UNIVERSITÀ DEGLI STUDI DI TRENTO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Informatica (triennale)

Elaborato Finale

Sistemi per la Gestione dei Contenuti: analisi dello stato dell'arte e proposta di un modello architetturale basato su eventi

Relatore:
Prof. Maurizio Marchese

Laureando:
Tarcisio Fedrizzi

Correlatore:
Ing. Guido Brugnara

Anno Accademico 2004/2005

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Obiettivi della tesi

Il presente lavoro di tesi continua l'esperienza di tirocinio svoltasi presso la ditta Leader.IT. Tale azienda si occupa principalmente di:

Attività sistemistica: in questo campo la ditta effettua installazioni e manutenzione di server e postazioni. Si occupa inoltre della creazione di reti e della loro gestione.

Sviluppo software: Leader.IT progetta e sviluppa principalmente applicazioni web solitamente implementate utilizzando strumenti e sistemi Open Source.

Durante il tirocinio sono stato coinvolto in attività di ricerca e sviluppo riguardanti le tematiche dei CMS (Content Management System). Al termine del progetto di tirocinio si è quindi delineata la possibilità di proseguire l'attività con lo svolgimento di questa tesi.
L'obiettivo principale del lavoro di tesi è stato uno studio di fattibilità di una soluzione nel campo dei CMS.
In dettaglio si sono approfonditi i seguenti argomenti:

- l'analisi dello stato dell'arte dei CMS;
- l'analisi degli strumenti utilizzati;
- la definizione di una nuova architettura per CMS basata sugli eventi.

Le motivazioni principali che hanno portato alla proposta di tale nuova architettura sono:

- il superamento di alcuni limiti e l'introduzione di alcune innovazioni rispetto alle soluzioni esistenti ad oggi sul mercato;

- l'aumento del know-how dell'azienda nel dominio dei CMS come obiettivo strategico aziendale.

Il lavoro di tesi descritto nel seguito è così strutturato:

- nel capitolo 2 viene definito il concetto di CMS ed analizzati tre prodotti del dominio;
- nel capitolo 3 vengono presentati gli strumenti sui quali l'architettura verrà basata;
- nel capitolo 4 viene presentata la proposta di architettura;
- nel capitolo 5 vengono esposte le conclusioni e gli sviluppi futuri;
- in ultimo si chiude con la biografia.

Capitolo 2

Stato dell'arte dei CMS

2.1 Cosa sono?

L'acronimo CMS sta per "Content Management System" ovvero: "Sistema di Gestione dei Contenuti". In pratica un CMS è qualsiasi sistema progettato per organizzare e facilitare la creazione collaborativa di contenuti. Il CMS permette a degli autori di inserire testi solitamente in forma di articolo. L'inserimento generalmente è facilitato (ed anche arricchito di funzionalità) dall'utilizzo di un linguaggio di markup[1] il quale permette di descrivere il testo dal punto di vista strutturale lasciando il compito della formattazione al CMS stesso (in maniera simile a LaTeX 2_ε[Lat]) ottenendo così uno stile uniforme senza interventi manuali. Il CMS si occupa anche di gestire le situazioni di concorrenza (ad esempio quando due autori vogliono lavorare sullo stesso blocco di testo) garantendo la consistenza del contenuto o avvisando l'utente nel caso in cui non sia possibile agire automaticamente. Il termine CMS originariamente era riferito a dei sistemi per la pubblicazione di articoli (sia online che su carta); ora, con l'avvento del web, il significato di questo termine si è allargato verso quello di "programma per la gestione di siti web".

Esistono svariate tipologie di CMS create e definite intorno al tipo di contenuto che andranno a gestire. Alcune di queste sono:

Portale: questa tipologia di CMS serve per la gestione di portali i quali sono

[1] Il termine markup (o marcatura) deriva dall'ambiente tipografico dove veniva usato per definire le annotazioni fatte su una bozza, allo scopo di segnalare al compositore o al dattilografo il modo con cui alcune parti del testo andavano evidenziate o corrette. La tecnica di composizione del testo utilizzando marcatori o codici, richiede la definizione di una serie di convenzioni, ovvero di un linguaggio di markup. In generale questo descrive i meccanismi di strutturazione e di rappresentazione del testo che utilizzando convenzioni standardizzate sono utilizzabili su più sistemi.[Wik]

dei "raccoglitori" di informazioni molto eterogenee tra loro come news, tempo, cinema, ... (un esempio di portale è yahoo!);

Blog: il blog è un CMS a "contenuto personale": la controparte online di un diario;

Piattaforme E-Commerce: in questa tipologia il contenuto centrale è il commercio online; tramite questo CMS è possibile gestire un sito di commercio elettronico: in pratica un negozio online. Il gestore tramite un'apposita interfaccia può "inserire" i prodotti negli scaffali virtuali permettendo così agli interessati di acquistarli;

Groupware: i groupware sono dei software creati per gestire l'attività di collaborazione all'interno di gruppi, associazioni, ...; esso gestisce una serie di aspetti necessari quali: posta, calendario, invio di messaggi, ...;

Forum: i forum sono dei siti che permettono di discutere in maniera simile alle chat, con la particolarità che, in questo caso, i messaggi permangono nel tempo permettendo così la consultazione e la ricerca da parte più persone interessate;

Piattaforme e-Learning: sono applicazioni online create per gestire l'organizzazione di una struttura di istruzione, come un'università. Questi CMS rispecchiano l'organizzazione della struttura di istruzione fornendo ai professori una maniera per gestire i loro corsi, potendovi inserire materiale di studio, programmi di lezione, incontri, notizie a cui possono attingere gli studenti;

Image Gallery: questa tipologia di CMS è costruita attorno ad un particolare tipo di contenuto: le immagini. L'image gallery permette di catalogare le immagini secondo particolari criteri decisi dal gestore per poi agevolarne la visione da parte del pubblico;

Wiki: il wiki, contrariamente alle altre categorie di CMS, adotta un approccio piuttosto libero. Esso, infatti, è costruito in maniera tale da gestire qualsiasi tipologia di contenuto. Questa caratteristica è preziosa in alcuni casi: permette infatti di organizzare le informazioni secondo le proprie necessità. Il punto di forza costituito dalla libertà diventa però talvolta una debolezza, in quanto rende difficile produrre contenuti uniformi soprattutto se all'interno di un gruppo.

Come abbiamo visto esistono svariate applicazioni dei CMS. Talvolta si lascia molta libertà (vedi wiki) sacrificando per questo la semplicità di inserimento e tralasciando la generazione di meccanismi per facilitare la creazione di contenuti uniformi; in altri casi invece si pongono dei confini ben precisi per facilitare la gestione del contenuto e per ottenere un'omogeneità nelle informazioni.

Nel seguito verrà presentata l'analisi di tre CMS appartenenti ciascuno ad una diversa categoria:

- Plone
- Bricolage
- Twiki

2.2 Presentazione ed analisi di strumenti esistenti

2.2.1 Analisi di Plone

Plone[Plo] è un sistema per il content management scritto in Python[Pyt] pensato e implementato come strumento per creare e gestire portali. Questo prodotto si appoggia sul web application server Zope[Zop], scritto anch'esso in Python. Zope è una soluzione che include tutto il necessario per costruire una generica "applicazione web" (siti web, CMS, Blog, ...). Esso comprende un Server Web, un ODBMS (Object DataBase Management System) ed anche un sistema CMF (Content Management Framework) sul quale Plone si basa. Il CMS in descrizione è nato nel corso del 2001, alcuni anni dopo la creazione di Zope, con l'ambizioso obiettivo di divenire uno dei migliori e più completi sistemi CM del mondo. Plone ha diverse caratteristiche che lo rendono un buon progetto come il supporto built-in per l'internazionalizzazione, l'estensibilità, l'aderenza agli standard ed altro che verrà descritto più avanti. Tuttavia, essendo un progetto ancora piuttosto giovane, ha una serie di carenze da colmare.

Diverse caratteristiche positive contraddistinguono Plone:

Estrema semplicità di installazione: l'installazione è molto semplice e lineare ed è possibile avere un server funzionante con Plone installato in una lasso molto ristretto di tempo;

Multipiattaforma: Plone essendo scritto in un linguaggio interpretato, come il Python, può girare senza alcuna modifica sulle piattaforme per le quali esiste un interprete Python; tra queste: GNU/Linux, Windows, MacOS, Solaris, BSD;

Interfacciabilità a Web Server e DBMS: Plone è in grado di interfacciarsi a Webserver e DBMS diversi da quelli di Zope (che comunque non può essere eliminato essendo Plone basato su questo framework). Esiste infatti la possibilità di utilizzo in combinazione con tecnologie quali ad esempio Apache[Apa] e IIS[IIS] per la parte del webserver, e per quanto riguarda il backend esistono svariate opzioni talune commerciali come Oracle[Ora], MS SQL Server[MSS] e altre opensource PostgreSQL[Pos], MySQL[MyS], ...

Internazionalizzazione: Plone supporta l'internazionalizzazione e la sua interfaccia è tradotta in una gran quantità di lingue (circa 40); questa caratteristica determina delle possibilità di diffusione anche a livello mondiale non indifferenti;

Aderenza agli standard: il sistema CM Plone è stato progettato e implementato in maniera che i contenuti prodotti siano conformi alle specifiche dell'XHTML e che facciano uso di CSS (Cascading StyleSheet) per quanto riguarda la parte di presentazione. Questo garantisce la separazione dei contenuti dalla presentazione e quindi semplifica la gestione e la manutenzione del portale;

Interfacciabilità usando protocolli e formati standard: presenta possibilità di interfacciamento non indifferenti quali: RSS (RDF Site Summary, formato basato su XML per la distribuzione di contenuti), WebDAV (Web-based Distributed Authoring and Versioning, estensione del protocollo HTTP che rende il web un media scrivibile come era stato originariamente pensato da Tim Berners-Lee[BLCL+94]), XML-RPC (eXtensible Markup Language - Remote Procedure Call, sistema per l'invocazione di procedure remote basato su XML);

Accessibilità: possibilità di fruizione del web anche da parte delle persone disabili. La comunità Plone ha lavorato anche su questo ottenendo la certificazione AA del W3C (World Wide Web Consortium) per quanto riguarda l'accessibilità;

Possibilità di estensione: esistono parecchi Add-On, sia gratuiti che non, i quali svolgono altrettante funzioni. Questa caratteristica apre molte

strade permettendo a questo CMS di avere possibilità di utilizzo in campi differenti da quello dei portali;

Sicurezza: Plone utilizza le access list con diritti specificabili per utenti, gruppi e tutti. Inoltre è possibile creare una gerarchia tra gli utenti nella quale alcuni sono "gestori" e possono pubblicare mentre altri sono "scrittori" ed inviano i loro articoli ai gestori per la revisione ed eventuale pubblicazione;

Strumenti: Plone fornisce molti strumenti tra i quali: un sistema per il templating utilizzabile per definire template differenti da quello predefinito, un sistema per la definizione di nuovi tipi di contenuto e un sistema di workflow per la gestione della "vita" degli articoli.

Alcuni punti a sfavore di Plone sono invece:

Difficoltà di personalizzazione: Plone essendo progettato in maniera modulare offre molte possibilità per quanto riguarda la personalizzazione e l'espandibilità, tuttavia questo aspetto richiede delle conoscenze piuttosto buone di Zope (molto potente ma allo stesso tempo complesso), Python e della struttura di Plone stesso.
L'utente che vuole adattare questo prodotto alle proprie esigenze deve affrontare una ripida curva di apprendimento;

Carenza di adeguata documentazione: questo è un grande ostacolo per chiunque voglia modificare ed adattare Plone alle proprie esigenze. A causa di ciò la scelta potrebbe facilmente ricadere su altre soluzioni;

Scarse performance: Plone nell'installazione di base non è molto performante, è quindi necessario migliorarne le prestazioni con accorgimenti quali cache ed altro;

Necessario utilizzo dell'interfaccia di Zope: per utilizzare alcune funzionalità di amministrazione di Plone, tra cui alcune basilari, è necessario accedere all'interfaccia di gestione di Zope che è completamente separata; non essendo quindi completamente centralizzata la gestione diviene più difficile.

2.2.2 Analisi di Bricolage

Bricolage[Bri] è un prodotto molto più somigliante a Plone che a un wiki infatti esso può essere utilizzato per produrre siti di qualsiasi dimensione.

Questo CMS è scritto in Perl, tuttavia esso non è stato progettato per girare sotto i sistemi Windows. Bricolage utilizza come backend dati il DBMS PostgreSQL, mentre come webserver Apache (1.3) e mod_perl (un modulo per Apache che permette di utilizzare Perl nella configurazione e nelle pagine pubblicate). Come già accennato Bricolage gira soltanto su piattaforme Unix quali Linux, BSD, AIX, HPUX, Mac OS X, ... Il CMS si appoggia su un CMF (anche se è un po' improprio definirlo così infatti esso fornisce ad esempio ottimi strumenti per il templating ma non si pone come obiettivo di essere un CMF completo) chiamato Mason[Masa] il quale permette di suddividere logicamente le pagine in componenti ed assemblarle secondo i template disponibili.

Le caratteristiche che contraddistinguono Bricolage sono:

Usabilità: Bricolage ha un'interfaccia piuttosto intuitiva studiata per essere semplice e per permettere a coloro che lo utilizzano di inserire i propri contenuti senza conoscere nulla dei linguaggi che verranno utilizzati per pubblicare le notizie;

Utilizzabile tramite il browser: Bricolage è stato pensato per essere utilizzato da browser senza l'ausilio di alcun plugin, questo ne permette l'uso con una vasta gamma di browser (Internet Explorer, Opera, Mozilla, Firefox, Netscape) anche se non troppo aggiornati;

Supporta i Workflow: questo meccanismo permette di definire dei "percorsi" che i contenuti devono compiere. Questi definiscono, ad esempio, che un articolo deve passare attraverso un revisore prima di poter essere pubblicato. Il meccanismo dei workflow si rivela molto potente in quanto è possibile adattare il comportamento del CMS alle proprie esigenze a seconda dei campi in cui viene utilizzato;

Sicurezza: anche in questo CMS come negli altri presentati esiste un sistema per gestire la sicurezza: è infatti possibile assegnare e gestire i permessi per ciascun utente o gruppo; inoltre bricolage permette agli utenti di accedere tramite connessioni protette da SSH/TLS (Secure SHell/Transport Layer Security) così da poter avere una certa garanzia di sicurezza sulla connessione;

Gestione della versione: Bricolage ha un sistema per la gestione delle versioni dei contenuti: esso permette di aggiornare e tenere traccia delle modifiche apportate ai documenti potendo così annullare eventuali errori;

Modellazione documenti: questa feature di Bricolage è molto importante: permette infatti di creare dei modelli che i contenuti inseriti dovranno rispettare permettendo così la definizione di uno stile uniforme con il quale verranno presentati. I contenuti hanno una struttura ad albero costituita da elementi. Questi ultimi sono dei "contenitori" il cui compito è la suddivisione logica delle informazioni. Essi possono ospitare dei sottoelementi composti dai campi nei quali verranno inseriti i dati (ad esempio text fields, textarea, ...);

Separazione contenuto presentazione: anche in questo caso molta enfasi viene data a questo aspetto. Bricolage in particolare è uno strumento che si focalizza sulla gestione del contenuto demandando la parte di presentazione ad altri software (il già citato Mason ad esempio). Infatti Bricolage prevede la gestione di quelli che vengono chiamati "Output Channel" che gli permettono di presentare il contenuto nei formati più svariati a seconda delle proprie necessità;

Template: in Bricolage la forma dei contenuti è specificata attraverso i modelli di documento, essi descrivono anche la struttura dei widget per l'inserimento. Questo permette di mantenere la separazione vista poc'anzi. Il CMS fornisce un motore per il templating che, attraverso gli Output Channel, è in grado di dare la struttura desiderata al contenuto e di produrre in output il formato desiderato;

Categorizzazione dei contenuti: in bricolage è possibile definire delle categorie entro le quali verranno inseriti i contenuti. Le categorie sono gerarchiche (ad esempio è possibile definire una gerarchia quale: Animali $\rightarrow$ Mammiferi $\rightarrow$ Gatti) e corrispondono alla struttura delle cartelle del sito prodotto, è così possibile catalogare le informazioni in maniera ordinata. Questa possibilità è utile anche nel caso di indicizzazione e ricerca tramite un motore; si è così in grado di estrarre le informazioni sulla base della loro categoria di appartenenza;

Gestione siti multipli: il sistema CM che stiamo analizzando permette di gestire tramite una sola installazione siti differenti. Questa possibilità è molto utile soprattutto in contesti enterprise, permette infatti di mantenere separati i vari siti condividendo però caratteristiche tra di loro e potendoli gestire da un'unica interfaccia;

Interoperabilità: Bricolage è stato costruito anche con una attenzione particolare rispetto all'interoperabilità, è infatti possibile interagire con questo CMS anche attraverso SOAP (Simple Object Access Protocol).

