedere piccole parti della pagina dinamicamente. Questo consente di non dover ricaricare tutta la pagina a seguito di un cambiamento anche minimale da parte del client. In realtá AJAX non é una tecnologia, ma un insieme di tecnologie che combinate assieme permettono di produrre un cambiamento nelle applicazioni web non indifferente. Agisce tramite JavaScript che dialoga con il Dom scambiando dati con il server attraverso XML come linguaggio di markup. Ora, il fatto che sia un insieme di tecnologie ci permette di pensare che, effettivamente, sia possibile adattare l'idea alle proprie esigenze. L'evoluzione in campo informatico spinge spesso la collettivitá ad usare soluzioni che non calzano esattamente le esigenze di tutte le tipologie di

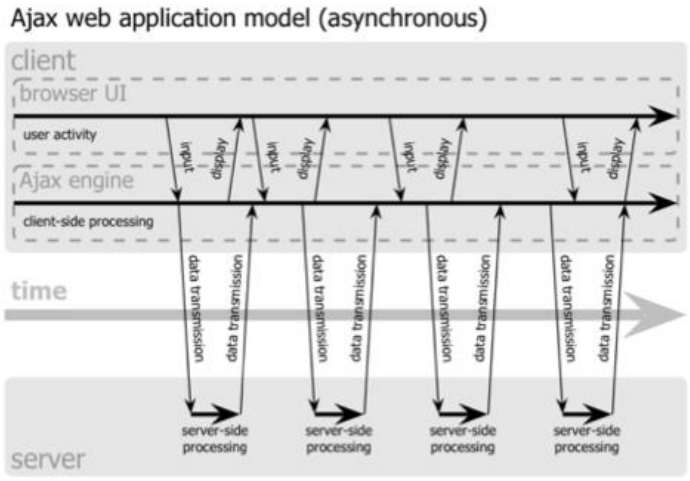

Figura 2.4: La maniera di procedere della tecnologia Ajax.

utenti. Ad esempio Java é un linguaggio che puó andare bene per le grandi aziende, mentre non é assolutamente indicato per soluzioni artigianali.

Allo stesso modo XML, oramai diffusissimo, non é assolutamente una soluzione ottimale per tutte le situazioni, come alcuni sembrano pensare. É anzi un peso allorché i due soggetti che devono dialogare usano lo stesso protocollo. La forza di XML risiede infatti nella versatilitá e nella possibilitá di poter descrivere dei dati con un ingente quantitá di informazioni, in maniera tale che chi li riceva abbia una visione globale e perfetta della struttura e di come utilizzare quei dati. L'esempio che puó essere suggerito é quello di due aziende diverse, magari di nazioni diverse, che devono condividere i loro database centrali. É ragionevolmente ovvio che se una delle due aziende si trova in un paese dove non si usa l'alfabeto latino, ad esempio il Giappone, la traduzione del database potrebbe richiedere moltissimo tempo. Inoltre non solo la lingua, ma anche l'organizzazione del database in tabelle potrebbe rappresentare un problema. Se lo studio rispettivo dei database ha portato a soluzioni sensibilmente diverse una traduzione dall'uno all'altro rappresenterebbe un problema non da poco. In questo caso viene in soccorso l'XML che astrae le singole organizzazioni e ne produce una terza di comune accordo. In questo modo lo scambio di dati tra i due database diventa un problema piú semplice. Lo svantaggio di XML risiede nel volume di dati scambiati. In una applicazione client-server entrambi utilizzano la stessa organizzazione dei dati e quindi non c'é diversitá nella struttura. XML diventa quindi un peso perché per strutture complesse il volume dei dati utili diventa di poco conto rispetto a quelli che ne descrivono l'organiz-

zazione. Di conseguenza si avrá un overhead sensibile, senza togliere anche il tempo per fare il parsing dei dati. Questo porta ad un sovraccarico della rete e del client stesso, perché se é vero che per un server leggere file di 1K o 1M non fa molta differenza non si puó dire lo stesso per un client, per non parlare della banda maggiore da utilizzare ed il tempo di parsing che su alcuni client non puó essere ignorato. Tutto questo é inaccettabile e c'é necessitá di trovare un'altra soluzione. Per gli stessi motivi non si puó prendere in considerazione nemmeno *SOAP*[34] appunto perché derivato da XML. Nell'esperienza svolta presso l'azienda Leader.IT e dal lavoro di tesi di Fedrizzi Tarcisio[48] si é giunti alla conclusione che un protocollo di interscambio dati comodo, flessibile e leggero adatto all'applicazione che si vuole studiare é JSON[7].

JSON , É un protocollo di interscambio dati molto leggero, l'acronimo sta per JavaScript Object Notation e vi si evince che é un sottoinsieme del linguaggio JavaScript. Permette lo scambio di strutture dati verso JavaScript, ma la sua forza sta appunto nel basarsi completamente sul testo per descriverle. Questo infatti permette facilmente di scambiare dati con tutti i linguaggi esistenti ed é infatti supportato da molteplici di essi quali C, Java, JavaScript per l'appunto, Perl di nostro interesse, ma anche Python o PHP. I tipi di dato supportati da JSON sono i seguenti:

- *boolean* attributi che possono assumere valore 'false' o 'true';
- *string* attributi costituiti da 0 o piú caratteri Unicode con carattere d'escape per i caratteri speciali (il carattere d'escape serve per far considerare all'interprete un carattere come parte di una stringa nonostante sia appartenente alla sintassi del linguaggio, in pratica si inserisce il carattere di escape '\' seguito dal carattere speciale che altrimenti sarebbe interpretato e non considerato parte della stringa);
- *array* sono costrutti che rappresentano un'ordinata sequenza di valori separati da una virgola e racchiusi in parentesi quadre, i valori possono essere indifferentemente boolean, stringhe od oggetti fino addirittura anche ad altri array, senza limite di annidamenti;
- *object* collezioni di coppie chiave/valore separati da virgole e racchiuse in parentesi graffe, i valori possono essere boolean, stringhe, array od altri oggetti, senza limite di annidamenti.

Un esempio di oggetto JSON é fornito di seguito:

```
{
    "nome": "Marco",
    "cognome": "Albano",
```

```
    "recapito": {
        "comune": "Civezzano",
        "provincia": "Trento",
        "indirizzo": "via Argentario"
    },
    "numeri_telefono": [
        "123 4567890",
        "098 7654321"
    ]
}
```

ricevendo un oggetto JSON cosí strutturato in JavaScript si puó interpretarlo con la direttiva *eval* che lo rimette nella forma nella quale era stato serializzato e spedito.

```
var person = eval("(" + Marco_JSON + ")");
```

Per accedere ai vari campi dell'oggetto ricomposto si adotta la notazione propria del linguaggio con cui si é interpretata la struttura dati, in JavaScript ad esempio i vari valori sono accessibili in questa maniera:
per il nome

```
person.nome;
```

per il cognome

```
person.cognome;
```

per il recapito ed i vari elementi di cui é composto

```
var rec = person.recapito;

rec.comune;

rec.provincia;

rec.indirizzo;
```

e per i numeri di telefono, rispettivamente per il primo ed il secondo

```
person.numeri_telefono[0];

person.numeri_telefono[1];
```

Naturalmente la sintassi varia da linguaggio a linguaggio, ma il risultato é lo stesso. Dopo una valutazione di un oggetto JSON si ricostruisce la struttura serializzata in partenza. Per questo motivo é da citare il fattore

sicurezza. Essendo la sintassi di JSON basata interamente su testo Unicode l'operazione di *eval* descritta sopra non é altro che un parsing della struttura JSON con la successiva istanziazione del contenuto. É quindi consigliabile, quando si lavora con oggetti JSON che si ritengono non sicuri, per provenienza o altre carattestiche, procedere ad un parsing manuale evitando di incorrere in eventuali problemi ricorrendo a quello automatico. Qui risiede uno degli svantaggi di JSON. XML é infatti dotato di parser programmabili che svolgono il lavoro necessitando in input solo di un set di regole e di un file XML. Cosa invece completamente mancante in JSON.

JSP

Le JSP[35] furono presentate nel 1998 in occasione di una manifestazione organizzata dalla Sun Microsystem[36] in America. Non riscossero molto successo per mancanza di organizzazione dell'azienda che non aveva preparato esempi pratici di applicazione. Piú tardi, quando si comprese il loro effettivo valore, furono massicciamente impiegate per la costruzione di pagine web dinamiche per vari motivi. Sono scritte in Java e ne ereditavano quindi il successo, separano nettamente nelle pagine web l'organizzazione dei contenuti dai dati stessi e su server aziendali sono molto piú veloci ad esempio delle ASP[37], per costruzione. Le JSP funzionano in definitiva su un principio molto semplice, costruire una pagina HTML con l'uso di direttive Java che vengono interpretate e tradotte a loro volta in HTML. Per questo il sorgente di una pagina JSP non interpretata é una servlet[19], un misto cioé di codice HTML e tag $<$ %% $>$ all'interno delle quali si trova codice Java.

```
<% FilmDisponibile[] list = Interfaccia.getFilmDisponibili();
for(int i = 0; i < list.length; i++){%>

    <tr>
        <TD><%out.println(list[i].getTitolo());%></TD>
        <TD><%out.println(list[i].getProtagonista());%></TD>
        <TD><%out.println(list[i].getRegista());%></TD>
        <TD><%out.println(list[i].getData()+" "+list[i].getOra());%></TD>
        <TD><%out.println(list[i].getNSala());%></TD>
        <TD><%out.println(list[i].getPostiDisponibili());%></TD>
    </tr>
<% }%>
```

[19]le JSP vere e proprie hanno una sintassi diversa dalle servlet e vengono usate per semplificare il codice in quanto sono piú simili all'HTML di quanto lo siano a Java e questo permette al programmatore di scrivere codice piú complicato con piú disinvoltura e non obbligandolo a concentrarsi eccessivamente sulla leggibilitá di ció che scrive. Sono solo nominate e non trattate perché il codice JSP é interpretato da Tomcat (nel caso di JSP ci si appoggia maggiormente all'estensione di Apache) e tradotto in servlet prima di essere interpretato e convertito in HTML

Riportato nell'esempio c'é un'ipotetica richiesta da parte di una servlet di conoscere la lista dei film disponibili in un cinema. Come si puó vedere comincia con la tag < % che indica che le righe seguenti sono codice da tradurre. Viene infatti chiesta la lista dei film disponibili e per ognuno, come si puó vedere dal codice all'interno del ciclo *for*, si crea una riga di una tabella e nelle varie colonne vengono inseriti, sempre con richieste all'oggetto lista, i vari campi del film che si sta trattando in quel momento. Nel caso in cui i film siano ad esempio due, la pagina HTML avrebbe il seguente output

```
<tr>
    <TD>$(TITOLO PRIMO FILM)</TD>
    <TD>$(PROTAGONISTA PRIMO FILM)</TD>
    <TD>$(REGISTA PRIMO FILM)</TD>
    <TD>$(DATA PROIEZIONE PRIMO FILM) $(ORA PROIEZIONE PRIMO FILM)</TD>
    <TD>$(NUMERO SALA PROIEZIONE PRIMO FILM)</TD>
    <TD>$(POSTI DISPONIBILI PRIMO FILM)</TD>
</tr>
<tr>
    <TD>$(TITOLO SECONDO FILM)</TD>
    <TD>$(PROTAGONISTA SECONDO FILM)</TD>
    <TD>$(REGISTA SECONDO FILM)</TD>
    <TD>$(DATA PROIEZIONE SECONDO FILM) $(ORA PROIEZIONE SECONDO FILM)</TD>
    <TD>$(NUMERO SALA PROIEZIONE SECONDO FILM)</TD>
    <TD>$(POSTI DISPONIBILI SECONDO FILM)</TD>
</tr>
```