Anche l'utilizzo di questo protocollo può essere protetto attraverso l'uso di tecniche di crittografia;

Scalabilità: questo prodotto fornisce possibilità di configurazione a cluster per ottenere una scalabilità molto elevata.

Esistono alcune problematiche piuttosto comuni tra CMS così vasti. Tra queste:

Complessità: pur essendo studiato per essere semplice da utilizzare Bricolage è un prodotto abbastanza complesso, ciò rende l'apprendimento più lungo e difficoltoso;

Requisiti: la vastità del prodotto oltre a pesare sulla difficoltà di apprendimento grava anche sui requisiti per il funzionamento e quindi sulle performance in quegli ambienti dove non è possibile avere hardware veloce;

Incompatibilità Windows: altro punto a sfavore di questo prodotto, è l'impossibilità di utilizzo in accoppiata con Windows ed IIS. Ciò gioca sicuramante a sfavore di questa soluzione, anche se una gran parte dei server web montano sistemi Unix e Apache.

2.2.3 Analisi di TWiki

TWiki[TWi] é un tipo di CMS che differisce profondamente dai due presentati precedentemente. Infatti, Bricolage e Plone sono stati creati, come abbiamo già visto, per gestire in maniera abbastanza schematica e predefinita un genere di contenuti ben preciso mentre un wiki, è una tipologia di CMS che permette di creare e gestire contenuti in maniera molto più aperta. La modalità in cui un wiki gestisce i contenuti è molto simile a quella di LaTeX 2_ε nella quale il testo viene inserito corredato di markup che, come spiegato precedentemente, è utile per dare un significato strutturale al testo permettendo all'applicazione (il wiki) di eseguire la formattazione secondo un particolare stile (separazione contenuto presentazione).
Il sistema CM TWiki è scritto in Perl che è un linguaggio interpretato, quindi molto portabile, grazie a ciò TWiki gira su parecchie piattaforme (Unix, Linux, FreeBSD, OpenBSD, NetBSD, HP-UX, Windows (2000, 2003, XP), Solaris, AIX). Per quanto riguarda la parte di WebServer TWiki può essere installato ed utilizzato sia su Apache che su IIS. Il supporto sul quale vengono scritti i dati inseriti in TWiki in questo caso sono dei file, anche se è possibile attraverso un plugin fare sì che TWiki scriva i suoi dati su DB.

TWiki è un CMS molto elaborato e vasto (esiste da 7 anni, prima release nel 1998) e possiede quindi una grande quantità di caratteristiche interessanti:

Controllo revisioni: permette di tracciare le varie modifiche apportate alle pagine, compresa data, ora ed autore; il sistema inoltre è in grado di mostrare le modifiche apportate alle varie pagine nel formato diff di UNIX;

Visibile da qualiasi browser: TWiki non fa uso di tecnologie particolari (Flash, Java, ...) quindi sul client è molto leggero ed ha dei requisiti (sempre lato client) molto bassi;

Supporto internazionalizzazione: TWiki ha un supporto per l'internazionalizzazione, tuttavia esso è in fase di miglioramento ed è quindi attualmente abbastanza limitato;

Permessi utente: anche TWiki presenta il classico schema per la protezione in stile UNIX con permessi di lettura/scrittura per utenti, gruppi e altri; i permessi sono definibili in maniera granulare e precisa sui vari contenuti inseriti;

Generazione dinamica di contenuti: attraverso delle direttive inseribili all'interno del testo, che verrà poi formattato dal motore di TWiki, è possibile creare dei contenuti generati dinamicamente (per esempio un indice o una tavola dei contenuti (TOC));

Non c'è necessità di database: come spiegato prima TWiki salva i contenuti sul filesystem. Questo fattore è allo stesso tempo un pregio e un difetto, infatti, questo permette di utilizzare TWiki anche su macchine non molto aggiornate, che farebbero fatica a gestire un DBMS; tuttavia spesso un quest'ultimo permette di abbassare i tempi di risposta grazie alle ottimizzazioni delle query, cosa che non è possibile o risulta molto complessa a livello di semplice file;

Plugin: TWiki ha la possibilità di fare uso di plugin, i quali aggiungono funzionalità come ad esempio grafici, calendari, il supporto a DBMS accennato prima e molte altre (al momento attuale, 07.06.2005, ne sono listati 163). Questa opzione permette a TWiki di rimanere sempre aggiornato restando al passo coi tempi e le tecnologie;

Ricerca: il fatto che gli URL (Uniform Resource Locator) dei contenuti di TWiki non contengano la parte di query semplifica l'indicizzazione di eventuali motori di ricerca, inoltre TWiki integra un motore di ricerca

interno piuttosto completo: è infatti possibile fornire una serie di parametri, tra cui le sezioni in cui cercare, fare uso di espressioni regolari ed altro;

Template e skin: TWiki offre la possibilità di creare dei nuovi template in maniera da poter riorganizzare la visualizzazione dei contenuti secondo le proprie necessità, inoltre esistono anche le skin che permettono la modifica di header e footer del template;

Topic Locking: un problema sempre presente nelle applicazioni collaborative riguarda il controllo di concorrenza il quale può essere gestito in molte maniere. TWiki lo risolve effettuando un locking delle pagine in stato di modifica ovvero impedendo l'accesso ad esse;

Standard: TWiki non sembra porre il rispetto degli standard come uno dei suoi principali obiettivi, tuttavia il template di default è scritto in maniera da essere XHTML 1.0 Transitional Compliant;

Statistiche: altro strumento utile fornito da TWiki è quello delle statistiche il quale permette di verificare, in maniera molto semplice, gli utenti più attivi, i topic più visti, ... Questi potrebbero essere molto utili all'interno di un gruppo.

Anche TWiki, pur essendo un prodotto valido e nonostante abbia alle spalle 7 anni di sviluppo, soffre di alcune carenze:

"Troppa libertà": l'ampia libertà concessa nell'inserimento dei contenuti è un'arma a doppio taglio: infatti se da un lato questo è un vantaggio, da un altro punto di vista rappresenta un problema in quanto causa una grande difficoltà nella creazione di contenuti uniformi. Con questa libertà si ottiene quindi qualcosa di difficilmente gestibile (soprattutto se utilizzato da grandi gruppi di persone);

Contenuti non strutturati: in TWiki è possibile creare delle sezioni in cui mettere contenuti differenti. Questa è una buona possibilità tuttavia la suddivisione si ferma qui, nel senso che i contenuti creati all'interno di questi gruppi potrebbero essere, a loro volta, suddivisi ed organizzati in sottocartelle in maniera da renderli meglio gestibili. I contenuti possono, quindi, essere strutturati manualmente (all'interno della pagina) da un punto di vista logico senza che TWiki fornisca un supporto per questo;

Poco intuitivo: l'uso di questo programma può presentarsi difficile. Infatti, non disponendo di un'interfaccia di tipo WYSIWYG (What You See

Is What You Get), TWiki, impedisce a chi non conosce il linguaggio utilizzato un approccio immediato;

Lentezza: un altro problema, notato anche durante l'esperienza personale avuta nella ditta presso la quale ho svolto lo stage, è la lentezza di questo prodotto. Questo può essere dovuto alla grande complessità della soluzione ed anche al fatto che tutti i contenuti risiedono su file il che potrebbe portare su sistemi non troppo veloci a tempi di risposta molto lunghi (dovuti, ad esempio, alla lentezza del controller del disco).

2.2.4 Analisi e conclusioni

A seguito di queste analisi posso affermare che vi è una tendenza di crescita in una direzione particolare ovvero l'orientamento è quello di generalizzarsi sempre più per gestire una classe più ampia possibile di contenuti. Infatti i prodotti presentati, pur essendo nati con l'intenzione di gestire particolari tipologie di contenuti, presentano molte caratteristiche in comune. I prodotti manifestano questa propensione attraverso due approcci:

Limitazione della libertà: è abbracciato da Plone e Bricolage e permette l'inserimento di nuovi contenuti solo a seguito di una descrizione;

Libertà massima: viene invece seguito da TWiki e permette all'utente di inserire i più svariati contenuti come crede.

In sostanza, i primi due tendono ad essere maggiormente rigidi per ottenere contenuti più gestibili rinunciando, di conseguenza, ad un po' di libertà, TWiki invece permette massima (o quasi) libertà determinando, tuttavia, maggiore difficoltà nella gestione.
Io credo che la tendenza all'uniformazione sia destinata a proseguire, tuttavia una conseguenza indesiderata di ciò è l'aumento delle dimensioni dei CMS che rischiano di crollare sotto il loro stesso peso. Questo svantaggio esiste nelle soluzioni analizzate che purtroppo manifestano talvolta qualche difficoltà. In conclusione ritengo che sia giusto seguire la strada già tracciata dalle soluzioni presenti oggi cercando di mantenere la leggerezza (uniformando il più possibile la gestione di qualsiasi tipologia di informazione) ed il rigore nella classificazione dei contenuti rendendo però più semplice la modifica degli stessi.

Riporto infine una serie di tabelle comparative e riepilogative per i tre CMS presentati tratte da http://www.cmsmatrix.org/[CMS]:

Requisiti di sistema	Bricolage	Plone	TWiki
Application Server	mod_perl	Zope	(opzionale)
Costo approssimativo	Open-Source (Gratuito)	Free	Open-Source (Gratuito)
Database	PostgreSQL	Zope	File
Licenza	BSD (Modificata)	GNU GPL	GNU GPL
Sistema operativo	Unix (Linux, BSD, AIX, HPUX, Mac OS X, etc.)	Qualsiasi	Unix, Linux, FreeBSD & All BSD's, HP-UX, Windows 2000, 2003, XP, Solaris, AIX
Linguaggio	Perl 5.6+	Python	Perl 5.6+
Necessario root	N/A	No	Sì
Installabile da shell	N/A	No	Sì
Web Server	Apache	Apache, IIS, Zope	Apache, IIS

Tabella 2.1: Bricolage, Plone, TWiki: Requisiti

Sicurezza	Bricolage	Plone	TWiki
Tracciamento modifiche	Sì	Sì	No
Sistema anti-bot (Captcha)	N/A	No	No
Approvazione dei contenuti	Sì	Sì	No
Verifica e-mail	N/A	Limitato	No
Privilegi granulari	Sì	Sì	Sì
Autenticazione Kerberos	No	Add-On gratuito	No
Autenticazione LDAP	No	Add-On gratuito	Add-On gratuito
Storico dei login	No	Add-On gratuito	No
Autenticazione NIS	No	Add-On gratuito	No
Autenticazione NTLM	No	Sì	Add-On gratuito
Plug-in di autenticazione	No	Sì	Add-On gratuito
Notifica problemi	No	No	No
Sandbox	Sì	Sì	Sì
Gestione sessioni	No	Add-On gratuito	No
Autenticazione SMB	No	Add-On gratuito	No
SSL compatibile	N/A	Sì	Sì
Login SSL	N/A	No	No
Pagine SSL	N/A	No	No
Gestione versioni	Sì	Sì	Sì

Tabella 2.2: Bricolage, Plone, TWiki: Sicurezza

Supporto	Bricolage	Plone	TWiki
Programma di certificazione	No	No	No
Manuali commerciali	No	Sì	No
Supporto commmerciale	Sì	Sì	No
Corsi commerciali	Sì	Sì	No
Comunità degli sviluppatori	Sì	Sì	Sì
Aiuto online	Sì	No	No
API documentate	Sì	Sì	Sì
Hosting Professionale	Sì	Sì	No
Servizi Professionali	Sì	Sì	No
Forum pubblico	No	Sì	Sì
Mailing List pubblica	Sì	Sì	Sì
Sviluppatori esterni	Sì	Sì	Sì
Conferenze utenti	N/A	Sì	No

Tabella 2.3: Bricolage, Plone, TWiki: Supporto

Facilità d'uso	Bricolage	Plone	TWiki
Contenuti trascinabili	No	Add-On gratuito	No
Email nelle discussioni	No	Add-On gratuito	No
URL amichevoli	Sì	Sì	Limitato
Ridimensionamento immagini	N/A	Add-On gratuito	No
Linguaggio Macro	Sì	Sì	No
Upload di massa	N/A	Sì	No
Prototipizzazione	N/A	No	No
Linguaggio Server Page	Sì	Sì	No
Correttore automatico	N/A	Add-On gratuito	No
Sottoscrizioni	N/A	No	No
Linguaggio di templating	Sì	Sì	Limitato
Customizzazione UI	Sì	Sì	Limitato
Undo	Sì	Sì	No
WYSIWYG Editor	Sì	Sì	Add-On gratuito

Tabella 2.4: Facilità d'uso

Performance	Bricolage	Plone	TWiki
Caching avanzato	No	Sì	No
Replicazione Database	N/A	A pagamento	No
Bilanciamento del carico	No	Sì	No
Caching delle pagine	No	Sì	No
Esportazione dei contenuti statici	N/A	Add-On gratuito	No

Tabella 2.5: Bricolage, Plone, TWiki: Performance

Gestione	Bricolage	Plone	TWiki
Gestione pubblicità	No	Add-On gratuito	No
Gestione Asset	Sì	Sì	Sì
Clipboard	No	Sì	No
Schedulazione Contenuti	Sì	Sì	No
Condivisione Contenuti	Sì	Add-On gratuito	No
Amministrazione Inline	No	Sì	Sì
Amministrazione Online	Sì	Sì	Limitato
Pubblicazione package	No	Sì	No
Sotto-siti	Sì	Sì	No
Temi / Skins	Sì	Sì	Sì
Cestino	Sì	Add-On gratuito	Sì
Statistiche Web	No	Add-On gratuito	Sì
Gestione Stili/-Template Web-based	Sì	Sì	Limitato
Gestione della Traduzione Web-based	No	Add-On gratuito	No
Motore di Workflow	Sì	Sì	Limitato