Ovviamente le JSP forniscono anche degli oggetti particolari per servire al meglio il programmatore dandogli accesso a tutte le variabili che una richiesta HTTP istanzia. Si ha infatti la possiblitá di mandare cookies[20] al client, di farsi riportare l'oggetto *request* al quale si possono richiedere informazioni essenziali. Il numero di sessione della connessione ad esempio che permette effettivamente di personalizzare il contenuto della pagina in base all'identitá (da verificare con un login) dell'utente connesso.

```
if(sessionID == null || sessionID.equals("")){
    RequestDispatcher rd = request.getRequestDispatcher("login.jsp");
    rd.forward(request, response);
}
```

In questo esempio si vede come sia possibile rimandare un utente alla pagina di login se questo non si é autenticato e non gli é stato quindi assegnato un *sessionID*. Una procedura di questo tipo previene la possibilitá di

[20] i coockie (biscottini) sono dei file di testa che riportano alcune informazioni utili per mantenere la sessione o qualsiasi altra cosa il server voglia che sia piú o meno persistente sul client

caricare pagine diverse da quella di login senza prima essersi autenticati. Il processo mediante il quale una pagina viene tradotta in base alle direttive Java in essa contenute in codice HTML avviene una sola volta nel server che conserverá la servlet in memoria continuando a servire quella copia alle richieste successive. Questo fino a quando Tomcat non si accorge che il codice sorgente della servlet é cambiato e quindi procederá alla ricompilazione per poi sostituire la copia che si trova in memoria. Analizzando questo procedimento descritto in termini di massima si puó facilmente capire che la forza delle JSP risiede nella velocitá di servizio permessa dal prefetch della servlet tradotta. Quindi sará pesante in fase di primo caricamento, ma le successive richieste verranno evase con molta facilitá. Permette inoltre di interfacciare le applicazioni con database di diverso tipo (MySQL, PostgresSQL ad esempio). Tramite l'oggetto *Connection* di *java.sql* ed i driver appropriati si puó aprire una connessione verso un database supportato dal driver ed eseguire query senza alcuna limitazione.

```
public DatabaseManagement() {
    if (conn == null) {
        try {
            Class.forName("org.postgresql.Driver");
            System.out.println("Tento la connessione al database...");
            conn = DriverManager.getConnection("jdbc:postgresql://");
            System.out.println("Connessione riuscita: ");
        }
        catch (SQLException sqlE) {
            System.out.print("Connessione non riuscita, errore: ");
            sqlE.printStackTrace();
        }
        catch (ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

Complessivamente risulta un sistema *server-side* assolutamente affidabile, veloce, relativamente semplice (Java é un linguaggio dai concetti base molto facili) e con la peculiare caratteristica di non appesantire il carico di lavoro del client incentrando tutto sul server. Proprio per questo motivo é inadatto a ció che si sta studiando. La necessitá di usare AJAX lato client per avere piú flessibilitá e portare parte della logica di programmazione sul browser impone di utilizzare lato server una tecnologia diversa dalle JSP che, come se non bastasse, sono progettate per servire richieste e risposte. Non adottano quindi la logica ad eventi illustrata nei capitoli precedenti e sono quindi inadatte allo scopo di questa tesi.

Perl

Perl é un linguaggio interpretato che nasce dal lavoro di Larry Wall che nel 1987 ne pubblicó la versione 1.0. Nacque come linguaggio per poter amministrare alcuni server, ma via via che si andava avanti lo si usó anche per scrivere un tool per la reportistica. Nacque dunque *Awk* che peró mancava in alcune caratteristiche a Wall necessarie e che lo spinse a creare un linguaggio nuovo chiamato appunto Perl[21]. Fin dal principio Perl ricevette molte adesioni per la sua velocitá e potenza e questo portó alla creazione di una comunitá che conta oramai tantissime persone. Una delle features che ha reso maggiormente famoso Perl riguarda l'implementazione delle espressioni regolari tanto che si é trasformato tutto in uno *standard de-facto*. La visione che si sta dando di questo linguaggio potrebbe trarre in inganno. Il fatto che Perl sia indicato come linguaggio di scripting per amministrazione di sistema[22] non deve essere considerato come limitativo. Infatti un'altra caratteristica peculiare di Perl é la disponibilitá online di numerosissime librerie tutte organizzate in quello che si chiama CPAN[40]. La forza di questo sistema é la possibilitá di scaricare le librerie che servono senza preoccuparsi (in generale) della loro installazione e della risoluzione delle dipendenze. CPAN si occupa, difatti, di tutto automaticamente ed una volta scaricata la libreria che interessava basta includerla nello script in cui si vuole usarla. Per rendere l'idea di cosa sia CPAN basti dare uno sguardo ai dati riportati nella homepage. In data odierna, 04/01/2007, vengono segnalati 3441 MB di librerie distribuiti in 280 mirrors, per un totale di 11022 moduli complessivi. L'approccio di Perl é sensibilmente diverso dalla maggior parte degli altri linguaggi. Si evidenziano fin da subito i tipi di dato che in Perl sono scalare, vettore ed hash(un vettore con indice alfanumerico). Questi tre tipi si differenziano tra loro dall'uso di caratteri particolari prima della variabile.

Tipo Variabile	Prefisso
scalare	$
vettore	@
hash	%

Tabella 2.1: prefissi per la tipizzazione di variabili Perl

Altri tipi come le stringhe vengono gestiti in maniera automatica dato che Perl é un linguaggio tipato dinamicamente, ad esempio $''3'' + ''3''$ in Java é una concatenazione di stringhe, mentre in Perl é la somma tra i due numeri.

[21] Linguaggio Pratico per l'Estrazione e la Reportistica

[22] permette infatti, con la sua implementazione delle regex, di svolgere automaticamente anche lavori piuttosto complessi in maniera non elementare, ma sicuramente efficace e veloce

Espressione	Risultato Java	Risultato Perl
″3″ + ″3″	concatenazione di stringhe	somma tra interi
″3″ . ″3″	//	concatenazione di stringhe
3 + 3	somma tra interi	somma tra interi

Tabella 2.2: differenze tra risultati di espressioni Perl e Java

In Perl esistono anche gli oggetti. Nonostante sia improprio chiamarli in tal modo perché non esistono, come in Java o C++, dei metodi atti alla loro creazione, ma si utilizza uno stratagemma per istanziarne uno. In pratica si crea una hash e poi la si *benedice* (funzione bless()) indicandogli di che tipo deve essere. In pratica si crea un qualcosa e si passa il suo reference alla funzione bless() indicandogli anche il nome del package con cui deve essere "benedetto". In termini di codice si crea il package

```
#---------Umano.pm---------#
package Umano;

1;
```

e lo si richiama nel codice dove lo si vuole utilizzare con la direttiva *use Umano.pm;*. Il costruttore che deve "benedire" l'oggetto puó essere incluso nel package Umano.pm ed essere chiamato come si preferisce. In generale peró si usa chiamarlo *new*.

```
sub new{

    my $class = shift;
    my ($nome, $cognome, $sesso, $eta) = @_;

    $this = {};

    $this->{'nome'} = $nome;
    $this->{'cognome'} = $cognome;
    $this->{'sesso'} = $sesso;
    $this->{'eta'} = $eta;

    bless($this, $class);

    return $this;
}
```

Questo stralcio di codice é abbastanza intuitivo, ma ci preoccuperemo di spiegarne i punti che potrebbero risultare non chiari, come ad esempio le righe

```
my $class = shift;
my ($nome, $cognome, $sesso, $eta) = @_;
```

La prima prende la chiamata della funzione e ne estrapola il primo parametro assegnandolo alla variabile $class. La seconda riga prende ció che rimane della chiamata di funzione, che é un array, e inserisce il valore degli elementi al suo interno dentro le variabili ordinate passate come parametro all'assegnamento.

```
bless($this, $class);

return $this;
```

Le ultime due righe invece rappresentano la "benedizione" e la chiusura della funzione. Viene infatti chiamata la funzione bless() che assegna alla variabile $this il tipo $class, "benedicendo" l'oggetto appena nato. Questo codice é funzionante e la chiamata per creare l'oggetto Umano va fatta come segue:

```
my $marco = new Umano ( 'Marco', 'Albano', 'maschio', 24 );
```

Riferendosi alle prime due righe si puó capire ora, essendo la chiamata *Umano ('Marco', 'Albano', 'maschio', 24);* passata come parametro alla funzione, come lo *shift* prenda il package *Umano* come parametro e quello diventi automaticamente la classe a cui si dovrá poi benedire l'oggetto.

2.2.3 Framework

Struts

Jakarta Struts[3], semplicemente chiamato Struts é un framework sviluppato seguendo il paradigma MVC[23]. Nato dalla collaborazione del Jakarta Project[4] con l'ASF[24], Struts permette di creare applicazioni che usano le tecnologie J2EE[46], tra le quali Java Servlet, JSP e JSP Tag Libraries[25]. Ritrovando la divisione di una applicazione nei tre livelli dell'MVC torniamo sul concetto della divisione dei compiti riferendoci ora ad una web application:

[23] Model View Controller

[24] Apache Software Foundation

[25] Le JSP Tag Libraries sono delle librerie di tag che semplificano il lavoro del web programmer aggiungendo la possibilitá di incapsulare procedure all'interno di tag e riproporle in un qualsivoglia frangente

- **Model**, in una applicazione, sia web che canonica, rappresenta i dati a cui bisogna avere accesso (tipicamente contenuti in un database) ed il set di regole (business rules) a cui bisogna fare riferimento per manipolare questi dati[26];

- **View**, rappresenta l'UI[27] dell'applicazione, nel caso di una web application si tratta della pagina web proposta dal server e visualizzata da client;

- **Controller**, é il sistema con cui si ricevono i dati, nelle web application é sia l'input dell'utente che le richieste di connessione che arrivano al server.

In parole povere un'applicazione Struts non é altro che una normale web application con l'aggiunta di alcuni particolari elementi tra cui possiamo distinguerne tre principali:

- le librerie Java di Struts tra le quali ci sono le JSP tag libraries e le classi principali di Struts: Action, ActionForm e ActionServlet;

- il file di configurazione di Struts, in XML, configura il comportamento di tutte le componenti del framework;

- Property Files, che racchiudono le impostazioni per i messaggi multilingue.

Il file di configurazione é molto importante per le applicazioni sviluppate con Struts perché si occupa di definire regole per l'utilizzo di una gran quantitá di componenti anche inerenti il comportamento delle servlet:

- ActionForm (componente aggiuntivo di Struts nominato in precedenza);

- Action Mappings;

- regole globali di forward;

- posizione di file di proprietá;

- altri componenti come plugins;

Dovendo quindi tenere conto di questo file di configurazione, il suo percorso deve essere un'entry del file web.xml (ad esempio nella directory *Web Module* di Tomcat) sotto la voce:

[26] in alcuni casi il Model puó essere pensato come la libreria JDBC
[27] User Interface, interfaccia utente

```
<init-param>
    <param-name>config</param-name>
    <param-value>\posizione\struts-configfile.xml</param-value>
</init-param>
```

Struts inoltre incorpora alcune tag libraries tra le quali la Struts HTML che fornisce varianti per i maggiori moduli Controller dell'HTML, come forms, radio buttons, checkboxes; e gli Struts Bean: libreria per accesso e creazione di Java Beans con prerogative particolari. Sotto viene riportato un esempio di triviale "Welcome Page" per un'ipotetica web application sviluppata con Struts[28]:

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>

<HTML>
<HEAD>
<TITLE>Welcome!</TITLE>
<html:base/>

</HEAD>
<BODY>

    <logic:present name="user">
    <H3>Welcome <bean:write name="name" property="username"/>!</H3>
    </logic:present>

    <logic:notPresent scope="session" name="user">
    <H3>Welcome World!</H3>
    </logic:notPresent>

<html:errors/>

<UL>
<LI><html:link forward="logon">Sign in</html:link></LI>
<logic:present name="user">

<LI><html:link forward="logoff">Sign out</html:link></LI>
</logic:present>
</UL>

<html:image src="struts-power.gif" alt="Powered by Struts"/>
```

[28] http://www.oracle.com/technology/products/jdev/collateral/papers/10g/Developing_Struts.pdf

```
</BODY>
</HTML>
```

Come si puó vedere dal codice riportato sopra la notazione di Struts essendo organizzata come tag libraries non é compatta, ma di semplice interpretazione. Con questa organizzazione é possibile giungere ad una buona conoscenza, ed alla possibilitá di scrivere semplici pagine dinamiche, in poco tempo. Ora portiamo l'attenzione in particolare ai tre componenti principali di un'applicazione web sviluppata con il framework Struts:

- ActionServlet, questo componente Struts permette di definire quello che nel modello MVC é il Controller, cioé il componente che si occupa di fornire i dati alla business logic. Chiamando una pagina "*.do" si stabiliscono delle regole per la ricerca dei dati, ad esempio una query strutturata in maniera particolare, quando, in tal senso, si debbano eseguire query con molti *join* e altre condizioni complicate. Un trigger intercetta la richiesta di pagine con estensione ".do" ed esegue le direttive contenute nel file. Quindi il risultato della query viene reso disponibile al contesto JSP per la visualizzazione[29].

- ActionForm, anche chiamati "Form Bean" sono componenti che mappano tag di input HTML su dei Java Bean fornendo tutti i metodi del caso. Una form HTML in una pagina JSP puó avere campi di testo mappati dal Form Bean. Inoltre, in alcuni casi, i form sono generati dinamicamente, per questo esiste un componente a parte, basato su XML, DynaActionForm che puó essere usato senza che questo richieda una classe Java per il Form Bean.

- Action, sono i veri e propri mattoni delle applicazioni Struts e regolano tutto ció che non ha a che fare con le UI. Devono essere riportate nel file di configurazione indicato in "web.xml" e devono essere mappate ad un URL usando un *ActionMapping*. L'ActionMapping mappa, per l'appunto, una richiesta con una classe Action. Quando l'Action ha rilevato un login effettuato con successo avendo ricevuto e controllato tutti i campi necessari ha bisogno di poter chiamare la pagina che indichi il risultato. Questo si fa con una Action Forward anch'essa definita nel file di configurazione XML come figlio dell'Action in questione.

  ```
  <action name="login" path="\loginAction" input="login.jsp"
  type="strutsfun.view.LoginAction">
  ```

[29] C'é un meccanismo uguale in Mason per cui richiedendo una pagina con un'estensione particolare il server si incarica di eseguire le query con la logica specificata in questa pagina e poi restituirla come risultato

```
<forward name="success" path="\success.jsp"/>
</action>
```

DOJO

Dojo[9], o meglio il Dojo toolkit, non é altro che un RAD[30] per applicazioni web. É un framework OpenSource scritto dalla DOJO Foundation[10] in JavaScript giunto ora alla versione 0.4.1. Fornisce agli sviluppatori di applicazioni web un set di API[31] JavaScript, quindi utilizzabili con la maggior parte dei browser, per aggiungere funzionalitá al lato client. Di seguito viene fornito un elenco delle maggiori funzionalitá di Dojo:

- possibilitá di includere nelle pagine editor di testo o addirittura attivare la possibilitá di cambiare il contenuto di una pagina dinamicamente editandola direttamente mentre la si visualizza;
- funzionalitá di *drag-and-drop* in alberi di elementi con possiblitá di aggiungere, spostare a cancellare nodi;
- possibilitá di utilizzo di bottoni con una scelta dell'operazione particolare da svolgere da un menu a tendina e di bottoni con un proprio sfondo;
- gestione semplificata di effetti quali fading[32], sliding[33], wiping[34] ed altri ancora;
- widget per scegliere una data da un calendario, un colore da una tavolozza o anche combobox e checkbox personalizzate, oppure widget per validare o correggere l'input ricevuto dall'utente dalla tastiera;
- gestione di tabelle con il sorting automatico per attributi selezionati dal click dell'utente;
- menu che ricalcano la rinomata barra di MacOS;
- box di descrizione di oggetti personalizzati;
- funzionalitá di introduzione nella pagina di layout predefiniti per facilitare sensibilmente l'uso di applicazioni piú semplici se organizzate in un certo modo (figura 2.5) o addirittura con funzionalitá come le finestre che si trovano in un sistema operativo (figura 2.6).

[30] Rapid Application Development, sviluppo rapido di applicazioni

[31] Advanced Programming Interface, interfaccia di programmazione avanzata

[32] si dice fade di un oggetto quando la sua visilitá scema o aumenta per farlo diventare rispettivamente invisibile o visibile

[33] si dice slide di un oggetto quando questo si sposta

[34] l'effetto di wipe su una finestra ad esempio é quello per cui questa finestra gradualmente scompare arrotolandosi su se stessa

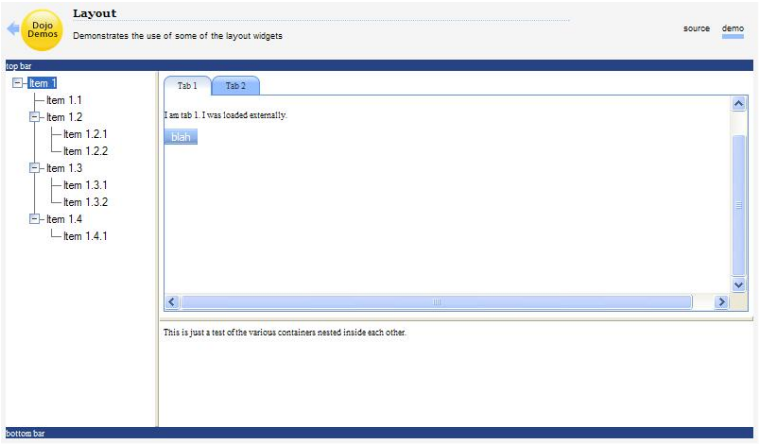

Figura 2.5: Esempio di layout di pagina con DOJO.

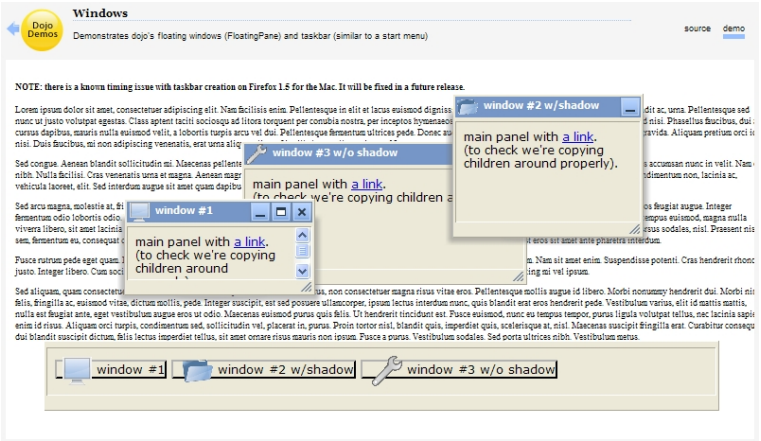

Figura 2.6: Esempio di layout di pagina in stile MS Windows con DOJO.

Con l'uso di questo framework si potranno effettivamente creare applicazioni web complesse, *user-friendly* e addirittura accattivanti per far dimenticare all'utente che in realtá sta usando un browser web. Inoltre sará possibile concentrarsi sulle funzionalitá piú che sulla forma dato che DOJO incamera opzioni che se combinate diventano opportunitá enormi per chiunque sviluppi web applications.

Capitolo 3

Proposta di un nuovo Application Server Event-driven

3.1 Motivazioni

La soluzione che attualmente l'azienda Leader.IT sta utilizzando per la proprie applicazioni é basata sulle teconologie che abbiamo visto fin'ora. In generale si tratta di un server GNU/Linux con Apache e Mason interfacciati tramire mod_perl che servono pagine HTML dinamiche JavaScript con tecnologia AJAX prendendo i dati da un database PostgresSQL. Le applicazioni sviluppate dall'azienda sono supportate da parte degli strumenti illustrati nel corso di questa tesi. In questo capitolo ci sará la spiegazione di cosa si vuole fare con questo lavoro. A che punto si vuole arrivare nel dedicare il lato server ad un lavoro particolare e non alla normale attivitá di un classico server HTTP. Effettivamente, partendo per gradi, la prima cosa che emeglio far notare é che il server per una applicazione web sull'onda di quelle sviluppate dall'azienda Lader.IT non deve essere un server web. Deve piuttosto farsi catalogare nella categoria degli Application Server. Non deve infatti rispondere a delle richieste, o restituire pagine. Deve piuttosto agire di per se ed in base ad un input sotto forma di evento. Chiariamo ora, per il nostor software, cosa é un evento, o meglio come lo intendiamo. Per noi un evento é una qualunque situazione in cui una qualunque cosa é cambiata. Volendosi spiegare meglio ci possiamo appoggiare ad una definizione di server web: il server web eún programma in grado di servire pagine HTML con dei dati contenuti. Questi dati, per quanto ci riguarda, sono presenti su un database, e sono soggetti a variazioni. É necessaria la notifica di un evento dunque quando cambiano i dati su cui si sta facendo affidamento. Per tutti i tipi di eventi si puó seguire questa logica. Anche degli errori possono essere eventi nel cui corpo sono specificate le condizioni per la compilazione, ad esempio, del file di log. Un evento é quindi una qualunque situazione che si venga a verificare e che cambi dei dati sensibili. Un aumento di un grado della CPU infatti non é un dato sensibile. Se quel grado peró permette alla CPU di superare la soglia di guardia, questo superamento é un evento da riportare.