Tabella 2.6: Bricolage, Plone, TWiki: Gestione

Interoperabilità	Bricolage	Plone	TWiki
Content Syndication (RSS)	Sì	Sì	Sì
Supporto FTP	Sì	Sì	No
Supporto UTF-8	N/A	Sì	Sì
Conformità WAI	No	Sì	No
Supporto WebDAV	No	Sì	Add-On gratuito
Conformità XHTML	No	Sì	No

Tabella 2.7: Bricolage, Plone, TWiki: Interoperabilità

Flessibilità	Bricolage	Plone	TWiki
Supporto CGI-mode	No	Add-On gratuito	Sì
Riutilizzo del contenuto	Sì	Sì	No
Profili utente estensibili	No	Sì	Sì
Localizzazione interfaccia	Sì	Sì	Limitato
Metadati	N/A	Sì	Sì
Contenuto multilingue	N/A	Add-On gratuito	Sì
Pubblicazione automatica contenuti multilingue	N/A	Add-On gratuito	No
Pubblicazione di siti multipli	Sì	Sì	No
Accorciamento URL	Sì	Sì	Sì
Dotato di Wiki	N/A	Add-On gratuito	Sì

Tabella 2.8: Bricolage, Plone, TWiki: Flessibilità

Applicazioni Built-in	Bricolage	Plone	TWiki
Blog	No	Sì	Add-On gratuito
Chat	No	Add-On gratuito	No
Classifieds	No	No	No
Rubrica	Sì	Add-On gratuito	No
Inserimento dati	No	Add-On gratuito	Sì
Rapporti Database	No	Limitato	Sì
Forum / Discussioni	No	Sì	No
Gestione Documenti	Sì	Sì	Sì
Calendario eventi	No	Sì	Add-On gratuito
Rapporti spese	No	No	No
Gestione FAQ	No	Add-On gratuito	No
Distribuzione file	Sì	Sì	Sì
Grafici e tabelle	N/A	No	Add-On gratuito
Groupware	No	Add-On gratuito	No
GuestBook	No	Add-On gratuito	No
Aiuto online / Gestione Bug	No	Add-On gratuito	No
HTTP Proxy	Sì	No	Sì
Stato utenti	N/A	No	No

Tabella 2.9: Bricolage, Plone, TWiki: Applicazioni built-in parte 1

Applicazioni Built-in	Bricolage	Plone	TWiki
Liste di collocamento	No	No	No
Gestione link	No	Add-On gratuito	No
Mail Form	No	Add-On gratuito	No
My Page / Dashboard	No	Limitato	Limitato
Newsletter	N/A	Add-On gratuito	No
Galleria fotografica	No	Add-On gratuito	Add-On gratuito
Questrionario	No	Add-On gratuito	Limitato
Gestione prodotti	No	Sì	Limitato
Tracciamento progetti	No	Add-On gratuito	Add-On gratuito
Motore di ricerca	No	Sì	Limitato
Mappa del sito	N/A	Add-On gratuito	No
Indagini	No	A pagamento	No
Syndicated Content (RSS)	No	Add-On gratuito	Sì
Test / Quiz	No	Add-On gratuito	No
Tracciamento tempo	No	No	Add-On gratuito
Contribuzione utenti	No	Sì	Limitato
Accesso tramite Web Services	Sì	No	No

Tabella 2.10: Bricolage, Plone, TWiki: Applicazioni built-in parte 2

Commercio	Bricolage	Plone	TWiki
Tracciamento affiliati	No	No	No
Gestione inventario	N/A	No	No
Plug-in pagamento	N/A	No	No
Plug-in consegna	N/A	No	No
Plug-in tasse	N/A	No	No
Punto di vendita	N/A	No	No
Carrello acquisti	No	Add-On gratuito	No
Sottoscrizioni	N/A	No	No
Lista desideri	N/A	No	No

Tabella 2.11: Bricolage, Plone, TWiki: Commercio

Capitolo 3

Strumenti per CMS

In questo capitolo verranno presentati gli strumenti (intendendo con il termine "strumenti": applicazioni, protocolli, formati, supporti) sui quali si baserà la soluzione proposta in questa tesi ed altri che verranno analizzati per trovare quelli che più si avvicinano alle necessità del progetto.
Questa parte è suddivisa in sezioni che raccolgono strumenti affini tra loro:

- Backend Dati
- Linguaggi
- Tecnologie
- Protocolli

3.1 Backend Dati

La scelta del backend sul quale salvare i dati è una delle fasi più importanti ai fini del progetto. Infatti il repository deve soddisfare una serie di requisiti chiave che garantiscono altrettante caratteristiche necessarie all'applicazione. In particolare i requisiti da rispettare sono:

robustezza: il prodotto deve garantire una buona stabilità, i guasti dovrebbero essere rari, il data repository dovrebbe essere in grado di gestire correttamente anche situazioni non previste;

scalabilità: questa caratteristica è necessaria in quanto il risultato della tesi dovrebbe essere la progettazione di un'ambiente valido per aziende sia piccole che grandi, quindi le prestazioni devono rimanere accettabili;

velocità: uno degli obiettivi del progetto dovrebbe essere proprio questo: abbassare i tempi di risposta canonici delle applicazioni web; un backend efficiente può quindi influire parecchio sulle prestazioni generali del sistema;

sicurezza: questa è una caratteristica molto importante. La sua centralità è confermata da una recente legge che prevede misure di gestione dei dati molto rigide e severe. Inoltre, il fatto di avere come obiettivo le aziende, rende necessaria la possibilità di definire dei permessi sui dati plasmati a seconda della funzione svolta dal dipendente.

Sono state individuate alcune possibili soluzioni:

- DBMS
- LDAP
- FileSystem
- RDF Database

ciascuna verrà approfondita in una sezione apposita.

3.1.1 DBMS

Descrizione

I DBMS, comunemente definiti database, sono i "contenitori", specificamente concepiti ed ottimizzati per contenere dati di tipo eterogeneo, più utilizzati. Essi permettono di eseguire operazioni di inserimento, ricerca, cancellazione, ...in maniera molto efficiente; le ottimizzazioni studiate per aumentare la velocità di questi repository dati sono moltissime. Esistono diversi modelli dati nel campo dei database. Quello relazionale è senz'altro uno dei più utilizzati: il suo studio ha radici negli anni '60 e culmina con il rilascio del documento prodotto dal Dr. Edgar F. Codd[Cod70] nel 1970 (intitolato: "A Relational Model of Data for Large Shared Data Banks") in cui il modello viene descritto. Questo ha il grande pregio di essere slegato dall'implementazione infatti tutto è rappresentato sotto forma di dati compresi, ad esempio, i collegamenti tra tabelle (in questo caso le relazioni). La storia di SQL nasce quando IBM sviluppa nel 1970 un linguaggio chiamato SEQUEL per l'accesso ad un suo prodotto denominato System R (un database relazionale scritto per scopi di ricerca). Nel 1979 la Relational Software, Inc. (ora divenuta Oracle) implementa la prima soluzione commerciale SQL. Da allora vennero sviluppati molti dialetti del linguaggio fino a che nel 1986 ANSI e nel 1987 ISO decisero di creare uno standard chiamato appunto SQL.

Linguaggio

Il linguaggio per l'interazione con l'RDBMS è SQL (Structured Query Language) il cui nome non ne evidenzia tutte le potenzialità. Infatti questo è molto di più che un solo linguaggio di interrogazione, esso permette la totale amministrazione dell'RDBMS ovvero la gestione delle tabelle, degli utenti, la configurazione del DBMS, ...
Esistono parecchie implementazioni di questo linguaggio e purtroppo (o per fortuna a seconda dei punti di vista) ogni vendor introduce delle caratteristiche che rendono i software, che utilizzano parti proprietarie del linguaggio, non portabili.

Permessi

Per quanto riguarda la gestione dei permessi il modello di SQL è piuttosto completo: sia il linguaggio che le soluzioni SQL presenti sul mercato gestiscono in maniera totale utenti e permessi. Tuttavia spesso il modello dei permessi integrato nei sistemi DBMS non copre interamente la casistica prevista a livello progettuale, quindi nella maggior parte dei casi si opta per realizzare un motore di gestione dei permessi integrato con il prodotto in sviluppo. Questo approccio però introduce un ulteriore strato di complessità al software in progettazione e limita l'interoperabilità nativa della base dati che necessita dello strato per la gestione dei permessi per poter funzionare secondo progetto.

Modello dati

Il modello dati più utilizzato correntemente è il modello relazionale: esso sfrutta il concetto matematico di relazione per rappresentare i dati e potervi svolgere delle operazioni logiche che permettono di trovare i dati richiesti.
Un altro modello esistente e tuttora in studio (anche se esistono diverse implementazioni) è quello ad oggetti il quale risulterebbe comodo nel caso del progetto preso in esame per la tesi. Infatti, essendo obiettivo di questo progetto creare una libreria che mappi degli oggetti da una base dati ad una serie di form su web, l'approccio ad oggetti sarebbe l'ideale, tuttavia è un campo ancora in fase di sviluppo e i prodotti attualmente esistenti sono ancora giovani e quindi non troppo affidabili.

Pro e contro

Pro:

- le soluzioni RDBMS hanno raggiunto una grande stabilità (intendendo con ciò prodotti esistenti da parecchio tempo);
- il modello relazionale è ormai consolidato e maturo;
- è uno standard.

Contro:

- il modello dei permessi fornito dagli RDBMS è rigido rispetto alle esigenze di progetto.

Possibili soluzioni tecnologiche

- PostgreSQL
- MySQL

3.1.2 LDAP

Descrizione

L'acronimo LDAP sta per "Lightweight Directory Access Protocol" ed è un protocollo per l'accesso ad un servizio di directory definito nell'RFC 3377 ed in altri otto RFC (elencati nell'RFC 3377). LDAP è un protocollo aperto pensato come semplificazione dello standard X.500 che è molto complesso e soprattutto basato sulla pila di protocolli OSI tuttora utilizzata soltanto per scopi didattici. Questo standard si basa sullo stack TCP/IP (Transmission Control Protocol/Interne Protocol)ed utilizza connessioni TCP, anche se sono disponibili implementazioni basate su UDP (User Datagram Protocol). Lo scopo di LDAP è quello di eliminare parti poco utilizzate di X.500 per semplificarne l'implementazione, l'utilizzo e per slegarlo da OSI (Open System Interconnection).
Un servizio di directory è una base dati disegnata e ottimizzata per fornire un rapido accesso in lettura e nelle ricerche. Le informazioni sono inserite in maniera gerarchica, come in un albero, all'interno di quelle che vengono chiamate directory; il tipo di informazioni che vengono inserite possono essere le più eterogenee. All'interno di un'azienda, ad esempio, il repository LDAP può contenere i dati dei dipendenti e fornire loro l'autenticazione ai vari servizi necessari per le loro attività. Proprio per il fatto che è molto utilizzato, soprattutto come contenitore di informazioni in ambito aziendale, LDAP si presenta come un possibile candidato tra i repository.

Linguaggio

Lo standard di LDAP non fornisce le specifiche per un linguaggio di accesso, le uniche descrizioni incluse riguardano: il formato in cui specificare nuovi attributi, quello per filtrare le ricerche e la struttura dell'URL tramite il quale è possibile interagire con il repository. Solitamente per accedere a backend LDAP si utilizzano delle librerie per il linguaggio scelto (ad esempio nel caso di Perl è possibile utilizzare l'implementazione Mozilla::LDAP::API che è un'interfaccia per chiamare la libreria C che interagisce con LDAP).

Permessi

Il modello di permessi di LDAP fa utilizzo delle ACL (Access Control List) tramite le quali vengono specificati permessi sugli oggetti contenuti nel repository. I permessi sono specificabili a livello di utenti, gruppi e tutti come nei sistemi Unix. Lo standard LDAP non specifica i tipi di permessi ma fornisce soltanto delle linee guida da seguire nell'implementazione di un directory server.

Modello dati

Il modello dati LDAP è strutturato ad albero ed i dati sono raggiungibili in maniera simile ad un filesystem. Per navigare l'albero è necessario specificare il percorso dalla radice fino al nodo di destinazione. In questa maniera è possibile identificare qualsiasi informazione in maniera non ambigua.

Pro e contro

Pro:

- utilizzato e quindi già presente in molti ambienti (scuole, università, aziende);
- è un protocollo standard.

Contro:

- è ottimizzato per la lettura, quindi le performance in scrittura potrebbero essere scadenti.

Possibili soluzioni tecnologiche

- OpenLDAP[Ope] open source, scrittura su BD (Brekley DB);
- Fedora Directory Server[Fed] ex Netscape Directory Server da poco rilasciato con licenza GPL.

3.1.3 Filesystem

Descrizione

Il filesystem è senza ombra di dubbio, tra quelli presentati, il backend dati più stabile ed affidabile (concettualmente), infatti l'idea di filesystem esiste da molto. Un filesystem sostanzialmente è una struttura dati ad albero nella quale i nodi sono le cartelle (contenitori di file) e le foglie sono i file (contenitori di dati). Esistono diverse tipologie di filesystem ognuno progettato per essere impiegato in un diverso settore: si va dalla catalogazione dei dati su un disco (classico filesystem), al mantenimento delle informazioni su un sistema (il procfs di GNU/Linux), all'accesso ad un disco remoto (filesystem di rete, per esempio NFS (Network FileSystem) di Sun) e molto altro.

Linguaggio

Il filesystem non possiede un linguaggio di interrogazione proprio, infatti è utilizzabile attraverso API disponibili pressochè in ogni linguaggio di programmazione.

Permessi

I permessi, nel caso del filesystem, sono specificabili tramite il classico modello UNIX: utente, gruppo, altri e comprendono i permessi di lettura, scrittura ed esecuzione. Alcuni filesystem implementano modelli più complessi in grado di gestire liste di utenti e di gruppi per ogni oggetto residente nel FileSystem: questa tipologia di permessi viene definita ACL (Access Control List).

Modello dati

Il modello dati di un filesystem ha una struttura ad albero nel quale sono inseriti i file che contengono i dati. Ogni dato è raggiungibile (diritti permettendo) in maniera univoca specificando il percorso dalla radice al dato che si intende richiedere.

Pro e contro

Pro:

- è un modello molto utilizzato, quindi anche molto testato;
- le performance nel recupero dei dati sono solitamente molto buone.

Contro:

- le performance in ricerche di una certa complessità sono minori rispetto a DBMS;
- è un modello poco versatile per l'inserimento di informazioni complesse.