Cosa comporta questo *modus operandi*? Molti vantaggi. Per cominciare si manderanno gli aggiornamenti solo in momenti necessari, meno banda e potenza utilizzate e quindi tempi di risposta piú bassi. Meno tempo di calcolo per aggiornamenti parziali invece che totali di un insieme di record ad esempio. Ed una conseguente velocizzazione dell'accesso ai dati da parte dell'utente. Possibilitá di avere piú finestre aperte in contesti diversi ed avere tutti i dati, in esse residenti, comunque aggiornati. Perché non sono io utente che chiedo che i miei dati vengano aggiornati, ma sará il server a dirmi che un dato é cambiato e mi invierá quello consistente. Questa architettura va immaginata piú come un insieme di servizi che client e server si scambiano che come una risposta che segue una richiesta. Infatti un evento potrá passare inosservato se non interessa ad alcun client. Come abbiamo giá detto gli eventi saranno oggetti JSON che viaggiano attraverso la rete.

Porteranno con se i dati necessari per effettuare l'aggiornamento per cui sono stati sollevati. Inoltre potranno essere di vario genere, sará quindi premura di chi é interessato a quel particolare dato, registrarsi debitamente per averne notifica.

Perché tutto questo, non si possono utilizzare strumenti esistenti? Questa domanda é piú che naturale ed é stata sollevata piú volte. La risposta sta nella domanda stessa, "Perché no?". Uno schema di questo tipo potrebbe fornire un'alternativa a molti applicativi giá esistenti. Si pensi, ad esempio, a qualcosa tipo rrdtool-based per polling di informazioni da varie macchine. Ad esempio temperatura CPU occupazione memoria principale e disco, carico di lavoro della CPU ed altro ancora. La maggior parte di questi tool fa polling sulle macchine monitorate. Questa maniera di procedere non é sbagliata perché permette di non installare nulla a parte un piccolo driver sulla macchina da monitorare. Nel caso invece di un server strutturato come spiegato prima e dei client che sono solo browser che giá di per se eseguono JavaScript uno schema di questo tipo permette di non dover installare nemmeno il driver. La logica di fondo, in definitiva sta nell'essere disturbato solo quando si ha la necessitá, e solo su cose per cui si é chiesta la notifica. Quindi é fattibilissimo con strumenti giá esistenti, o, ad ogni modo, si aggira il problema trovando altre soluzioni. Peró non é un'idea da scartare e provarci é dunque d'obbligo, perché si é comunque certi, non sará una perdita di tempo.

Da non sottovalutare, inoltre, l'alta modularitá che si puó raggiungere con questo tipo di programmazione. Sia all'interno della stessa web application, che in altri ambiti. La parte di *view* restarebbe debitamente inconsapevole del *controller* e questo aumenterebbe la modularitá. Inoltre la registrazione di eventi sarebbe "gratuita" dal punto di vista del client e del server. Nel senso che non ci sarebbe upload di *business logic* dal server sul client, perché il client sarebbe autonomo nel tradurre le volontá dell'utente nel registrarsi come ascoltatore di eventi o meno. Una volta elaborata l'interfaccia ad eventi su client e server quindi il programmatore potrebbe "perdere" piú tempo con l'interfaccia od i cicli di controllo. Perché la trasmissione dati resterebbe indipendente dal resto sia come programmazione che come concezione.

3.2 Architettura

Un'altra domanda che potrebbe nascere riguarda il perché la scelta degli strumenti affrontata in questa tesi sia stata cosí pilotata verso particolari tecnologie. Bisogna innanzitutto precisare che questo lavoro di tesi viene svolto in seguito e come approfondimento della tesi di Tarcisio Fedrizzi[48].

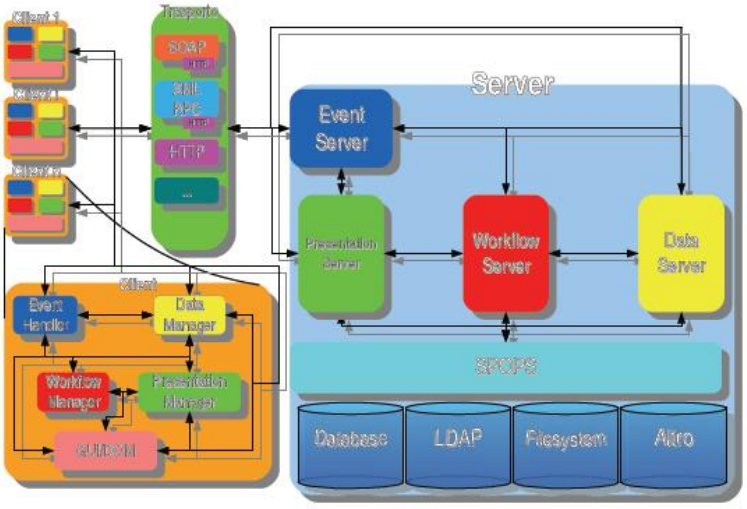

Figura 3.1: Architettura di un nuovo tipo di CMS.[49]

L'architettura in figura 3.1 si riferisce ad un CMS completo. In questa tesi si va elaborando una soluzione alla parte che riguarda il dialogo tra client e server. Parte non approfondita nel lavoro di tesi sopra citato. In questi capitoli si sono, infatti, definite le intenzioni e le incognite sono state sostituite da soluzioni piú o meno adatte. Sono state discusse svariate tecnologie e si é giunti ad alcuni punti fermi. La parte dunque che si vuole espandere riguarda non la struttura interna di un server o di un client. Ma il loro modo di comunicare e come sia opportuno farlo.

Nella figura 3.2 si possono notare le parti rilevanti ritagliate dall'architettura generale. Come spiegato nei capitoli precedenti si vuole giungere a definire come dovrebbe essere quello che si vuole sviluppare. Ció che viene definito come *Event Server* e *Event Handler*. Siamo arrivati alla conclusione che l'architettura deve essere ad eventi e permettere una forte interazione tra client e server. In questo capitolo cercheremo di definire il come questi eventi debbano essere organizzati per permettere questa interazione. Ora, un punto fermo di questa archiettura é che il server deve fare un upload di *business-logic* sul client inviandogli codice JavaScript con tecnologia AJAX[1].

[1]cambiando la X(XML) di AJAX con JSON

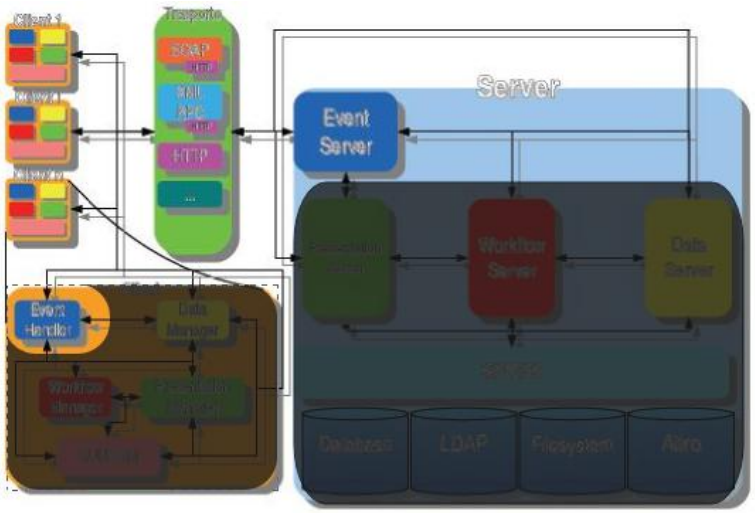

Figura 3.2: Parte illustrata in questa tesi ritagliata dell'architettura generale.

Di consueguenza, come é logico, il server sa cosa il client sta eseguendo. Questo gli permette di definire degli ascoltatori, tramite eventi JavaScript, sulle azioni che l'utente puó fare. Direttamente nel codice JavaScript che gli invia all'avvio della applicazione. Il client di conseguenza puó avere la possibilitá di aggiornare dati sul database gestito dal server senza dover premere alcun pulsante, ma solo cambiado i dati nel form di visualizzazione. Ad esempio il server ha definito un evento che manda un oggetto JSON che definisce il dato cambiato direttamente ad un *onChange* del campo del form. In questo caso ad ogni cambiamento del contenuto del form si avrá un upload di dati. Questo non é assolutamente consigliabile, ma se altrimenti l'upload del dato viene effettuato quando il dato campo di form perde il focus[2] l'utente non si dovrá preoccupare di premere il tasto di salvataggio, perché ogni edit sará salvato automaticamente. Questo é solo un esempio di cosa si puó fare e di come dovrebbe essere organizzato. Il fatto di sapere esattamente cosa il client stia eseguendo permette al server di organizzare il proprio lavoro come meglio crede. Ovviamente si deve trovare un compromesso tra il salvataggio di dati repentino e quello a compilazione terminata (restando sempre sull'esempio del form). Ognuna ha svantaggi e vantaggi. Senza addentrarci nei problemi di programmazione che esulerebbero dallo scopo di questa tesi, abbiamo, a grandi linee, illustrato come il server dovrebbe comportarsi avendo a disposizione strumenti del genere illustrato. Il client, da

[2]cioé quando da quel campo si va ad editarne un altro o comunque il mouse clicca altrove

parte sua, ha gli stessi mezzi per avere un efficiente e vantaggioso dialogo ad eventi con il server. La *business logic* infatti deriva comunque dal server dove il programmatore ha definito pagine HTML (il front end) corredate da codice JavaScript. Nel caso del client infatti si potrá avere la possibilitá di registrare per l'update automatico alcuni o tutti i campi visualizzati, in maniera tale da avere sempre un elenco aggiornato dei dati su cui si sta lavorando, tutto senza dover premere il pulsante di aggiornamento e dover attendere il ricaricamento della pagina o di tutti i dati in essa contenuti come avviene nella maggior parte dei web server attuali. Ogniqualvolta il server aggiornerá un campo nel database potrá inviare un oggetto JSON di aggiornamento ad ogni client che si é registrato per quel campo. La registrazione, ovviamente, sará vincolata strettamente all'effettiva visualizzazione del dato da parte del client[3].

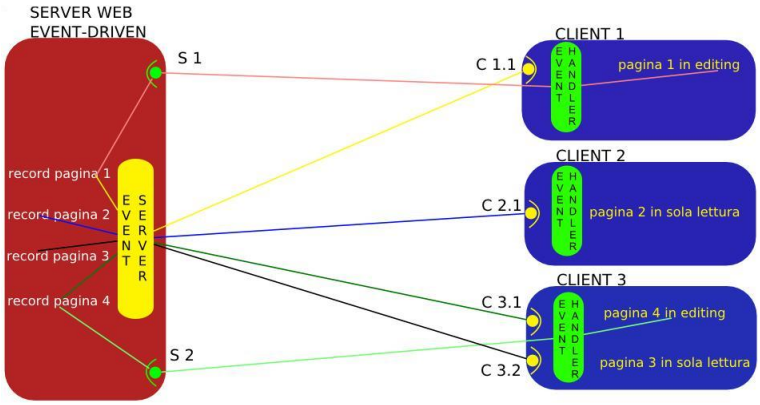

Figura 3.3: Esempio di client e server con eventi registrati.