Possibili implementazioni

ext, ext2, ext3: Extended FileSystem ovvero il filesystem di linux;

FAT: File Allocation Table, il filesystem di MS-DOS ancora molto utilizzato ad esempio nelle unità di memoria flash;

JFS: Journalling FileSystem progettato da IBM;

NTFS: New Technology FileSystem il filesystem correntemente utilizzato dai sistemi Microsoft;

ReiserFS: FileSytem progettato da Hans Reiser;

Reiser4: nuova versione del filesystem di Reiser;

XFS: filesystem progettato da SGI;

FUSE: filesystem in userspace, questa implementazione fornisce un'interfaccia per creare un filesystem nell'userspace;

3.1.4 RDF Database

Descrizione

RDF (Resource Description Format) è uno standard elaborato dal W3C. Esso costituisce uno dei mattoni su cui si dovrebbe basare il semantic web. RDF è stato progettato per rappresentare i metadati necessari a descrivere risorse ed utilizza la notazione N3 per rappresentare i dati in forma di soggetto, predicato ed oggetto. Nel caso specifico il soggetto è la particolare risorsa da descrivere, il predicato è la caratteristica che dev'essere riportata ed infine l'oggetto è il valore assunto dal predicato. Questa notazione potrebbe sembrare a prima vista incompleta permette tuttavia di rappresentare grafi di informazioni arbitrariamente grandi.
Vista la versatilità di questo linguaggio, di cui esiste anche una versione basata su XML ovvero RDF/XML, lo stesso viene riproposto (ed anche utilizzato ad esempio da Mozilla per il salvataggio di informazioni) come repository

dati interrogabile in maniera simile ai database. Esistono diversi linguaggi di interrogazione per basi di dati RDF che sono ancora allo stato di proposta, come ad esempio: RDQL o SPARQL. Nonostante RDF sia un linguaggio piuttosto giovane è già utilizzato ed esistono diverse implementazioni di base di dati RDF comprensive di parecchie utilità. Per quanto riguarda le basi dati RDF vi sono diverse possibilità; tra le più complete: l'implementazione di RedLand e di OpenRDF.

Linguaggio

Per quanto riguarda il linguaggio per l'interazione con basi di dati RDF esistono vari candidati in quanto il W3C non ha ancora adottato uno standard. Tra i candidati spiccano:

RDQL[RDQ]: proposto da parte di Hewlett Packard[HP] come possibile standard per l'interrogazione delle basi dati RDF;

SPARQL: proposto sempre da Hewlett Packard insieme al W3C come linguaggio standard per l'interrogazione delle basi dati RDF;

SeRQL: è il linguaggio ufficiale per il database RDF Sesame.

Tutti e tre si basano sui linguaggi di interrogazione già esistenti (come SQL), tuttavia avendo i database strutture concettuali totalmente differenti (SQL $\rightarrow$ Relazionale, RDF $\rightarrow$ N3) è necessario adattare il linguaggio alla diversa organizzazione dei dati.

Permessi

Essendo i database RDF ancora in fase di introduzione per ora le implementazioni si limitano a fornire la base dati e il/i linguaggio/i di interrogazione. Il supporto per i permessi è assente nelle implementazioni analizzate e verrà probabilmente introdotto solo quando le implementazioni diverranno standard e, quindi, largamente utilizzate.

Modello dati

Il modello dati di RDF si basa sul concetto di NTripla: questa è costituita da tre blocchi fondamentali: un soggetto, un predicato ed un oggetto, che, collegati tra loro, sono in grado di descrivere una risorsa. Il soggetto è la risorsa che dev'essere descritta, il predicato è la proprietà che si vuole esprimere e l'oggetto corrisponde al valore della proprietà. Ad esempio l'immagine 3.1(tratta da [RDF]) afferma che l'utente descritto dalla risor-

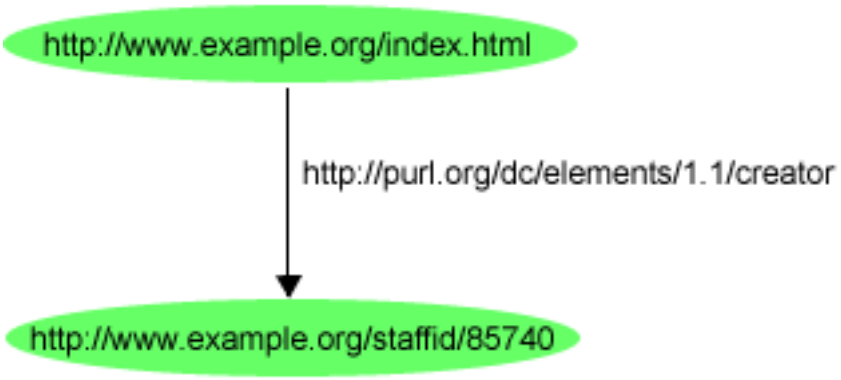

Figura 3.1: Rappresentazione di una struttura rdf.

sa http://www.example.org/staffid/85740 è creatore (il predicato è definito a http://purl.org/dc/elements/1.1/creator) della risorsa reperibile presso http://www.example.com/index.html.
La stessa cosa scritta nella sintassi RDF/XML è rappresentata in tabella 3.1.

```
1. <?xml version="1.0"?>
2.   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3.    xmlns:dc="http://purl.org/dc/elements/1.1/"
4.    xmlns:exterms="http://www.example.org/terms/">
5.   <rdf:Description rdf:about="http://www.example.org/index.html">
6.     <dc:creator rdf:resource="http://www.example.org/staffid/85740"/>
7.   </rdf:Description>
8. </rdf:RDF>
```

Tabella 3.1: Esempio di RDF/XML

Pro e contro

Pro:

- è una tecnologia molto semplice ma potente;
- essendo pensata per un utilizzo su web potrebbe aprire le strade alla creazione di un sistema fortemente interoperabile.

Contro:

- è ancora piuttosto giovane ed in fase di discussione;
- i prodotti che la implementano non supportano ancora uno standard per l'interrogazione e per la gestione dei permessi.

Possibili implementazioni

Sesame: implementazione di OpenRDF di una base dati RDF;

RedLand: altra implementazione di base dati RDF da parte di RedLand.

3.2 Sistemi di persistenza degli oggetti

Uno degli obiettivi da tenere in considerazione è quello di riuscire a creare un sistema CM in grado di interfacciarsi a backend differenti in maniera (il più possibile) trasparente. Inoltre vi è l'esigenza di poter scrivere e leggere oggetti in maniera "nativa" dalla base dati. Per poter fare ciò è necessario adottare una soluzione per la persistenza degli oggetti. Tra le tante disponibili per il linguaggio Perl è stato scelto SPOPS[SPO] che verrà presentato nella sezione seguente.

SPOPS

SPOPS è un sistema di persistenza di oggetti Perl scritto, a sua volta, in Perl. Esso permette di salvare oggetti in maniera persistente su diversi supporti. Per introdurre questo ambiente per la persistenza degli oggetti facciamo notare, innanzitutto, il significato dell'acronimo SPOPS che sta per: "Simple Perl Object Persistence with Security" ovvero "Semplice Sistema di Persistenza di Oggetti Perl con Sicurezza". La scelta è ricaduta su questo sistema per alcuni motivi:

Interfacciabilità: SPOPS è un ambiente che ha la possibilità di interfacciarsi a backend differenti quali LDAP, DBMS, ...;

Trasparenza: il sistema è stato progettato in maniera che l'utilizzo di dati provenienti da fonti differenti sia il più simile possibile, esistono tuttavia delle differenze intrinseche dei modelli dati implementati dai vari backend che impediscono una trasparenza totale a seconda delle situazioni;

Sicurezza: questo modulo tiene in considerazione anche la questione della sicurezza dei dati. Questo è un aspetto che spesso viene trascurato ma

in ambito di applicazioni multiutente diviene un pilastro fondamentale. In SPOPS è possibile definire i permessi in maniera granulare: questi sono specificabili per la singola istanza di oggetto ed è addirittura possibile legarli alla definizione dell'oggetto in maniera che questo li possa ereditare se sprovvisto. I permessi sono specificati all'interno di access list nel classico formato UNIX (Utente/Gruppo/Mondo) con le seguenti possibilità:

NONE: l'utente/gruppo/mondo non ha alcun permesso;

SUMMARY: l'utente/gruppo/mondo può conoscere l'esistenza dell'oggetto;

READ: l'utente/gruppo/mondo può leggere il contenuto dell'oggetto;

WRITE l'utente/gruppo/mondo può leggere, scrivere e cancellare l'oggetto.

Modificabilità: SPOPS è strutturato in maniera da essere facilmente modificabile così da poter aggiungere supporto per nuovi backend o adattare la libreria alle proprie necessità.

3.3 Linguaggi

Gli strumenti principali per la realizzazione di progetti sono sicuramente i linguaggi di programmazione. Alcuni verranno utilizzati all'interno del progetto e quindi descritti in questa sezione. I linguaggi utilizzati saranno:

- Perl
- JavaScript
- XHTML

3.3.1 Perl

Il linguaggio Perl[Per] nasce dal lavoro di Larry Wall negli anni '80, precisamente la release 1.0 esce nel 1987. Questo linguaggio è nato dalla necessità di una serie di tool per la manutenzione e configurazione di alcuni server. Successivamente si ebbe anche il bisogno di creare un tool per l'elaborazione della reportistica di un servizio di news: fu così che Wall pensò di scrivere quest'ultimo in un linguaggio per l'elaborazione di testi chiamato Awk. Tuttavia ben presto scoprì che mancava di alcune caratteristiche a lui necessarie. Decise quindi di creare un nuovo linguaggio che dopo molta indecisione venne

chiamato Perl ovvero "Pratical Extraction and Report Language" ("Linguaggio Pratico per l'Estazione e la Reportistica")[WCO00]. Questo linguaggio già dalla prima release si dimostrò veloce, potente e versatile e così naque la comunità Perl che ora conta moltissime persone. Una delle feature che ha reso famoso Perl è costituita dal motore per le espressioni regolari: infatti l'implementazione delle regex di Perl, ed anche la sintassi, nel mondo UNIX sono uno standard de facto. Questo strumento è molto potente e garantisce una versatilità enorme nell'elaborazione di testi cosa che ha fatto sì che il Perl si guadagnasse l'appellativo di "linguaggio colla". Esso fornisce agli amministratori di sistema una piattaforma in grado di aiutarli nel loro lavoro che gli permette di automatizzare anche processi molto complessi. Di primo acchito Perl potrebbe sembrare un linguaggio poco potente con il quale sviluppare solo script per l'amministrazione di sistema. Invece, questo linguaggio può essere tranquillamente utilizzato anche per costruire ad esempio interfacce grafiche, server, ... (tant'è che è utilizzato addirittura nel campo della bioinformatica).
Uno strumento che fa di Perl un ambiente di sviluppo molto comodo è CPAN[CPA] (Comprehensive Perl Archive Network) ovvero una serie di server online nel quale vengono raccolte un'enormità di librerie, per i più svariati utilizzi, scritte da autori di tutto il mondo (al momento attuale, 29.06.2005, il sito ospita 2906 MB di progetti consistenti in 8257 moduli). La maggiore comodità di CPAN è il fatto che vi sia un tool per accedervi che in maniera automatica si occupa di installare i moduli risolvendone anche le dipendenze.
Il linguaggio Perl ha un approccio differente rispetto a molti altri, infatti i suoi tipi sono scalari, vettori ed hash (vettori con indice alfanumerico). Ogni variabile è preceduta da un carattere che ne indica il tipo secondo la tabella 3.2. Nell'esempio alla tabella 3.3 invece viene mostrato l'uso dei tipi di

Tipo	Simbolo
Scalare	$
Array	@
Hash	%

Tabella 3.2: Prefissi indicanti il tipo delle variabili

dato Perl. I tipi di dato (intesi come intero, stringa, ...) vengono gestiti in maniera automatica, infatti Perl è un linguaggio tipato dinamicamente e quando necessario l'interprete converte il dato in un altro tipo.
Un'altra caratteristica che rende il Perl differente è il fatto che non sono gli operatori ad adattarsi al contesto come ad esempio in Java dove

```
1. 3 + 3;
```

esegue la somma tra interi e

```
1. "3" + "3";
```

concatena i numeri in stringa. In Perl è l'operatore che converte gli operandi in base al contesto, ad esempio:

```
1. "3" + "3";
```

esegue la somma tra i numeri, al contrario di quanto ci si potrebbe aspettare. La concatenazione delle stringhe si esegue invece in questa maniera:

```
1. "3" . "3";
```

è quindi l'operatore utilizzato a determinare il risultato dell'operazione.
Anche lo scoping è differente nel linguaggio Perl, e ne esistono tre tipologie:

Globale: una variabile se non viene specificato lo scope automaticamente viene posta come globale ed è visibile all'interno del package come nell'esempio:

```
#!/usr/bin/perl

# Esempio di scope globale

{
        $variable = "something";
}

{
        # Stampa "something"
        print $variable . "\n";
}
```

Locale: una variabile se viene specificato lo scope "local" assume una visibilità di tipo dinamico e viene vista dal punto in cui è dichiarata in poi:

```
#!/usr/bin/perl

# Esempio di scope locale
```

```
$variable = "something";

sub a_sub
{
        print $_[0] . ". " . $variable . "\n";
}

{
        local $variable = "something other";
        # Stampa "1. something other"
        print "1. " . $variable . "\n";
        # Stampa "2. something other"
        a_sub(2);
}

{
        # Stampa "3. something"
        print "3. " . $variable . "\n";
        # Stampa "4. something"
        a_sub(4)
}
```

Lessicale: infine lo scope lessicale si ha quando viene specificato il modificatore "my", in questo caso la variabile viene vista solo all'interno del blocco che la contiene ed ha scope statico:

```
#!/usr/bin/perl

# Esempio di scope lessicale
$variable = "something";

sub a_sub
{
        print $_[0] . ". " . $variable . "\n";
}

{
        my $variable = "something other";
        # Stampa "1. something other"
        print "1. " . $variable . "\n";
        # Stampa "2. something"
```

```
        a_sub(2);
}

{
        # Stampa "3. something"
        print "3. " . $variable . "\n";
        # Stampa "4. something"
        a_sub(4)
}
```

In Perl esistono anche i package i quali sono utili per due motivi:

visibilità: i simboli dichiarati globali verranno salvati nella symbol table del package in uso insieme alle subroutine; in questa maniera è possibile scrivere moduli aventi nomi di variabili e nomi di funzioni uguali senza che questi collidano;

modularità: è possibile suddividere tramite i package una serie di funzionalità le quali possono poi essere importate ed utilizzate dove necessario.

In Perl sono implementati anche gli oggetti, tuttavia lo sono in una maniera piuttosto singolare e non molto rigorosa: in Perl un oggetto non è altro che una struttura dati costituita o da uno scalare o da un array o da un hash o da una loro composizione che viene legata ad un package. In questa maniera la variabile conosce il proprio tipo ed è in grado di chiamare i metodi del package a cui è legata. L'assenza di rigore sta nel fatto che non è possibile definire esplicitamente una visibilità sugli attributi, il concetto di visibilià esiste soltanto come convenzione.
L'ultimo aspetto che vorrei citare del Perl consiste nella sua estrema ricchezza. Molti linguaggi cercano di ridurre al minimo le caratteristiche di base, Perl invece adotta l'approcio opposto definendo operatori che potrebbero essere tranquillamente espressi tramite altri. Questa caratteristica rende (solitamente) un programma Perl molto più conciso del corrispondente scritto in un altro linguaggio. L'estrema ricchezza è quella che ha fatto sì che uno dei motti di Perl sia TMTOWTDI ovvero "There's More Than One Way To Do It" ("C'è Più Di Una Maniera Di Farlo").

3.3.2 JavaScript

JavaScript è un linguaggio di scripting orientato agli oggetti e viene solitamente utilizzato per aggiungere dinamicità alle pagine web. Questo lin-

guaggio nasce per opera di Brendan Eich impiegato alla Netscape Corporation. Inizialmente denominato Mocha diverrà poi LiveScript. Il nome di JavaScript è adottato da Netscape in concomitanza con l'introduzione del supporto a Java nel browser probabilmente per sfruttare l'onda del brand di Sun. In seguito questo linguaggio è stato standardizzato dalla European Computer Manufacturer Association (ECMA[ECM]) che lo ha denominato ECMAScript (oltre ad essere uno standard ECMA è anche uno standard ISO).
La sintassi di JavaScript assomiglia a quella di C, non possiede i propri costrutti per l'input/output ma si basa su quelli fornitigli dall'ambiente che lo ospita. Solitamente JavaScript è utilizzato all'interno delle pagine web per corredarle di una parte dinamica (ad esempio sostituzione di un'immagine al passaggio del mouse per ottenere il cosidetto effetto di rollover). Il browser, per permettere al linguaggio di interagire con le pagine, esporta un'interfaccia ad oggetti denominata DOM (Document Object Model). Attraverso quest'interfaccia l'interprete JavaScript è in grado di modificare le pagine e collaborare con il browser. Oltre a svolgere questi compiti il linguaggio riesce ad intercettare e gestire gli eventi di interfaccia generati dall'interazione dell'utente con il browser (per esempio: il doppio click in un punto fa espandere una tabella).
JavaScript nelle sue innumerevoli implementazioni è utilizzato anche in altri ambiti:

Adobe Acrobat e Adobe Reader: questi programmi supportano JavaScript all'interno dei documenti PDF e in questa maniera è possibile rendere interattivi i form all'interno dei file PDF (esempi di utilizzo di JavaScript nei documenti PDF possono essere reperiti al seguente indirizzo: http://www.planetpdf.com/developer/article.asp?contentid=6575&ra);

Mozilla: le varie applicazioni prodotte dalla casa Mozilla (Mozilla Suite, FireFox, ThunderBird, SunBird, ...) sono tutte basate su due elementi ovvero:

JavaScript: per quanto riguarda la parte di programmazione della logica dell'interfaccia.
In Mozilla l'engine che esegue il codice si chiama SpiderMonkey[Spi] ed è scritto in C;

XUL (XML User Interface Language): per la parte di interfaccia Mozilla si affida invece a XUL un metalinguaggio basato su XML attraverso il quale vengono definite le GUI (Graphics User Interface) e al quale viene collegata la gestione degli eventi. Questo lin-

guaggio oltre ad essere utilizzato nativamente da Mozilla può essere sfruttato anche per creare delle pagine web molto più complesse ed interattive (come quella reperibile a http://www.hevanet.com/acorbin/xul/top.xul);

Microsoft WSH: WSH sta per "Windows Script Host" ed è un'implementazione di JavaScript di Microsoft utile per la gestione del sistema operativo, attraverso questo linguaggio e una sorta di DOM è possibile agire su varie impostazioni (ad esempio si può aggiungere una variabile d'ambiente).

Una delle difficoltà maggiori nello sviluppo di JavaScript risiede nel fatto che le implementazioni di questo linguaggio sono spesso incompatibili e non conformi allo standard. Questi "dialetti" creano parecchi problemi agli sviluppatori che devono aggirarli in maniere spesso non troppo eleganti. Uno delle punti dolenti risiede nella rilevazione dell'implementazione che si sta utilizzando. Talvolta il browser o l'implementazione JavaScript espongono tra gli altri oggetti il nome e la versione, mentre in altri casi (Internet Explorer) la rilevazione del browser dev'essere fatta controllando alcune proprietà (particolarità non-standard esistenti solo in un browser) del client. Per creare quindi un'applicazione JavaScript "multipiattaforma" è necessario sviluppare una sorta di "middleware" (che si occupa di appianare le differenze) e basare il programma su di esso.
Le più famose implementazioni sono:

JavaScript: è utilizzata da Mozilla. Essa non segue alla lettera lo standard ma vi è comunque più vicina rispetto all'implementazione utilizzata da Internet Explorer;

JScript: è la versione di Microsoft, inserita nel browser e sfruttata anche da WSH. Essa è molto differente dallo standard.

Un'altro problema è costituito da differenze a livello di DOM nei browser, anche queste creano diversi grattacapi ai programmatori costretti ad eliminare/attenuare le differenze alla bell'e meglio.
Per quanto riguarda il linguaggio JavaScript supporta sia lo stile di programmazione imperativo che quello ad oggetti in una maniera inconsueta che verrà esposta più avanti. Il linguaggio è tipato dinamicamente quindi le variabili, le quali vengono definite dall' assegnazione di un valore o tramite la keyword "var", possono cambiare tipo di contenuto a run-time. Lo scope delle variabili dipende dal contesto, infatti se queste sono definite all'interno

di una funzione lo scope è limitato ad essa mentre la dichiarazione all'esterno comporta una visibilità globale.
JavaScript supporta gli oggetti ma non le classi, infatti, in esso non è possibile dichiarare una classe ma ne viene definito un prototipo in maniera addittiva ovvero "appiccicandovi" funzioni ed attributi. Ad esempio se volessimo realizzare un oggetto Person corredato di funzione di stampa lo dovremmo fare in questa maniera:

```
// Definizione della funzione di stampa dell'oggetto
function PrintPerson()
{
        document.write("Name: " + this.Name);
        document.write("Surname: " + this.Surname);
        document.write("Address: " + this.Address);
}

// Definizione del costruttore dell'oggetto
function AddressBook(Name, Surname, Address)
{
        this.Name = Name;
        this.Surname = Surname;
        this.Address = Address;
        this.Print = PrintPerson;
}

// Creazione di un nuovo oggetto
MarioRossi = new Person("Mario", "Rossi", "via Mazzini, 12");
// Stampa del contenuto dell'oggetto
MarioRossi.Print();
```

Gli oggetti non sono altro che array associativi essendo possibili, infatti, due forme di accesso agli attributi, come ad esempio:

```
// Accesso al valore con la sintassi ad oggetto
alert(MarioRossi.Name);

// Accesso al valore con la sintassi ad array associativo
alert(MarioRossi["Name"]);
```

ovvero quella nella notazione ad oggetto e quella nella notazione ad hash.
Esistono in JavaScript alcuni oggetti predefiniti:

- Array (vettori);
- Boolean (booleani);
- Date (data e ora);
- Function (funzione);
- Math (funzioni matematiche);
- Number (numeri);
- Object (oggetti);
- RegExp (espressioni regolari);
- String (stringhe);

e altri nel browser (attraverso il DOM):

- window (finestra);
- document (documento);
- form (controlli);
- link (collegamenti);
- ....

Un utilizzo frequente di JavaScript è la gestione degli eventi generati dal browser automaticamente o in seguito ad intervento da parte dell'utente. Una lista di questi eventi è la seguente:

onAbort: lanciato alla pressione del tasto di stop del browser;

onBlur: lanciato alla perdita del fuoco da parte di un componente;

onChange: lanciato al cambiamento di qualcosa;

onClick: lanciato al click del mouse;

onDblClick: lanciato al doppio click del mouse;

onDragDrop: lanciato durante un Drag & Drop;

onError: lanciato in caso di errore;

onFocus: lanciato al guadagno di fuoco di un componente;

onKeyDown: lanciato all'abbassamento di un tasto;

onKeyPress: lanciato alla una pressione e successivo rilascio di un tasto;

onKeyUp: lanciato al rilascio di un tasto;

onLoad: lanciato al caricamento di una pagina;

onMouseDown: lanciato all'abbassamento di un tasto del mouse;

onMouseMove: lanciato al movimento del mouse;

onMouseOut: lanciato all'uscita del mouse da un componente;

onMouseOver: lanciato all'entrata in un componente;

onMouseUp: lanciato al rilascio di un tasto del mouse;

onMove: lanciato al movimento di una finestra o frame;

onReset: lanciato alla pressione del tasto "annulla" di un form;

onResize: lanciato alla modifica della dimensione della finestra del browser;

onSelect: lanciato alla selezione del testo in una casella di input;

onSubmit: lanciato all'invio del contenuto di un form;

onUnload: lanciato all'uscita da una pagina.

Questi eventi (esposti tramite il DOM) permettono agli sviluppatori di creare pagine e siti in grado di interagire agli input dell'utente. Le ultime versioni di JavaScript sono anche in grado di effettuare una gestione degli errori tramite blocchi try-catch, in maniera molto simile a C++ o Java.
Per la parte client la scelta è ricaduta su JavaScript soprattutto per il fatto che per l'utilizzo di questo linguaggio non è necessaria l'installazione di alcun plug-in (a meno di non avere installato un browser molto vecchio). In più essendo una tecnologia fortemente integrata nel browser semplifica l'interazione con esso e ha velocità di esecuzione (a patto che sia scritto bene) maggiori rispetto a soluzioni alternative. Anche l'utilizzo di memoria è minore essendo questo linguaggio molto snello e povero di librerie.[Wik]

3.3.3 XHTML

L'XHTML è un linguaggio di markup basato su XML creato per sostituire l'HTML.
Principalmente esso è stato introdotto con cinque scopi:

- rendere più severe le regole di sintassi;
- accentuare la separazione contenuto presentazione;
- riformulare HTML 4 come una specifica XML;
- rendere fruibile il web da dispositivi mobili;
- rendere la specifica di HTML modulare in maniera da poterla espandere.

L'XHTML 1.0 è divenuto una specifica del W3C il 26 gennaio 2000 introducendo cambiamenti minimi rispetto all'HTML. La specifica di quest'ultimo è suddivisa in tre sottocategorie:

Strict: questa versione rimuove la possibilità di utilizzare elementi relativi alla presentazione nel markup;

Transitional: questa invece è meno restrittiva per permettere una transizione più facile dall'HTML;

Frameset: permette l'utilizzo dei frameset.

È stata in seguito fatta una revisione di XHTML 1.0 ed è così nato XHTML 1.1. Questo permette l'importazione di feature addizionali nel markup e prevede inoltre l'uso del markup ruby tramite il quale è possibile inserire caratteri asiatici.
La specifica ora in lavorazione da parte del W3C è XHTML 2.0. Essa più che una revisione andrebbe considerata come un nuovo linguaggio di markup poiché non si cura della compatibilità all'indietro.
Alcune delle principali feature sono:

- form HTML sostituiti con XForms;
- frame HTML sostituiti con XFrames;
- eventi del DOM sostituiti con XML Events.

Altre particolari famiglie di XHTML sono:

XHTML Basic: questa è una versione ridotta dell'XHTML pensata per quei device come telefoni cellulari o palmari che dispongono di poche risorse;

XHTML Mobile Profile: è basata sulla versione Basic ed è sviluppata per poter aggiungere elementi specifici per i cellulari.

Una fase importante per quanto riguarda l'XHTML è quella della validazione, infatti i documenti per poter essere visualizzati devono essere ben-formati (cioè correttamente definiti dal punto di vista strutturale) e conformi ai DTD (Document Type Declaration) riportati. Per quanto riguarda le varie versioni di XHTML vi sono diversi DTD:
XHTML 1.0 Strict:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