In figura 3.3 si puó vedere un'idea primitiva di come possano essere organizzate le registrazioni degli eventi tra client e server. Sulla sinistra si puó notare il server con il database[4] ed il nostro *Event Server*. Sulla destra invece tre client collegati al server ognuno con un'azione diversa.

- Client 1, ha caricato la pagina 1 e la sta editando. Quindi il client si deve preoccupare di registrarsi sul server per gli aggiornamenti a quella pagina. Questo compito é assolto dall'*observer C1.1* che si é registrato tramite l'*Event Server* come osservatore dei cambiamenti sulla pagina 1. Questa pagina peró é in editing da parte del client e questo significa che si potranno verificare dei cambiamenti su di essa da registrare sul database. Di questi si occupa l'observer *S1* che interfacciatosi tramite

[3]sarebbe inutile e sprecherebbe banda aggiornare il dato su un client che non lo sta visualizzando o che non lo ha in memoria

[4]contenente i record delle pagine, cioé i dati che esse mostrano

l'*Event Handler* nel client osserva i cambiamenti che questo porta ai record della pagina 1.

- Client 2, sta solo leggendo la pagina 2, di conseguenza l'unico observer che avrá (*C2*) sará collegato all'Event Server in ascolto sui cambiamenti che potranno essere effettuati da terzi sui dati della pagina 2.

- Client 3, ha caricato la pagina 4 in editing e la pagina 3 in sola lettura. Questo significa che il server dovrá avere l'observer *S2* per sincronizzare i cambiamenti effettuati dal client su quella pagina con il database. Il client invece dovrá avere observer sia per i cambiamenti sulla pagina 3 (*C3.2*) che sulla pagina 4 (*C3.1*).

Capitolo 4

Stato dell'arte dei Web Servers Event-Driven

4.1 Perché Web Servers Event-Driven

Il quesito che ha generato questa tesi riguarda la possibilitá di poter organizzare le applicazioni web in modo tale da permettere una certa autonomia al client ed al server sulle scelte che riguardano le interazioni tra loro. Come é giá stato illustrato negli scorsi capitoli il modello *request-response* non é in grado di soddisfare pienamente questa necessitá. Ci si é quindi concentrati sul modello *MVC* comprendendo che sarebbe stato un cammino utile da seguire per arrivare ad un risultato quantomeno soddisfacente. Ci si é quindi interrogati sulle tecnologie in grado di servire pagine web dal contenuto dinamico e dei web servers che le supportano. Dalle relazioni redatte nelle rispettive sezioni si é cercato di individuare quali potessero essere usate lato client e quali lato server. Per il primo sono stati individuati AJAX con JavaScript e JSON ed in un futuro, in vista di una release piú stabile, il framework DOJO. Per il lato server abbiamo studiato i due maggiori Web Server presenti sul mercato giungendo alla conclusione che non sono adatti a risolvere il problema. Nei prossimi capitoli presteremo attenzione alle tecnologie per server *Event-Driven* esistenti ed ai web servers ad eventi esistenti. Questa scelta potrebbe essere contestata con l'argomentazione dell'esistenza di web servers che, nonostante non siano *Event-Driven*, con alcuni accorgimenti di programmazione delle web applications possono fare comunque al caso nostro. La risposta a questa osservazione puó essere fornita dai due principali concetti della filosofia Perl:

- there is more than one way to do it[1];
- the goal is to make easy tasks easy and difficult tasks possible[2][38];

e la risposta é appunto un'unione delle due, ci sono varie strade per arrivare all'obiettivo di questa tesi e vogliamo percorrerne una difficile per poterla rendere possibile. Inoltre ci sono anche motivi aziendali. Come giá accennato questo lavoro di tesi si svolge al termine di un tirocinio presso l'azienda Leader.IT e con il tutor aziendale é stata decisa la traccia. In questo periodo d'attivitá presso l'azienda si é entrati in contatto con una realtá diversa da quella universitaria. Fatta si di continua evoluzione imposta dalle molteplici soluzioni presenti, ma anche di un'attenzione molto particolare al dover innovare solo alcuni aspetti delle conoscenze acquisite durante i vari lavori commissionati. Questo perché mentre le tecnologie vanno avanti e si specializzano o cambiano sostanzialmente[3], per un'azienda non é pensabile di dover riprendere a studiare uno strumento giá usato solo perché si

[1] c'é piú di un modo per farlo

[2] l'obiettivo é di rendere facili i problemi facili e possibili quelli difficili

[3] ad esempio .NET che in due versioni differenti cambiava vertiginosamente il modo di scrivere codice

userá una versione diversa. Ed inoltre per un problema cosí particolareggiato non si possono utilizzare strumenti standard, come giá esaurientemente spiegato in precedenza.

4.2 Tecnologie esistenti applicate ai Web Servers Event-Driven

Le tecnologie esistenti per quanto riguarda web servers ad eventi non sono effettivamente molte come si potrebbe pensare. JSP e quindi ASP, come illustrato nei capitoli precedenti, non possono fare al caso nostro. Per motivi aziendali inoltre particolare attenzione va rivolta verso tecnologie sviluppate con strumenti giá utilizzati e questo ci guiderá nelle scelte che andremo ad affrontare nei prossimi capitoli.

4.2.1 Client

Javascript come un client Event-Driven

Avendo avuto cura di illustrare il funzionamento basilare di JavaScript, in questo paragrafo entreremo un po' piú nel dettaglio della gestione che questo linguaggio fa degli eventi DOM. Come giá illustrato JavaScript interagisce con il browser usando il Document Object Model come interfaccia. Questo DOM permette la gestione di eventi come la pressione di un bottone o una particolare posizione del mouse. Con l'aiuto dei CSS[23] e dello stesso JavaScript, si possono programmare applicazioni client anche piuttosto complesse. Tutto questo sommato da la possibilitá di costruire applicazioni complesse lato client con una gestione ad eventi. Proprio quello che serve. Per essere sicuri di questo indaghiamo piú a fondo la gestione degli eventi in JavaScript cercando di capire fino a che punto si puó controllare il browser attraverso l'interfaccia DOM.

Oggetto Event , ad ogni evento che accade nella finestra del browser il DOM si preoccupa di interfacciarlo tramite un oggetto JavaScript *Event*. Questo oggetto porta con se delle proprietá. Ad esempio l'oggetto "MouseDown" si verifica ad ogni pressione di un qualunque pulsante del mouse all'interno della finestra del browser. Questo oggetto ha degli attributi che ne chariscono la natura, come ad esempio le coordinate del mouse al momento della pressione del pulsante. Altri attributi disponibili dalla chiamata sono quelli riguardanti quale pulsante del mouse é stato premuto, come quali tasti di controllo[4] erano premuti al momento della ricezione dell'evento. In questo modo si puó avere un quadro esatto delle situazione in cui si é verificato l'evento e procedere ad una sua gestione dai livelli piú basilari a quelli piú avanzati.

Catturare gli Eventi , questi eventi possono essere gestiti dall' oggetto su cui si verificano. Questo puó comunque essere aggirato se si vuole una gestione uniforme degli eventi tramite funzioni proprie. In questo senso si

[4]sono detti tasti di controllo i tasti *shift*, *ctrl*, *alt* e *meta*

possono intercettare gli eventi prima che raggiungano il gestore specifico tramite delle funzioni detti appunto *handler*:

- captureEvents, cattura l'evento il cui tipo é specificato nella chiamata;
- releaseEvents, ignora gli eventi del tipo specificato nella chiamata. Assieme alla funzione precedente puó essere usata per dividere eventi appartenenti ad uno stesso gruppo da gestire in maniera diversa;
- routeEvents, reindirizza un evento su un oggetto specificato, questa funzione é diversa dalla precedente perché si puó far seguire un iter ad un evento dipendentemente da quale handler si vuole che segua. Nel caso in cui nessuna delle funzioni implementate dal programmatore accetti l'evento quasto verrá automaticamente dirottato verso l'handler standard dell'oggetto che l'ha generato;
- handleHevents, gestisce l'evento catturandolo prima che questo venga gestito dalla gerarchia, un esempio di questo sono le righe seguenti che impongono a tutti gli eventi "Click" di andare ad interessare il primo link presente nella pagina:

  ```
  function clickHandler(e) {
      window.document.links[0].handleEvent(e);
  }
  ```

  Questa funzione, naturalmente, di per se non ha alcun effetto sul alcun evento. La gestione degli eventi infatti necessita della registrazione dell'EventHandler che si é dichiarato. La riga seguente effettua la registrazione dell'handler come ascoltatore di tutti gli eventi "Click":

  ```
  windows.onClick = clickHandler;
  ```

Questo tipo di programmazione inoltre permette di implementare un controllo dei dati giá a livello client, senza che il server si debba sobbarcare l'onere di farlo per tutti gli utenti collegati. In questo senso vengono in aiuto principalmente i metodi "onChange" e "onClick" che controllano rispettivamente un cambiamento in un form ed il click su un pulsante[5]. Cosí operando si possono facilmente aggirare molti ritardi e attese inutili velocizzando sia il processo client, che ha subito un responso sui dati inseriti, che quello server che in linea di massima non deve ricontrollare i dati.

[5]il click non deve essere necessariamente sul pulsante, si possono infatti intercettare pressioni dei pulsanti del mouse ovunque all'interno dell'oggetto *window*, ma in questo caso si sta facendo un esempio sulla pressione di un pulsante di submit di un form

4.2.2 Server

POE Event-Driven Web Server

POE[42] é un framework per creare applicazioni *Event-Driven Multitasking* scritto in Perl da Rocco Caputo[42]. La sua prima release fu nell'agosto 1998 e giá un anno dopo vinse il "Best New Module" award alla Perl Convention, ora OSCON[6]. É strutturato a grandi linee come un sistema operativo, cioé serve in multitasking cooperativo. Ogni sessione deve cooperare lasciando spazio alle altre lessioni per il possesso della CPU. POE é diviso nelle seguenti parti

- **Kernel**, l'oggetto caricato in memoria, smista eventi alle sessioni, eventi di rete, eventi lanciati dal sistema operativo come anche eventi creati manualmente;
- **Sessions**, sono simili ai processi per i sistemi operativi, sono quelle che svolgono il lavoro, contengono un heap e degli eventi;
- **States**, sono degli eventi che si ricevono, come quelli predefiniti i cui principali sono:
  - _start, chiamato quando la sessione viene creata, viene usata in generale per creare connessioni, aprire file ed altre operazioni da portare a termine all'apertura della connessione;
  - _stop, chiamata quando la sessione viene chiusa e deve essere "garbage collected", al contrario della precedente é usata per chiudere file, fare log e tutto ció che si debba fare prima di chiudere una connessione;
  - _signal;
  - _parent;
  - _child;
  - _default;
- **Filters**, filtri input, output ed in generale sugli stream;
- **Wheels**, scambiano dati attraverso i socket e le sessioni attraverso i filtri;

Volendo dare uno sguardo agli event-handler che sono messi a disposizione di una sessione si notano i piú importanti divisi in due categorie:

- ASAP[7]

[6] Open Source Convention

[7] *as soon as possible*, il piú presto possibile

  - $kernel->post($session, “event”, @arguments), restituisce il controllo ad una sessione particolare specificata dal parametro $session che puó essere un alias, un ID numerico od una reference;
  - $kernel->call($session, “event”, @arguments), usato quando sia necessario farsi dare il valore di ritorno da un evento;
  - $kernel->yield(“event”, @arguments), richiama la sessione corrente, il kernel sa quale é in esecuzione in quel momento;

- Delayed calls
  - $kernel->delay_set(“events”, $seconds, @arguments), usato per chiamare un evento qualche secondo dopo, eventi peró rigorosamente nella sessione corrente;
  - $kernel->alarm_set(“events”, $seconds, @arguments), usato per implementare funzionalitá come quelle offerte dal demone *cron* di UNIX;

Le wheels sono uno strato di astrazione di POE verso le periferiche di I/O. Sono create dalle sessioni e restituiscono a queste degli eventi quando ne accade qualcuno. Non sono automatiche e vanno create quando si istanzia una sessione e risiedono nel suo HEAP. I passi per poterla utilizzare sono la creazione, l'istanziazione nell'HEAP della sessione e l'implementazione che puó essere diversa dipendentemente da cosa si vuole che venga fatto quando viene sollevato l'evento a cui quella wheel deve rispondere. Ad esempio POE::Wheel::ReadLine legge l'input da riga di comando, fa lo stesso che si potrebbe fare con $line = <STDIN> in Perl, ma con la wheel questa chiamata non é bloccante come il normale comando Perl. Questi accorgimenti sono essenziali perché bisogna sempre ricordare che si é in un ambiente multitasking cooperativo. Cioé ci sono thread simulati concorrenziali nel tempo in cui il kernel POE utilizza la CPU. Questo significa che bloccando una sessione su una chiamata si blocca il processo POE tutto intero. I Filters invece, come suggerisce il nome, sono dei moduli che raggruppano i dati provenienti da una wheel customizzandoli dipendentemente da quale si é usato. Oltre ad avere la possibilitá di far girare piú sessioni[8] cooperative sulla stessa macchina, ma con il modulo build-in POE::Component::IKC[9] si potranno far comunicare e quindi cooperare piú kernel. Inoltre, essendo il modulo di comunicazione inter-kernel, basato sulla rete si potranno avere applicazioni distribuite per processi paralleli su macchine diverse.

[8] non é possibile, come illustrato prima, parlare di concorrenza, si dirá invece cooperativitá

[9] inter-kernel communications

4.3 Web Servers Event-Driven attuali

Ora che é assodato che per una corretta programmazione di web application con le caratteristiche prima illustrate c'é necessitá di un web server event-driven non possiamo non interrogarci su quali siano disponibili.

4.3.1 Comet Tecnology

La tecnologia Comet[43] nasce dopo l'avvento di AJAX proprio perché il modello request-response non si rivela adatto come soluzione del problema illustrato nelle prime parti della tesi. Comet infatti vuole poter rendere il client indipendente dal server per quanto riguarda l'aggiornamento, dalle pagine intere alle singole parti di una pagina. L'idea é quella di mantenere attiva una connessione rendendola persistente con l'invio saltuario di qualche byte ignorato dal server per evitare che il canale HTTP venga chiuso. Impedendo la chiusura del canale HTTP di puó trascurare di doverlo riaprire ogni volta che si vuole mandare un pacchetto d'aggiornamento o quando si aggiorna una pagina. Questo permette di rendere piú veloce la trasmissione dei dati ed elimina il fastidioso overhead per la riapertura della connessione. Inoltre Comet prevede che l'aggiornamento, qualunque esso sia, non venga "organizzato" da chi lo richiede, ma venga attuato da chi possiede le informazioni nuove e solo nel momento in cui le possiede. Questo implica di per se una gestione degli eventi ed ogni tentativo di implementare queste idee con un paradigma request-response, puó funzionare, ma resta una soluzione poco elegante[10].

I vantaggi non si contano peró solo sul lato server come la facilitá di inserimento di nuove features nel proprio software o l'uso minore di banda e CPU per servire le richieste di aggiornamento o l'apertura di nuovi canali HTTP. Sul lato client infatti questa maniera di procedere permette di avere tempi di risposta piú immediati, meno overhead su richieste minori e piú velocitá su scambio di ingenti quantitá di dati. C'é, nondimeno, la possibilitá di organizzare gli aggiornamenti in maniera semplice e meno invasiva, per esempio in background mentre l'utente sta facendo altro. Come si vede dalla figura 4.1 le differenze con il puro Ajax (figura 2.4) non sono rivoluzionarie, ma importanti. Come si vede, rispetto al puro Ajax, Comet prevede uno scambio di dati seguentemente al verificarsi di eventi da entrambe le parti. Ajax non prevedeva questa soluzione, ma prevedeva aggiornamenti solo a valle di un input del client. Questa dipendendenza é proprio quella che con questo lavoro si vuole evitare. Si spera di arrivare, e Comet é un bel passo avanti in questa direzione, ad uno scambio di dati tra client e server solo a seguito di eventi verificatisi sul primo o sul secondo.

[10] come é stato accennato é infatti la soluzione implementata attualmente dall'azienda Leader.IT per il proprio software

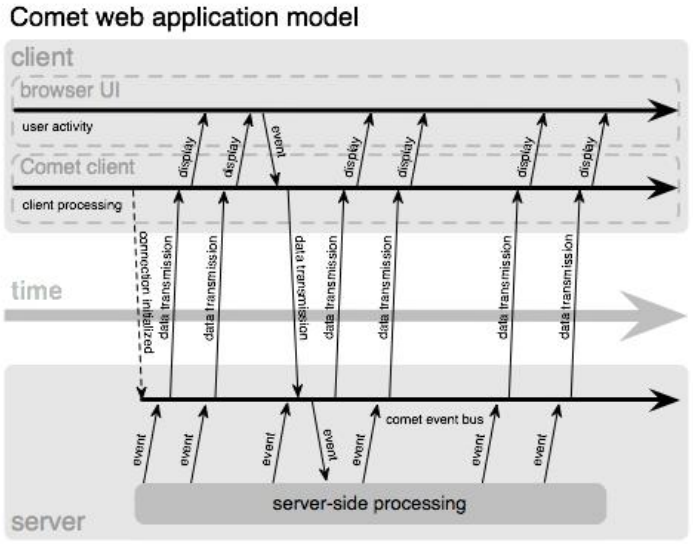

Figura 4.1: La maniera di procedere di Comet.

4.3.2 Flash Web Server

Flash[44] é un server web elaborato da uno studente di Princeton che ha incentrato il suo lavoro sulle performance raggiungendo ottimi risultati. Nel documento di presentazione viene messo in risalto come una tecnologia AMPED[11] possa eguagliare e superare le performance dei due piú performanti Web Server. Apache e Zeus[45] infatti vengono rapportati alla versione ufficiale di Flash ed ad alcune varianti e analizzati con queste in termini di prestazioni. Andando un po' piú nello specifico si puó presentare Flash come un web server event-driven basato su un solo processo a differenza di Apache che é MP[12]. Quando deve servire richieste provenienti dalla rete si comporta come un server SPED[13]. Diventa AMPED quando deve recuperare file dal disco. Per evitare che il singolo processo del web server stalli a seguito di una chiamata bloccante si lascia che questa venga servita da un processo *helper* supplementare.

Questo helper viene istanziato e incaricato di leggere il file dal disco e si blocca quindi in attesa che l'accesso al disco venga completato. Una volta fatto notifica al processo principale di aver esaurito il compito e gli restituisce il file caricato. Tale maniera d'agire permette al server di non bloccarsi qualunque richiesta faccia, bloccante o meno. L'idea di adottare questo procedimento al posto di un puro MultiProcess o MultiThreaded(MT) é giustifi-

[11] Asymmetric MultiProcess Event-Driven
[12] MultiProcess
[13] SingleProcess Event-Driven

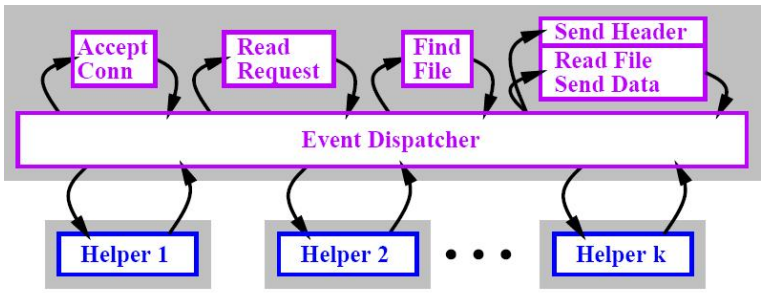

Figura 4.2: La struttura del Web Server Flash.

cato dai programmatori con la necessitá di velocizzare il server. Dimostrato infatti con sperimentazioni di velocitá su carico di lavoro che il sistema SPED di Flash supera in velocitá Apache(MP su UNIX e MT su Windows NT) e Zeus(SPED puro o SPED multiplo su macchine multiprocessori). Rispetto alle applicazioni MP, inoltre, un'applicazione AMPED permette di avere meno occupazione di memoria e risulta quindi piú leggera. Questo perché i processi che effettuano gli accessi su disco sono processi molto piccoli che non devono fare altro che quello. Non devono, per capirsi, gestire richieste o connessioni HTTP come nelle MP che hanno piú dispendio di memoria e piú overhead del kernel. Puó comunque risultare pesante creare un processo per ogni accesso al disco e per questo Flash mette a disposizione un buon numero di procedure di caching affatto trascurabili.

- Per prima cosa il caching delle informazioni richieste piú spesso. Questo punto potrebbe sembrare banale, ma in realtá non lo é. Si pensi ad esempio alle applicazioni MP e MT. Per mantenere informazioni di cache le prime devono avere delle pipe tra l'uno e l'altro processo o con uno ulteriore che si incarica di trattare le informazioni che riceve. Nel caso in cui il server sia MT allora ci devono essere delle politiche di accesso alla cache per evitare problemi nelle sezioni critiche.

- Connessioni senza interruzione, come abbiamo visto giá nel progetto Comet questo accorgimento permette di evitare di sovraccaricare la rete ed i client con connessioni non veloci dovendo ogni volta ricreare il canale HTTP sprecando tempo e banda.

- Caching delle traduzioni dei path, con questa tecnica Flash memorizza nella cache le richieste di traduzione di path e si ritiene che questo uso maggiore di memoria cache sia minore rispetto al tempo ed alla memoria usati per l'attivazione e l'uso di un helper per la traduzione ad ogni richiesta.

- Caching degli header dei file richiesti ripetutamente. Una volta che si costruisce un header di una risorsa richiesta questo viene memorizzato e servito fino a quando non avviene un cambiamento nella risorsa. A fronte di questo cambiamento viene rigenerato l'header e memorizzato al posto del vecchio.

- File mappati in cache con l'algoritmo LRU[14].