XHTML 1.0 Transitional:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

XHTML 1.0 Frameset:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Frameset//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-frameset.dtd">
```

XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

In base a queste dichiarazioni un documento può essere quindi controllato per verificare la sua conformità DTD utilizzato.[Wik]

3.4 Tecnologie

Questa sezione raccoglie una descrizione delle tecnologie su cui si basa la proposta di architettura:

- Ajax
- DOM
- POE
- Mason

3.4.1 AJAX

AJAX è un approccio differente al web. Il significato di questo acronimo è: "Asynchronous JavaScript + XML" il quale indica l'utilizzo di alcune tecnologie volte ad ottenere un aumento dell'interattività del web fortemente limitato dal modello Pagina → Richiesta → Pagina. Questa maniera di agire è dovuta alla natura ipertestuale del web. I punti dolenti di ciò sono sostanzialmente due:

- carenza di interattività;
- tempi di attesa fastidiosi per gli utenti.

L'approcio AJAX prevede sostanzialmente di inserire uno strato software, come visibile in figura 3.2, tra l'utente ed il server.
AJAX si basa su cinque componenti:

XHTML e CSS: questi componenti stanno alla base del we rispettivamente per contenuto e presentazioneb;

DOM: permette l'interazione tra il browser e JavaScript;

XML e XSLT: essi permettono lo scambio e la manipolazione dei dati;

XMLHTTPRequest: serve per gestire il recupero asincrono dei dati;

JavaScript: necessario per creare la parte interattiva e collegare tra di loro i componenti precedenti.

Tramite l'uso di questi cinque AJAX è in grado di gestire le richieste dell'utente in maniera indipendente dalla comunicazione con il server. Il funzionamento è piuttosto semplice: le richieste anziché essere inviate direttamente

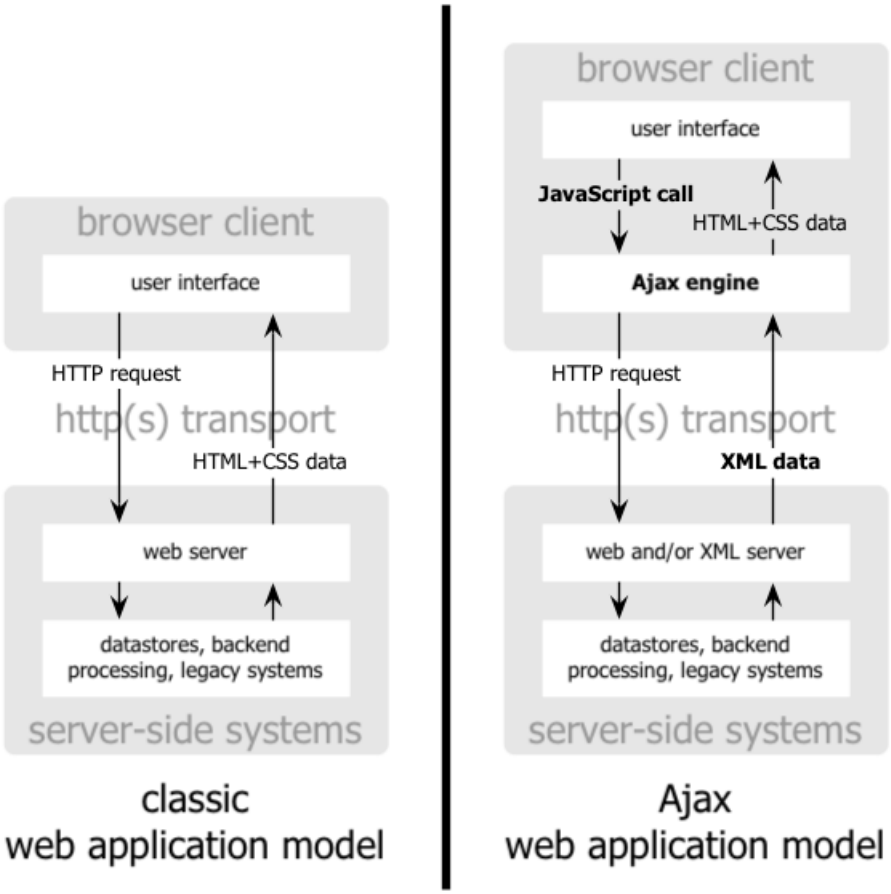

Figura 3.2: Il modello tradizionale confrontato con il modello Ajax.

al server vengono fatte in locale al motore JavaScript. Questo si occupa di verificare se la richiesta dev'essere inoltrata al server e nel frattempo risponde all'utente ad esempio visualizzando una piccola barra di caricamento.

Un ulteriore vantaggio dell'uso di questo modello è la possibilità di sostituire in maniera arbitraria parti del contenuto della pagina evitando di scaricare le porzioni invariate (ad esempio l'header, il menu e il footer). Questo avviene dinamicamente.

Un possibile scenario derivante dall'utilizzo di AJAX è un'applicazione per il controllo di una macchina remota, che visualizza i dati del carico e della memoria utilizzata affiancandovi anche dei grafici aggiornati in tempo reale. Altro fattore notevole derivante dall'utilizzo di questa tecnologia è la possibilità per il server di inviare qualsiasi cosa al client. Nell'esempio portato nel paragrafo precedente, infatti, potrebbe essere tranquillamente il server ad informare il client delle variazioni. Questo permette di evitare il traffico

derivante dal polling da parte del client.
Indirettamente questo modello porta ad un risparmio di risorse da parte del server. Infatti, la possibilità di aggiornare soltanto le sezioni modificate di una pagina, permette al server di risparmiare potenza e memoria necessarie alla costruzione dinamica delle pagine e la banda necessaria all'invio dei dati.
Qualcuno potrebbe obiettare che questo tipo di interazione è già ottenibile utilizzando tecnologie quali Applet Java e/o Flash. Tuttavia AJAX ha alcuni vantaggi in più:

standard: le tecnologie su cui si basa sono degli standard internazionali, questo amplia il ventaglio di browser utilizzabili come client (purtroppo accade molto spesso che le implementazioni differiscano o siano carenti rispetto agli standard);

no plugin: non è previsto l'utlizzo di alcun plugin, infatti i componenti utilizzati nell'architettura sono già presenti ed usati in ambito web;

leggero: l'utilizzo di elementi già presenti nel browser permette a questo strato software di essere più snello rispetto alle controparti le quali spesso prevedono il caricamento di librerie aggiuntive. Ciò è avvalorato dal fatto che JavaScript è un linguaggio molto semplice;

integrato: la maggiore integrazione nel browser permette di sfruttarne appieno le caratteristiche, cosa molto meno naturale se si utilizzano altre tecnologie;

stabile: i componenti su cui si basa AJAX sono sfruttati da molto tempo, possono essere quindi considerati piuttosto affidabili.

Il modello AJAX non esiste soltanto a livello teorico, vi sono infatti diverse applicazioni sul web che lo sfruttano come:

orkut: comunità online;

GMail: webmail di Google;

Google Groups Beta: gruppi di discussione di Google;

Google Suggest: motore di ricerca con completamento auomatico sulle parole inserite;

Google Maps: visualizzazione della terra dallo spazio;

Flickr: sito per lo scambio di fotografie;

A9: motore di ricerca di Amazon.

(per una lista maggiormente dettagliata è possibile visitare AJAXMatters). L'esistenza di queste implementazioni dimostra l'applicabilità a casi reali di qualsiasi dimensione.[AJA]

3.4.2 DOM

Il Document Object Model[DOMa] è un'interfaccia, indipendente da linguaggio e piattaforma utilizzati, per la rappresentazione di documenti strutturati. Esso fornisce accesso ai vari elementi (del documento) ed alle loro proprietà tramite una struttura ad albero. Si utilizza questo modello per rappresentare documenti ben-formati cioè strutturalmente conformi alle specifiche (XML o HTML). Il modello DOM proposto dal W3C fornisce un'interfaccia di accesso alle proprietà del browser e del documento visualizzato. Esso viene anche utilizzato nel parsing di file XML (esiste un'altra maniera di analisi dei documenti XML, ovvero SAX: "Simple API for XML" tramite la quale viene utilizza meno memoria in quanto il documento viene letto sequenzialmente come fosse uno stream). Questo rende molto semplice l'accesso, l'aggiunta, la cancellazione e la modifica di contenuti, attributi e stili.
La specifica DOM è suddivisa in 3 livelli:

Level 1: descrive le specifiche di base per navigare e manipolare documenti (X)HTML o XML;

Level 2: aggiunge il supporto agli stylesheet, ai namespace XML ed al modello degli eventi;

Level 3: il terzo livello infine aggiunge il supporto ai modelli per i contenuti (DTD e XML Schema) ed alla validazione oltre a specificare anche il caricamento, il salvataggio, le viste e la formattazione dei documenti e gli eventi della tastiera.

Ogni livello si suddivide inoltre in tre parti:

Core: definisce un insieme di oggetti per l'accesso a qualsiasi tipo di documento strutturato;

XML: definisce un insieme di oggetti per l'accesso a documenti XML;

(X)HTML: definisce un insieme di oggetti per l'accesso a documenti (X)HTML.

[Wik][DOMb]

3.4.3 POE

Il framework POE (Perl Object Environment) viene in aiuto in quei casi in cui si necessita di scrivere applicazioni Perl che gestiscano dei flussi in maniera asincrona. POE in pratica è una sorta di mini sistema operativo scritto in Perl il quale gestisce una serie di stati attivati da eventi. Esso implementa tutte quelle parti di architettura che devono essere riscritte in continuazione così da renderle disponibili e riutilizzabili definitivamente.
Per quanto riguarda la parte di rete POE fornisce tre livelli a cui poter agire:

Alto: a questo livello vengono forniti componenti per poter scrivere servizi anche piuttosto complessi con poche righe di codice;

Medio: i componenti forniti a questo livello sono un po' meno specializzati, essi quindi danno la possibilità di agire ad un livello più basso rispetto al precedente;

Basso: a questo livello la specializzazione è assente e tramite gli strumenti offerti è possibile scrivere componenti in grado di gestire nuove tipologie di servizi di rete.

Come accennato sopra, POE è una sorta di mini sistema operativo il quale gestisce i vari processi al suo interno in maniera parallela. Il multitasking di POE è basato sugli eventi e gli handler e, a differenza di altri linguaggi, non si appoggia sui thread del sistema operativo o sulla chiamata alla funzione di sistema "fork". Questa caratteristica permette a POE di essere utilizzato anche in casi in cui i thread non siano disponibili in maniera nativa.
All'interno di POE gli eventi sono messaggi che informano che è accaduto qualcosa come: I/O su rete, passaggio del tempo o stato dei processi, ... POE attraverso gli Event Watcher controlla varie situazioni e nel caso in cui avvenga qualcosa genera un evento.
Un'altra caratteristica di POE è quella di evitare alcuni problemi come la corruzione della memoria e l'overhead dovuto al multithreading. Il primo problema è aggirato eseguendo gli handler come blocchi critici. Questo tuttavia implica l'impossibilità di eseguire operazioni molto lunghe o bloccanti le quali comporterebbero la perdita il parallelismo. Il problema dell'overhead invece viene evitato da POE non utilizzando thread: in questo modo esso non occupa risorse aggiuntive. POE comunque può supportare attraverso l'uso di un componente anche i thread e i processi del sistema operativo. Tuttavia il supporto per i thread in Perl è ancora giovane, quindi se ne consiglia l'utilizzo soltanto in caso di necessità.
POE prevede anche un'architettura per il passaggio di messaggi tra parti dell'applicazione favorendo così la creazione di architetture fortemente modulari.

Esiste inoltre la possibilità di fare collaborare più kernel POE attraverso il passaggio di eventi, permettendo così di creare facilmente applicazioni distribuite.
Oltre a tutte queste possbilità POE gode anche della presenza di una comunità che ha creato e messo a disposizione attraverso il sistema CPAN una gran quantità di componenti e moduli. POE è basato su tre componenti:

Stati: è il blocco sul quale si basa tutto il resto: quando arriva un evento viene chiamata la funzione che si è registrata;

Kernel: si occupa, come il kernel di un sistema operativo, dell'esecuzione dei vari processi (in POE chiamati sessioni), della loro schedulazione, dei loro dati e di molto altro. Fornisce anche diversi servizi di più basso livello come la possibilità di registrare degli allarmi;

Sessioni: sono l'equivalente dei processi dei sistemi operativi costituiti da più stati tra i quali poter saltare. Ogni sessione ha il proprio insieme di dati salvato in un hash denominato "heap".

Ad un livello inferiore troviamo i seguenti componenti:

Driver: sono i componenti base utilizzati per l'I/O, essi leggono e scrivono da e verso le risorse più eterogenee;

Filter: questi oggetti sono necessari per effettuare la conversione tra diversi formati;

Wheel: questi componenti incapsulano parti utili di logica di alto livello. Essi permettono di riutilizzare quelle sezioni di codice di uso comune;

Component: infine i componenti sono delle sessioni controllabili da altre sessioni. Svolgono i compiti più disparati.

[POEb][POEa]

3.4.4 Mason

Mason è un framework per la creazione di grandi siti web. Esso è organizzato in maniera da aiutare nella produzione e nella manutenzione dei siti. Inoltre Mason stimola l'autore a pensare al sito in termini di una struttra piuttosto che di una collezione di script.
Vedremo ora una descrizione delle caratteristiche salienti di questo framework:

Componenti: una delle caratteristiche fondamentali di Mason riguarda la possibilità di creare dei componenti. Questi sono spezzoni di codice che possono ricevere un input e produrre un output. Il prodotto dei componenti è costituito da XHTML (anche se è possibile produrre altri tipi di output) mescolato a codice Perl e direttive Mason. I componenti possono chiamarsi a vicenda in qualunque punto del codice. In questa maniera è possibile scomporre logicamente la struttura del sito e delle stesse pagine, andando così ad organizzare in maniera strutturata il tutto. Questo permette di semplificare estremamente la manutenzione, infatti se ad esempio fosse necessario cambiare un elemento nell'header delle pagine basterebbe modificare il/i componente/i che lo producono e verrebbe automaticamente cambiato in tutto il sito;

Ereditarietà: una particolarità di Mason riguarda appunto il fatto che i componenti possono utilizzare l'ereditarietà in maniera simile agli oggetti. Il fatto che i componenti abbiano questa capacità apre la strada ad un mare di possibilità. Ad esempio si riutilizzare in maniera proficua e molto ordinata componenti già scritti;

Caching intelligente: Mason dispone di un meccanismo per il caching molto elaborato. Esso, infatti, permette di regolare il tempo di caching sulla base di molti parametri: dal tempo all'utente che si logga ed altro ancora.

Altra caratteristica peculiare è il sistema di caching automatico interno. Questo si basa sul fatto che le pagine all'interno del framework vengono generate seguendo alcune fasi le quali sono la compilazione da codice Mason nel linguaggio Perl e da Perl in bytecode. Questi passi se ripetuti in continuazione potrebbero risultare pesanti per questo Mason salva ogni risultato temporaneo su disco ed ogni volta che una pagina viene chiamata esegue il bytecode ed invia l'output al client. I vari risultati intermedi vengono rigenerati soltanto in seguito ad un controllo di ultima data di modifica. È inoltre possibile, una volta ultimato lo sviluppo, disabilitare il controllo per ottenere performance ancora migliori;

Integrazione Apache: Apache è probabilmente il server http più utilizzato nel mondo. Esso presenta caratteristiche molto avanzate tra le quali il mod_perl. Esso è un modulo di Apache che integra la versatilità di questo server http con la potenza di Perl. Mason è stato quindi progettato per essere in grado di interagire con Apache: in questa maniera è possibile legare i due ambienti e sfruttare Apache per l'ottima gestione

delle connessioni e tutto il resto e Mason per la generazione dell'output servito.

[RW02][Masb]

3.5 Protocolli

In questa sezione verranno illustrati i protocolli utilizzati:

- JSON
- SOAP

3.5.1 JSON

JSON è un sottoinsieme del linguaggio JavaScript, il significato dell'acronimo infatti è "JavaScript Object Notation". JSON è un protocollo molto leggero. Esso si pone l'obiettivo di permettere lo scambio di strutture dati ed oggetti tra vari linguaggi e soprattutto verso JavaScript. Essendo le strutture dati scritte in questo linguaggio possono essere interpretate dal browser in maniera molto efficiente (attraverso una chiamata alla funzione "eval" che è in grado di eseguire il contenuto di una stringa). JSON è un formato totalmente basato sul testo e ciò lo rende utilizzabile senza problemi in molti contesti.
Esso è supportato in molti linguaggi di programmazione: ActionScript, C, C#, ColdFusion, E, Java, JavaScript, ML, OCAML, Perl, PHP, Python, Rebol e Ruby.
JSON è basato su due strutture:

- una collezione di coppie chiave/valore;
- una lista ordinata di valori.

Queste ultime esistono nella maggior parte dei linguaggi di programmazione. e viene quindi naturale la costruzione di un protocollo di interscambio dati che si basi su di esse e JSON fa proprio questo.
In JSON è possibile fare uso dei seguenti elementi:

Oggetti: essi vengono espressi come una sequenza non ordinata di coppie nome/valore. Per quanto riguarda la sintassi essi iniziano con una parentesi graffa aperta "{" e finiscono con la parentesi graffa chiusa "}". Ogni nome è seguito dai due punti ":" e le coppie nome/valore sono separate tramite virgole ",";

Array: questo è invece una sequenza ordinata di valori. Sintatticamente è possibile dichiarare un array tramite una parentesi quadra aperta "[" mentre la chiusura avviene logicamente tramite la corrispondente parentesi quadra chiusa "]". I valori sono separati tra loro per mezzo di virgole ",";

Valori: questi possono essere: stringhe tra virgolette, numeri, i booleani "true" e "false", "null", oggetti od array. È possibile creare qualsiasi composizione di essi anche annidandoli;

Stringhe: esse sono costituite da zero o più caratteri Unicode racchiusi tra virgolette. I caratteri utilizzati dal linguaggio possono essere inseriti facendo uso delle sequenze di escape (in pratica il carattere speciale preceduto da "\"). Un carattere viene rappresentato come una stringa di lunghezza 1.

La grammatica del linguaggio viene esplicitata nella tabella 3.4. Le immagini riportate nelle figure 3.4, 3.5 e 3.6 sono utili ad una migliore comprensione della grammatica stessa.

3.5.2 SOAP

SOAP è un protocollo basato su XML utilizzato per l'invocazione di metodi in oggetti remoti. L'acronimo infatti aveva inizialmente il significato di "Simple Object Access Protocol" ovvero "Semplice Protocollo per l'Accesso agli Oggetti". Solitamente questo protocollo viaggia al di sopra del protocollo HTTP. SOAP è anche il protocollo che sta alla base della tecnologia dei web service.
Questo metalinguaggio è stato originariamente progettato da Microsoft, ma la specificazione ora viene mantenuta da un gruppo interno al W3C.
SOAP, come già accennato si appoggia su HTTP e questo sicuramente lo avvantaggia rispetto ad altre soluzioni in quanto di solito all'interno delle organizzazioni esso non viene filtrato.
La scelta di basarlo su XML è stata portata avanti per il fatto che questo formato è molto diffuso, che è uno standard a livello industriale e che è largamente utilizzato dal mondo open source. Proprio la diffusione dell'XML favorisce l'esistenza di tool e librerie su cui basare il parsing di SOAP.
Il protocollo in discussione eredita da XML le possibilità di interoperabilità ma anche la pesantezza del parsing il quale può divenire un collo di bottiglia.
I messaggi SOAP sono contenuti in un "envelope" il quale è a sua volta suddiviso in due sezioni: l'header ed il corpo del messaggio. All'interno dell'header vi sono solitamente informazioni riguardanti l'envelope mentre nel

corpo vi è la richiesta o la risposta. Nella sua sintassi XML il SOAP si basa sul namespace xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/". Nelle tabelle 3.5 e 3.6 sono riportate rispettivamente un esempio di richiesta ed uno di risposta facenti uso di SOAP (sono tratti da [JSO]