Per testare, infine, Flash rispetto a due web servers piú diffusi gli autori di questo software ne hanno implementato quattro versioni. AMPED, quella qui presentata, SPED di implementazione simile a Zeus, MP come Apache su macchine UNIX ed MT (Apache su macchine Windows NT). L'ultima peró non é presa in considerazione nell'esposizione delle prove di carico. L'esame della scalabilitá di Flash é fatto invece nelle altre tre forme e l'AMPED da risultati migliori di entrabi, Zeus ed Apache.

4.3.3 Catalyst Web Application Server

Catalyst[47] é un framework MVC-based con un server HTTP build-in con un alto grado di personalizzazione. É per questa ragione che é stato inserito in questo capitolo piuttosto che in quello riguardante i framework come Struts. Ha direttive infatti che modificano il comportamento del server, come ad esempio le mappature di URL, direttamente nel framework. Questo lo fa apparire molto simile a Mason, ma la sua struttura MVC non si poteva porlo nei Web Server tradizionali. Si é quindi scelto di posizionarlo in questo capitolo perche', oltretutto, essendo scritto in Perl potrebbe rappresentare una soluzione all'argomento sviluppato in questa tesi. Come giá detto Catalyst funziona con un paradigma MVC dove:

- Model, accesso e modifica di dati, tipicamente un database, gli strumenti individuati dal team di Catalyst sono Class::DBI, Net::LDAP ed altri;

- View, presentare il contenuto del database, gli strumenti qui sono HTML::Template o anche Mason(giá affrontato);

- Controller, gestire connessioni, controllare parametri, "flow-control", é Catalyst stesso.

Una cosa, inoltre, che fa di Catalyst un ottimo framework é la possibiltá di usate tutti i moduli CPAN senza alcun bisogno di usare plugin come ad esempio mod_perl per Apache con Mason. Qui é tutto automatico, e si é giá visto quanto software giá pronto e testato sia disponibile nell'archivio

[14]Last Recently Used é una tecnica di rimpiazzamento pagine che all'atto di caricare una nuova pagina con memoria piena scarica quella meno usata o con un algoritmo probabilistico una delle meno usate, quest'ultimo si chiama Pseudo-LRU

Perl. C'é la possibilitá con Catalyst di presentare dati reperiti con lo stesso modello attraverso differenti template di view come *Template Toolkit* oppure *PDF::Template*. Naturalmente l'applicazione sviluppata con Catalyst, dividendosi in diversi componenti, permette di riutilizzare questi ultimi in altre parti della stessa applicazione od in altre applicazioni. Catalyst permette di accedere tramite URL direttamente ad applicazioni chiamate "Actions" dichiarate nel codice con un particolare riferimento:

```
sub hello : Global {
    my ( $self, $context ) = @_;
    $context->response->body('Hello World!');
}
```

Questa action é accessibile direttamente digitando nel proprio browser emphhttp://localhost:3000/hello. In questa maniera é permesso al programmatore un ulteriore metodo di debug per le applicazioni in sviluppo o che presentano problemi. Non é necessario infatti scrivere applicazioni monolitiche, ma anche all'interno di ogni modulo, nel modello, nella vista e nel controllore si possono avere piccoli componenti. Ovviamente Catalyst supporta componenti quali Catalyst::Engine::Apache o Catalyst::Engine::CGI per mantenere compatibilitá ed usare questi strumenti. Fornisce un oggetto "Context" che mette a disposizione ad ogni componente informazioni quali l'oggetto *request* o la condivisione di risorse. Mette inoltre a disposizione una funzione di autoload dei componenti senza dover usare la direttiva *use*. Come giá detto infatti Catalyst permette di usare, ad esempio, piú View per una sola applicazione. L'utente potrebbe avere cosí la possibilitá di mandare l'output della richiesta per email, o trasformarlo in PDF o altro ancora. I moduli sono, come abbiamo giá detto, divisi nelle tre categorie del paradigma MVC e da questo Catalyst prende spunto per la separazione fisica dei componenti al suo interno:

- MyApp/Model/
  MyApp/M/
- MyApp/View/
  MyApp/V/
- MyApp/Controller/
  MyApp/C/

Questa divisione in directories consente giá a Catalyst di distinguere i componenti in moduli. Quelli che implementano il Controller saranno nella directory Controller, quelli del View saranno all'interno della directory View e via dicendo.

```
sub hello : Global {
    my ( $self, $c ) = @_;
    $c->stash->{template} = 'hello.tt';
}

sub end : Private {
    my ( $self, $c ) = @_;
    $c->forward( $c->view('TT') );
}
```

Queste righe sono un blando esempio di un primo, abbozzato, View. La prima funzione si riferisce alle righe di codice illustrate prima. L'unica differenza é la definizione del Template *hello.tt*. La funzione *end* viene eseguita da Catalyst alla fine, quindi quando appunto si dovrebbe visualizzare la pagina. E qui infatti viene mandata (*$c->forward*) la pagina con il viewer stabilito. Il modello, come giá accennato, non é necessario sia parte integrante dell'applicazione sviluppata con Catalyst. Anzi, puó tranquillamente essere un modulo esterno sviluppato per altri scopi. Le righe seguenti riportano come esempio l'inclusione di un modulo giá esistente per il collegamento ad un ipotetico database:

```
package MyApp::Model::DB;

use base qw/Catalyst::Model::DBIC::Schema/;
__PACKAGE__->config(
    schema_class => 'Some::DBIC::Schema',
    connect_info => ['dbi:SQLite:foo.db', '', '', {AutoCommit=>1}]
);
1;
```

E cosí facilmente "Some::DBIC::Schema" ora é parte dell'applicazione Catalyst con il riferimento "MyApp::Model::DB". Cosí puó essere utilizzata come un normale componente senza problemi di sorta. Una volta fatto ció lo scheletro della vera e propria applicazione Catalyst rimane il Controller:

```
package MyApp::Controller::Login;

use base qw/Catalyst::Controller/;

__PACKAGE__->config(
    actions => {
        'sign_in' => { Path => 'sign-in' },
        'new_password' => { Path => 'new-password' },
        'sign_out' => { Path => 'sign-out' },
```

```
    },
);

sub sign_in : Action { }
sub new_password : Action { }
sub sign_out : Action { }
```

Nella parte che riporta le *actions*, nelle ultime righe di codice riportate sopra, si possono notare i riferimenti alle funzioni vuote sottostanti. Sono semplicemente i riferimenti alla dichiarazione la cui implementazione dovrebbe essere nelle parentesi graffe. L'applicazione presentata non fa nulla, é solo un esempio molto semplice di come é strutturata una web application in Catalyst.

4.3.4 Comet Deamon Web Server

Il progetto di un server web basato su tecnologia Comet ha portato al repentino studio ed alla successiva implementazione da parte di David Davis[50] del web server event-driven Cometd. Questo server é disponibile nella versione in Perl ed in Pyton. Presto sará disponibile una versione in Java. Il progetto poco dopo aver preso vita é stato assorbito dalla Dojo Foundation ed é ora sotto la sua sponsorizzazione. Si trova in rete una versione del software non documentata ed in fase di sviluppo raggiungibile tramite un server SVN[51]. La versione attuale é alla revisione 261 ed é quella presa in considerazione aggiornata il 29 Gennaio 2007. La mancanza di documentazione esterna ed interna al software[15] rende particolarmente difficile una sua recensione. Il sistema fornito con la recensione sopra citata eád ogni modo funzionante e permette, con le opportune modifiche, di avere un server Parlbal[52] che risponde alle chiamate sulla porta 80. Il settaggio della porta é naturalmente modificabile nel codice sorgente[16]. Perlbal é un web server con *reverse proxy*[17] e *load-balancing*[18] anteposto a Cometd per pura comoditá. Perlbal implementa un sistema di plugin per cui epossibile "sovrascrivere" il comportamento del server con delle procedure proprie. Qui viene inserito Cometd, che di default eprogrammato per rispondere sulla porta 8080. Dal codice che si ha a disposizione si evince l'uso, per la gestione delle sessioni, di POE. Questo strumento é giá stato affrontato ed é risultato estremamente interessante. Il fatto che Cometd, nella sua versione Perl, lo utilizzi é decisamente un punto a suo favore. Inoltre per dialogare con i client DOJO si fa riferimento, sempre all'interno di Cometd, a Bayeux[53]. Un protocollo implementato per scambiare oggetti JSON con il

[15] sotto forma di commenti

[16] ricordiamo infatti che Perl é un linguaggio interpretato

[17] questo sistema permette ad un server proxy di reperire informazioni richieste da un client su un altro server rendendo peró questa operazione trasparente al client

[18] sistema che permette di bilanciare il carico

server di Cometd. Il server Cometd é implementato attraverso degli oggetti Perl che fungono da moduli. Nella versione presa in considerazione per avviare il server Cometd bisogna lanciare lo script Perl che avvia Perlbal. Dopo si puó avviare uno script *eventserver.pl* che ascolta sulla porta 8080 e delega ogni connessione su quella porta al server Cometd. Il server Cometd avvia un altro componente chiamato *Connection* che si prende cura di aprire un canale privato con il client. Il server Cometd rimane quindi in attesa di un evento dal client. Quando questo arriva si avvia una procedura di *local_accept* registrata da un ascoltatore di POE. La richiesta viene girata, come da istruzione POE, ad una funzione di *Input_Event* che si interroga di quali plugin siano stati registrati per quell'evento su quel server[19]. Se non sono stati registrati plugin per quell'azione viene chiamato un altro oggetto standard. Questo oggetto, naturalmente fornito dalla versione SVN presa in esame, non é programmato per rispondere alla richiesta. Ad ogni modo il metodo *handle_event* é programmabile dall'esterno per fare ció che si ha bisogno che faccia. Per ogni richiesta viene chiamato quel plugin e vengono passati come parametri stringhe che definiscono il tipo di chiamata. Questo oggetto *Plugin* quindi é interamente programmabile ed infatti nei radi commenti del codice che segue si possono notare le direttive per scrivere funzioni che rispondano alle diverse richieste che possono venire dal client o da direttive interne.

```
sub handle_event {
my ( $self, $event ) = @_;

return $self->$event( splice( @_, 2, $#_ ) )
if ( $self->can( $event ) );

return undef;
}

# Plugins can define the following methods
# all are optional

# ------------------------------------------------------------
# server
# ------------------------------------------------------------
# local_connected
# local_receive
# local_disconnected

# ------------------------------------------------------------
```

[19] anche Perlbal funziona in questo modo, per ogni richiesta esamina la coda di plugin registrati per generare la risposta

```
# client
# ------------------------------------------------------
# remote_connected
# remote_receive
# remote_disconnected (TODO not setup yet)
```

Come si evince leggendo il codice sorgente della versione scaricata questo progetto é ancora acerbo. Manca inoltre un'adeguata documentazione per l'utilizzo e quindi a maggior ragione per le modifiche necessarie per renderlo idoneo allo scopo. Quello che peró si puó dire di questo strumento in via di sviluppo é tutto positivo. Utilizza tecnologie stabili di cui si é discusso nei capitoli precedenti. Il fatto che sia ora sotto la Dojo Foundation ne aumenta l'appetibilitá aggiungendo la garanzia di innovazione e affidabilitá.

Capitolo 5

Osservazioni finali

5.1 Conclusioni

5.1.1 Osservazioni generali

Dallo studio portato avanti emergono alcuni aspetti interessanti riguardo le tecnologie illustrate in questo documento Per prima cosa appare ovvio che non esiste una soluzione pronta all'uso per il problema illustrato. Emerge altresí un importante fattore che riguarda la novitá delle problematiche affrontate. La mancanza, infatti, di una soluzione definitiva adatta al problema potrebbe scoraggiare un'eventuale messa in pratica di una tecnologia sensibilmente diversa da quelle disponibili attualmente. Inoltre gli strumenti individuati nel corso dello scritto sono per la maggior parte poco utilizzati in ambito accademico. Infatti ambienti e strumenti come Perl, quindi POE o JSON non sono affrontati nel corso di studi di cui questo é l'elaborato finale. I problemi che la Leader.IT deve affrontare non sono paragonabili a quelli di una grande azienda che si occupa di *industria del software*. Questo non significa che siano problemi minori o piú grandi, ma sono differenti perché aziende piccole si devono accupare di *artigianato del software*. Dipendentemente dalle scelte strategiche che questa azienda ha effettuato nel tempo, pagando gli errori e traendo vantaggi dalle scelte giuste, la logica di pensiero é variata portando gli studi e l'orientamento verso un certo tipo di soluzioni. Per questo strumenti come Java o XML non sono stati presi come riferimenti per la soluzione. Questi strumenti rappresentano una risorsa non indifferente per le grandi aziende che scrivono software, ma risultano troppo pesanti per realtá piú piccole. La scelta degli strumenti ed i giudizi oggettivi assegnati alle varie tecnologie prese in esame in questa tesi sono stati influenzati da queste premesse e da questa tipologia di realtá aziendali.

5.1.2 Conclusioni finali

La soluzione prospettata é una soluzione che, prima di tutto, risolve il problema, e che utilizza degli strumenti giá noti alla Leader.IT. POE, oltretutto, era stato individuato come primo candidato per lo sviluppo di un web server event-driven se mai non ci fosse stato qualcosa di esistente. La necessitá di utilizzare JavaScript lato client inoltre fornisce uno spunto per l'utilizzo di JSON per la trasmissione di dati. Essendo un protocollo supportato nativamente da JavaScript e disponibile in Perl tramite moduli CPAN la scelta é quasi obbligata. La soluzione individuata é il web server Cometd, ancora in via di sviluppo. Da ció che eémerso dalla sua analisi questo strumento avrá, una volta passato in fase di release, la flessibilitá e la robustezza adatte per la scrittura del server web event-driven caratterizzato nel capitolo precedente. Il fatto che sia ancora in fase di sviluppo potrebbe essere un deterrente nella scelta, ma di contro si ha l'inclusione di questo progetto nella gamma delle sponsorizzazioni della Dojo Foundation. Questa puó rappresentare la garanzia di serietá e sicurezza di compimento che in mano ad un privato

sarebbero potute essere l'anello debole della catena. L'utilizzo di strumenti giá ampiamente conslidati nell'ambito del software d'artigianato come POE e JSON rappresenta un vantaggio, sia per le ragioni illustrate nelle relative sezioni, sia perché sono giá utilizzati e conosciuti dal personale dell'azienda Leader.IT. Questo permetterá di essere piú veloci e sicuri quando si andrá a scrivere le parti di personalizzazione necessarie per rendere la soluzione operativa. La manutenzione e l'espansione saranno a loro volta agevolate e le licenze ne permetteranno una maggiore espansione.

Bibliografia

[1] Apache: http://www.apache.org home page del web server Apache. (02/05/2007)

[2] Tomcat: http://tomcat.apache.org home page del web server tomcat dell'Apache Software Foundation. (02/05/2007)

[3] Struts: http://struts.apache.org home page del web server MVC Jakarta Struts dell'Apache Software Foundation in collaborazione con il Jakarta Open Source Project. http://www.oracle.com/technology/products/jdev/collateral/papers/10g/Developing_Struts.pdf. (02/05/2007)

[4] Jakarta: http://jakarta.apache.org/ home page del Jakarta Open Source Project. (02/05/2007)

[5] GPL: http://www.gnu.org/copyleft/gpl.html sito ufficiale della licenza GPL (General Public License). (02/05/2007)

[6] AJAX: http://en.wikipedia.org/wiki/AJAX wikipedia spiega il paradigma "asincronous javascript and xml"
http://www.openajax.it sito che riporta alcuni esempi dell'uso di AJAX. (02/05/2007)

[7] JSON: http://www.json.org/xml.html introduzione a json un protocollo di interscambio dati leggero. (02/05/2007)

[8] XML: http://w3.org/XML/ sito ufficiale Exensible Markup Language http://en.wikipedia.org/wiki/XML wikipedia spiega l'XML. (02/05/2007)

[9] DOJO: http://dojotoolkit.org sito ufficiale del framework DOJO. (02/05/2007)

[10] DOJO Foundation: http://dojotoolkit.org/foundation/ parte del sito del DOJO Toolkit relativa alla fondazione. (02/05/2007)

[11] MVC: http://en.wikipedia.org/wiki/Model-view-controller wikipedia spiega il modello model-view-controller. (02/05/2007)

[12] HTML: http://www.w3.org/MarkUp/ sito dello standard W3C dell'HyperText Markup Language. (02/05/2007)

[13] XHTML: http://www.w3.org/TR/xhtml1/ sito dello standard W3C dell'Extensible HyperText Language. (02/05/2007)

[14] HTTP: http://www.w3.org/Protocols/ sito ufficiale del protocollo HTTP del W3C. (02/05/2007)

[15] W3C: http://www.w3.org sito ufficiale del World Wide Web Consortium. (02/05/2007)

[16] CGI: http://hoohoo.ncsa.uiuc.edu/cgi/ sito ufficiale delle Common Gateway Interface. (02/05/2007)

[17] Nexus: http://it.wikipedia.org/wiki/Nexus_%28browser%29 wikipedia racconta il browser Nexus. (02/05/2007)

[18] FTP: http://www.w3.org/Protocols/rfc959/ sito ufficiale del W3C sul File Transfer Protocol. (02/05/2007)

[19] IE: http://www.microsoft.com/windows/ie/default.mspx home page di Internet Explorer, browser proprietario di Microsoft. (02/05/2007)

[20] Firefox: http://www.mozilla.com/en-US/firefox/ home page del browser open-source di Mozilla. (02/05/2007)

[21] DOM: http://www.w3.org/DOM/ home page dello standard Document Object Model. (02/05/2007)

[22] JS: http://www.ecma-international.org/publications/standards/Ecma-262.htm sito ECMA dello standard. (02/05/2007)

[23] CSS: http://www.w3.org/Style/CSS/ home page dello standard Cascading Style Sheet. (02/05/2007)

[24] IIS: http://www.microsoft.com/windowsserver2003/iis/default.mspx home page dell'Internet Information Server. (02/05/2007)

[25] IPv6: protocollo che stabilisce un indirizzo ip in 6 byte anziché 4 come quello attualmente in uso.

[26] POSIX: Portable Operating System Interface, http://www.knosof.co.uk/posix.html. (02/05/2007)

[27] IIS Architecture http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/93ddbb51-5826-4ebd-a434-24c5fd103d3a.mspx?mfr=true. (02/05/2007)

[28] Apache Software Foundation http://www.apache.org. (02/05/2007)

[29] NCSA, National Center for Supercomputing Application http://www.ncsa.uiuc.edu/. (02/05/2007)

[30] Apache Structure www.apacheref.com/book/adr_ch2.pdf. (02/05/2007)

[31] Java http://www.sun.com home page della Sun Microsystem, inventrice e distributrice di Java e della JVM (Java Virtual Machine, interprete del byte-code java). (02/05/2007)

[32] ECMA http://www.ecma-international.org/ home page ufficiale della European Computer Manufacturer Association. (02/05/2007)

[33] ISO http://www.iso.org home page ufficiale dell'International Organization for Standardization. (02/05/2007)

[34] SOAP http://en.wikipedia.org/wiki/SOAP wikipedia spiega lo standard W3C(http://www.w3.org/TR/soap/) sul protocollo SOAP. (02/05/2007)

[35] JSP http://java.sun.com/products/jsp/ home page di Sun sulle Java Server Pages. (02/05/2007)

[36] SUN http://www.sun.com/ home page della Sun Microsystem. (02/05/2007)

[37] ASP http://www.asp.net/ home page della tecnologia delle Active Server Pages di Microsoft. (02/05/2007)

[38] Larry Wall creatore del linguaggio Perl.

[39] Perl http://www.perl.com sito ufficiale del Pratical Extraction and Report Language. (02/05/2007)

[40] CPAN http://www.cpan.org sito ufficiale del Comprehensive Perl Archive Network. (02/05/2007)

[41] Mason http://www.masonhq.com sito ufficiale. (02/05/2007)

[42] POE http://poe.perl.org/ sito ufficiale del framework Perl Object Enviroment sviluppato da Rocco Caputo http://poe.perl.org/?Rocco e slide sull'argomento http://axkit.org/docs/presentations/tpc2002/poe.axp/poe.pdf. (02/05/2007)

[43] Comet http://en.wikipedia.org/wiki/Comet_(programming) wikipedia spiega la tecnologia Comet. (02/05/2007)

[44] Flash http://www.cs.princeton.edu/ vivek/flash/ Web Server Event-Driven elaborato da Vivek S. Pai dell'universitá di Princeton, http://www.cs.princeton.edu/ vivek/flash_usenix _99/flash.pdf. (02/05/2007)

[45] Zeus http://www.zeus.com Web Server Event-Driven. (02/05/2007)

[46] J2EE http://it.wikipedia.org/wiki/J2EE wikipedia spiega le teconologie di J2EE. (02/05/2007)

[47] CATALYST http://search.cpan.org/ ash/ pagina CPAN della documentazione di del Catalyst Web Application Framework. http://www.catalystframework.org/ home page del framework Catalyst. http://search.cpan.org/ jrockway/Catalyst-Manual-5.700501/lib/Catalyst/Manual/Intro.pod introduzione alla programmazione Catalyst. (02/05/2007)

[48] Tarcisio Fedrizzo tesi svolta nell'anno accademico 2004/2005 e reperibile presso l'Universitá degli Studi di Trento, Facoltá di Scienze Matematiche Fisiche e Naturali, Povo.

[49] CMS http://it.wikipedia.org/wiki/Content_management_system wikipedia spiega il Content Management System. (02/05/2007)

[50] David Davis http://xantus.vox.com/ home page del maggiore sviluppatore di CometD. (02/05/2007)

[51] SVN http://subversion.tigris.org/ home page del sistema Subversion. (02/05/2007)

[52] Perlbal http://www.danga.com/perlbal/ home page del web server Perlbal. (02/05/2007)

[53] Bayeux http://svn.xantus.org/shortbus/trunk/bayeux/protocol.txt protocollo usato da David Davis per scambiare oggetti JSON con Cometd. (02/05/2007)