```
   # Scalari
   # Scalare numerico
 1. $scalar_num = 10;
   # Scalare stringa
 2. $scalar_str = "String";

   # Vettori
   # Array di numeri
 3. @array_num = (0, 1, 2, 3);
   # Stampa 2
 4. print($array_num[2] . "\n");

   # Array di stringhe
 5. @array_str = ("zero", "one", "two", "three");
   # Stampa "one"
 6. print($array_num[1] . "\n");

   # Array Misto
 7. @array_mix = ("zero", 1, 2, "three");
   # Stampa three
 9. print($array_num[3] . "\n");

   # Vettori con indice alfanumerico (hash)
   # Hash con indici numerici
10. %array_num_idx = (0 => "zero", 1 => "one",
                      2 => "two", 3 => "three");
   # Stampa "two"
11. print($array_num{2} . "\n");
   # Hash con indici stringa
12. %array_str_idx = ("zero" => 0, "one" => 1,
                      "two" => 2, "three" => 3);
   # Stampa 1
13. print($array_num{one} . "\n");
   # Hash con indici misti
16. %array_idx_mix = (0 => "zero", "one" => 1,
                      "two" => 2, 3 => "three");
   # Stampa 2
17. print($array_num[two] . "\n");
```

Tabella 3.3: I tipi in Perl.

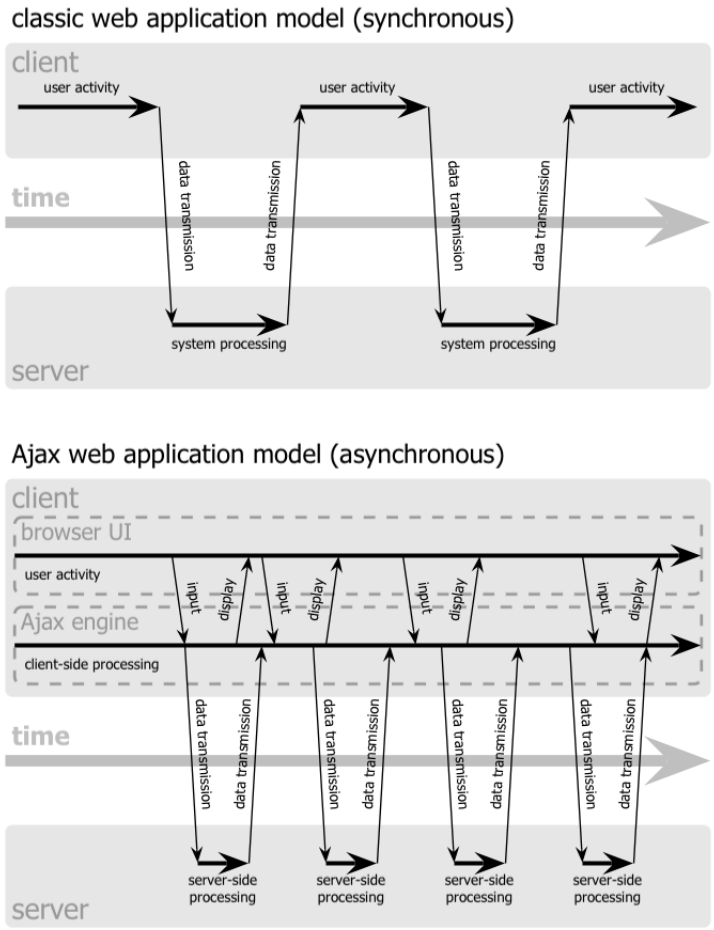

Figura 3.3: Il modello di interazione sincrono confrontato di un'applicazione tradizionale comparato al modello di interazione asincrono di un'applicazione ajax.

```
object
    { members }
    {}
members
    string : value
    members , string : value
array
    [ elements ]
    []
elements
    value
    elements , value
value
    string
    number
    object
    array
    true
    false
    null
```

Tabella 3.4: La grammatica di JSON.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails xmlns="http://warehouse.example.com/ws">
      <productID>827635</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

Tabella 3.5: Struttura di una richiesta SOAP.

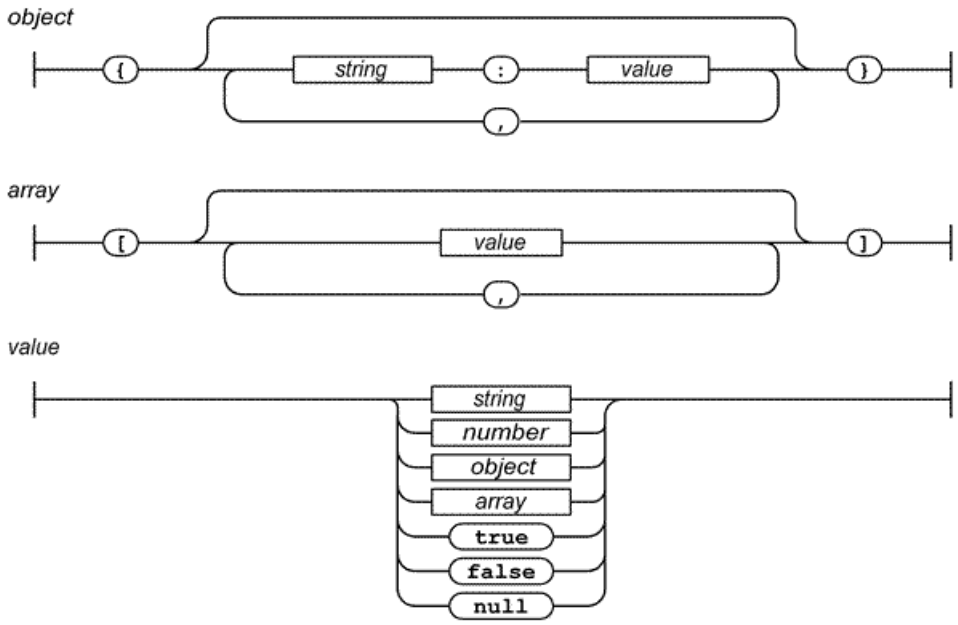

Figura 3.4: Le strutture dati di JSON.

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productName>Toptimate 3-Piece Set</productName>
        <productID>827635</productID>
        <description>3-Piece luggage set.  Black Polyester.</description>
        <price>96.50</price>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Tabella 3.6: Struttura di una risposta SOAP.

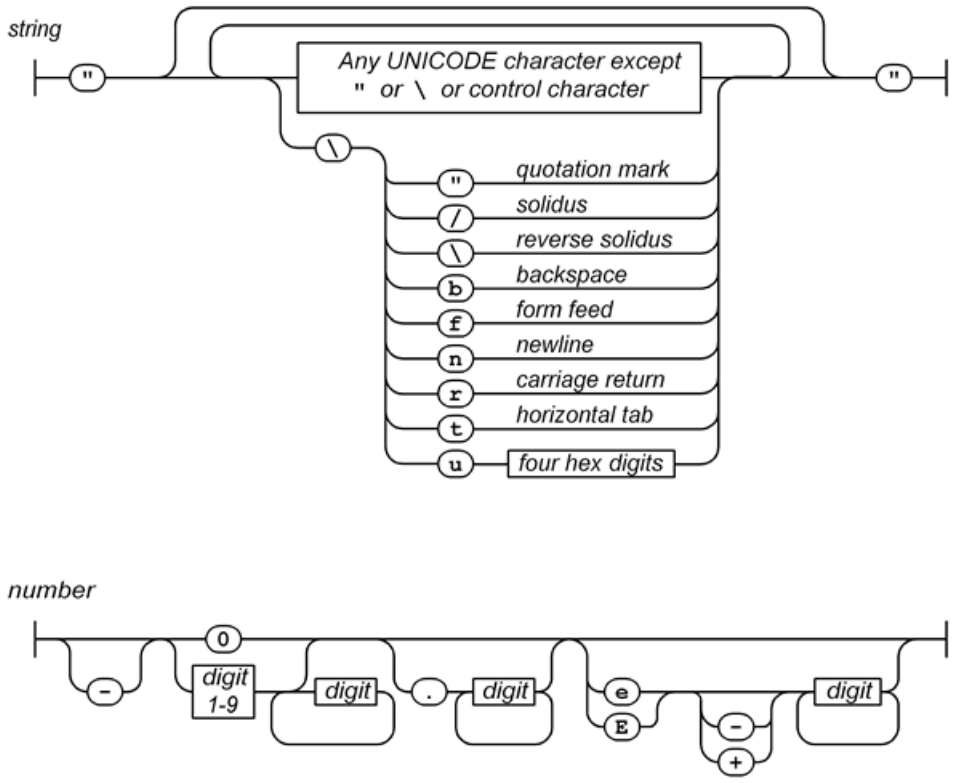

Figura 3.5: Codifica delle stringhe e dei numeri di JSON.

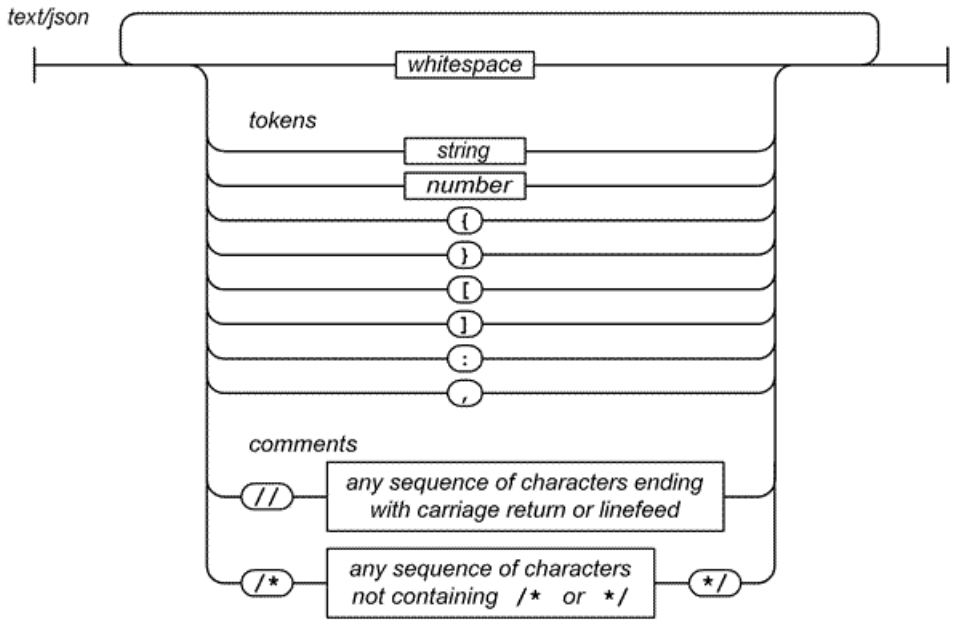

Figura 3.6: La sintassi totale di JSON.

Capitolo 4

Proposta architetturale di un CMS

4.1 Motivi per definire una nuova architettura

La domanda che sorge spontanea è: "Perché una nuova architettura?". A questo quesito le risposte sono molteplici e motivi individuati sono raggruppati in due categorie:

Tecnologici: in questo ambito possiamo inserire tutte le limitazioni delle soluzioni attuali ed anche alcune innovazioni;

Strategici: in questa categoria si vogliono inserire tutte le motivazioni riguardanti le scelte strategiche dell'azienda.

4.1.1 Motivi tecnologici

I motivi tecnologici si possono dividere in due sottogruppi:

Limiti: l'architettura si pone l'obiettivo di superare alcuni limiti posti da altre soluzioni:

Contenuti strutturati: i contenuti gestiti all'interno della maggior parte dei CMS sono di tipo testuale ed hanno una struttura molto semplice. In casi d'uso aziendale o in particolari domini applicativi (settore medico) vi può essere la necessità di creare dei dati complessi e strutturati. La proposta presentata è basata totalmente sugli oggetti, permette quindi di annidare i dati in base alle proprie necessità;

Modifica stessa interfaccia: le soluzioni attuali prevedono di fare l'editing dei dati del CMS da un'interfaccia differente rispetto a quella utilizzata per la visualizzazione e questo è sicuramente un limite. Nel progetto presentato è prevista la possibilità di modificare i dati direttamente dall'interfaccia di consultazione;

Innovazioni: l'architettura ne introduce alcune:

Asincronia: è forse la maggiore innovazione introdotta, dalla quale derivano molte possibilità. Infatti qualsiasi applicazione webè sempre funzionata in maniera sincrona (ciò è dovuto modello "richiesta → risposta"). In questo caso invece l'adozione dell'approccio AJAX e soprattutto degli eventi permette un comportamento di tipo asincrono;

Comunicazione bidirezionale: il server è in grado di inviare in qualiasi momento dati verso il client. Questo apre le strade a molte possibilità: ad esempio il server potrebbe informare i vari client con i dati provenienti da un sensore solo quando necessario. La cosa più importante è che questa caratteristica è ottenuta senza l'ausilio di plugin o componenti software esterne al browser;

Maggiore interazione client: altra caratteristica di innovazione è la scelta di aumentare le possibilità del client, dotandolo di maggiore interattività, e spostando operazioni solitamente svolte dal server su di esso. Il client, quindi, svolge un ruolo molto più importante del solito essendo la composizione delle pagine spostata dal server ad esso. È quindi il client a richiedere le pagine e combinarle.
Queste maggiori capacità del client aprono le strade ad una miriade di possibilità;

Ottimizzazioni della composizione dinamica: solitamente, in qualiasi applicazione web, le pagine vengono scaricate nella loro interezza continuando a prelevare anche parti che rimangono invariate. L'architettura proposta permette di assumere un approccio volto a modificare dinamicamente soltanto le parti necessarie risparmiando così potenza e banda sul server;

Separazione logica/presentazione/contenuti: le possibilità offerte da questa caratteristica sono molteplici e sono state esposte nella parte successiva. In ogni caso i benefici sono:

- maggiore manutenibilità;
- riutilizzo dei componenti;
- maggiore semplicità nella modifica;

- separazione dei ruoli.

Supporto a ruoli più complessi: una modifica nell'approccio verso i CMS richiesta dalla ditta era quella di estendere i ruoli di utenti e redattori. Solitamente esistono i redattori, i quali inseriscono i dati, e gli utenti che consultano ciò che è inserito dai redattori. Vi sono casi, come l'uso di un CMS in un'azienda, che prevedono un cambiamento di ruoli. L'utente non è più solo colui che consulta i dati, ma può anche inserire e/o modificare. Il ruolo del redattore, di conseguenza, diviene quello di creare e modificare le strutture dati volte a contenere ciò che è inserito dagli utenti.

4.1.2 Motivi Strategici

I motivi riportati in questa sezione riguardano la conoscenza e le necessità dell'azienda:

Conoscenza: attraverso l'esperienza di tesi l'azienda indende anche raccogliere know-how riguardo a tecnologie del campo dei CMS e soprattutto del web. Uno studio come questo (e relativa implementazione) comporta infatti un lavoro di ricerca che può migliorare le conoscenze aziendali nel settore;

Diminuzione tempi di sviluppo: un'esigenza di Leader.IT è quella di avere una soluzione versatile che permetta di diminuire i tempi di sviluppo mantenenendo la qualità a livelli elevati, non andando ad inficiare quindi sulla soddisfazione dei clienti;

Adattabilità ai cambiamenti: un'altra esigenza della ditti è quella di sfuggire ai continui cambiamenti forzati di molte tecnologie. Infatti ambienti quali .NET vengono modificati in continuazione determinando cosıuna svalutazione delle conoscenze acquisite e costringendo gli sviluppatori a investire tempo e denaro in formazione.

4.2 Schema dell'architettura proposta

L'architettura del sistema presentato è di tipo client-server. Internamente client e server sono suddivisi in blocchi ognuno dei quali svolge una specifica funzione. La scelta di creare un'architettura "a blocchi" è stata fatta per diversi motivi:

Modularità: tenendo completamente separate le varie parti del progetto ne è possibile la sostituzione (mantenendo ovviamente la stessa interfaccia di comunicazione tra blocchi): la semplicità di manutenzione aumenta notevolmente;

Interoperabilità: essendo separati i blocchi sono in grado di essere interrogati singolarmente, in questa maniera è possibile creare interfacce di accesso alternative (per esempio nel caso del data server si potrebbe esportare un'interfaccia SOAP per la richiesta dei dati);

Scalabilità: in una struttura a blocchi ogni parte può essere gestita da un server differente potendo così suddividere il carico, inoltre il fatto di gestire separatamente dati, logica e presentazione semplificherebbe la creazione di sistemi cluster poiché vi sono meno problematica da gestire;

Versatilità: una suddivisione dell'architettura in questa forma permette di avere maggiore flessibilità e possibilità nell'adattare la configurazione alle proprie esigenze, ad esempio quelle discusse nel punto precedente.

L'architettura proposta è illustrata nella figura 4.1: Una caratteristica che spicca immediatamente osservando la figura 4.1 è l'estrema somiglianza della configurazione di client e server. Ogni blocco (eccetto uno nel client ed uno nel server), infatti, ha il suo corrispondente nella controparte. Questa scelta è stata fatta per mantenere separata la gestione di dati, presentazione e logica: ogni blocco presente nel client può comunicare al server attraverso il gestore eventi od in maniera diretta. (La comunicazione tra le parti avviene in maniera asincrona. Questo è possibile tramite l'aggiunta di uno strato software nel client che, da un lato, gestisce la visualizzazione mentre dall'altro la comunicazione con il server) Tale decisione presenta notevoli vantaggi:

- maggiore semplicità nella modifica/sostituzione/manutenzione dei componenti utilizzati per implementare i comportamenti necessari ai siti progettati;
- il riutilizzo dei componenti è incentivato dalla separazione dati, logica, presentazione che permette di creare elementi in grado di gestire specifiche situazioni le cui parti sono slegate tra di loro (è possibile così implementare, ad esempio, un'area di testo per la visualizzazione dei dati, la quale se legata ad un "comportamento"' è in grado di gestire anche l'editing del contenuto);
- la separazione permette a chi usa questo progetto di creare i contenuti necessari per il proprio sito assegnando le varie parti a seconda delle

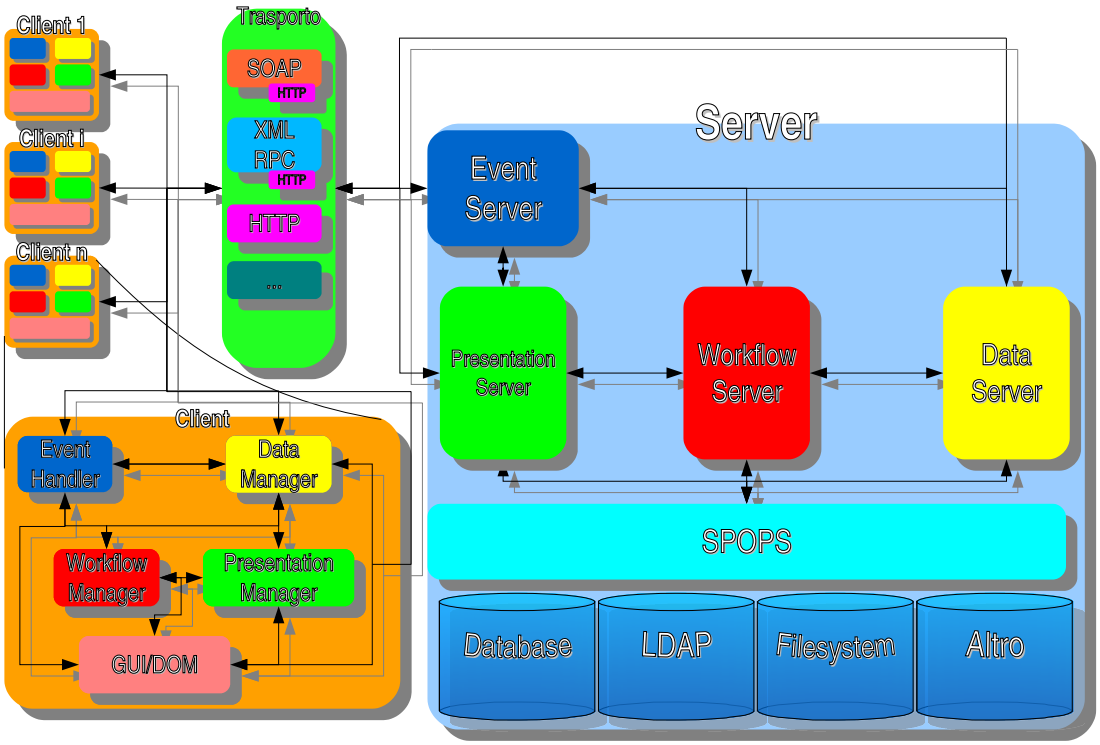

Figura 4.1: Struttura dell'architettura proposta

competenze (per esempio: il grafico si occupa della presentazione, il Database Administrator dei dati, il programmatore della logica).

Una cosa che vale la pena di notare è che la proposta presentata si pone come obiettivo di base quello di essere semplice. Per questo l'architettura è stata privata di tutto quello che non è indispensabile. Inoltre la funzione di composizione delle varie parti dei documenti, solitamente svolta dal server, viene in questo caso effettuata e gestita dal client. Questo permette di personalizzare al massimo i vari componenti, di alleggerire il server ed al client di comporre le pagine sfruttando sorgenti dati independenti tra loro.
All'interno dell'architettura proposta l'interazione tra i moduli è basata sugli eventi, dove per evento si intende l'arrivo di un messaggio che informa che è accaduto qualcosa (da un semplice click al timeout sulla connessione con il server). L'evento scatena una reazione da parte del gestore che agirà nella maniera programmata. Nel nostro caso sia il client che il server sono disegnati per agire in conseguenza di eventi. Nell'architettura proposta il client ed il server collaborano tra di loro scambiandosi eventi in base alle

necessità: ad esempio, se il client necessita di un dato invia un evento di tipo "richiesta dato" all'"Event Handler" il quale vedendo che è destinato al server lo inoltrerà a quet'ultimo. Successivamente dall'altro lato l'evento viene raccolto dall'"Event Server" il quale riconoscendo la richiesta dati la passa al "Data Server" che si occupa del recupero. Una volta che il dato è pronto viene agganciato ad un evento di tipo "dato trovato" il quale effettua il percorso inverso fino ad arrivare nel client per poter essere utilizzato. Ogni blocco sia sul client che sul server può inviare gli eventi, i quali verranno raccolti dall'"Event Handler", se ci troviamo nel client, dall'"Event Server" altrimenti. Questa possibilità permette ai vari blocchi di collaborare indirettamente (attraverso l'"Event Handler" o l'"Event Server") per ottenere il risultato desiderato, potrebbe e addirittura permettere ai client di interagire tra loro!
L'architettura è stata pensata in maniera che vi sia anche la possibilità per i moduli di una collaborazione diretta (senza l'ausilio degli eventi). Ciò potrebbe rivelarsi utile per esempio in una situazione in cui il client deve visualizzare dei contenuti statici (ad esempio immagini). In questa circostanza i dati possono essere richiesti e quindi ricevuti in maniera diretta saltando così alcune fasi e garantendo tempi di risposta minori.

Un'altra caratteristica che contraddistingue l'architettura presentata rispetto a quelle dei CMS già esistenti è la struttura del client. Solitamente quest'ultimo (essendo un browser) viene considerato un mero visualizzatore dimenticando che in realtà esso fornisce strumenti in grado di svolgere compiti ben più complessi. Spesso, infatti, il server viene programmato per fornire funzionalità (ad esempio la validazione di campi: un campo data deve avere la struttura GG/MM/AAAA) che sarebbero tranquillamente gestibili dal client. Al giorno d'oggi infatti i personal computer hanno una disponibilità di risorse sempre maggiore (talvolta anche esagerata per i compiti che devono svolgere) e spesso queste non vengono sfruttate (soprattutto in ambito web). Ciò porta ad avere una centralizzazione del carico di lavoro sul server rendendolo perciò un collo di bottiglia: se ad ogni richiesta (anche la più banale) viene coinvolto il server si avrà un sovraccarico di quest'ultimo e dei client totalmente "a riposo". È possibile evitare questa situazione demandando compiti che non devono essere necessariamente svolti dal server ai vari client. Questo tipo di approccio permette di trovare un equilibrio migliore e di ripartire il carico (dove possibile) tra client e server. La gestione di parte delle necessità effettuata dal client porta con sè tre vantaggi:

- diminuzione del carico del server con conseguente calo dei tempi di risposta su rete;

- utilizzo di una minore quantità di banda;
- maggiore velocità di reazione da parte del client (diminuzione di stress dell'utente).

La scelta di dotare il client di maggiori capacità rendendolo più interattivo apre le porte ad una miriade di possibilità, tra le quali:

- validazione dei contenuti in tempo reale;
- gestione di errori ed eccezioni senza l'ausilio del server (ad esempio in casi in cui venga a mancare la disponibilità della rete);
- possibilità di creare meccanismi complessi senza l'ausilio di plugin (Java[Jav], Flash[Fla], ...).

Verrano ora presentati singolarmente i componenti di client e server.

4.2.1 Client

Il client, come già accennato precedentemente, nella maggior parte dei casi e nelle varie implementazioni di CMS, non viene tenuto (o quasi) in considerazione, o meglio, viene ritenuto un componente meno importante. Al contrario nell'architettura presentata il client gioca un ruolo rilevante e viene rivalutato fornendogli capacità di agire in maniera autonoma. Per poter fare tutto ciò il client è stato rappresentato, progettato e considerato anch'esso come facente parte dell'architettura.
Ogni cosa all'interno del client è implementata tramite il linguaggio JavaSrcipt e l'ausilio del Document Object Model.
Una delle difficoltà maggiori che si hanno nell'utilizzo di JavaScript è costituita dalle differenze nelle implementazioni fornite dai browser. Purtroppo alcune di queste sono presenti sia nell'implementazione di JavaScript che in quella degli standard. Questo fattore rende necessaria la creazione di una sorta di "middleware" in grado di eliminare le differenze di comportamento/visualizzazione tra i vari browser così da creare un'implementazione il più compatibile possibile. La scelta di utilizzare JavaScript è stata fatta per alcuni motivi:

- è una tecnologia molto diffusa;
- viene eseguita direttamente dal browser, fornendo quindi performance migliori e minore occupazione di memoria rispetto ad altre soluzioni;
- essendo integrata nei browser non necessita di installazione di plugin.

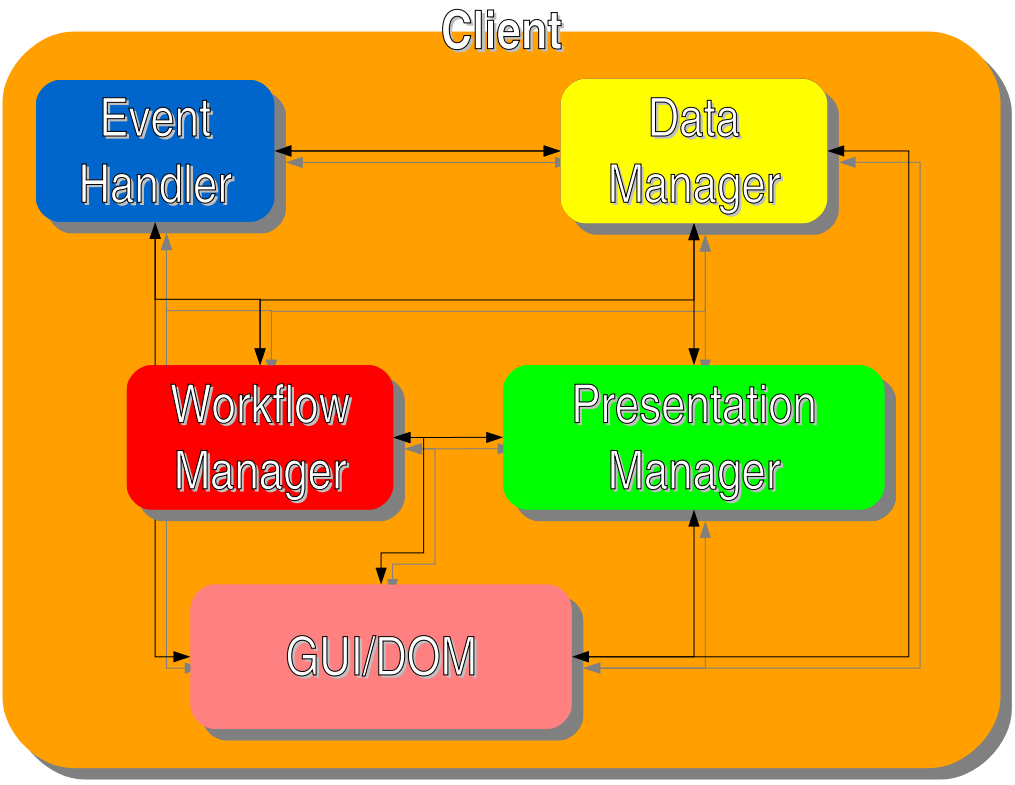

Figura 4.2: Struttura del client

La struttura del client, come si può notare dallo schema proposto, è suddivisa in 5 blocchi, che sono:

Event Handler: ha il compito di intercettare, smistare e gestire gli eventi;

Data Manager: si occupa del recupero e dell'invio dei dati (se modificati) da e verso il server;

Workflow Manager: permette di gestire comportamenti complessi all'interno del client;

Presentation Manager: si occupa della parte di presentazione, richiede e gestisce Widget (componenti per la visualizzazione del contenuto) compresa la parte di stile (CSS[CSS]);

GUI/DOM: attraverso questo blocco (fornito dal browser) è possibile interagire con il documento, i suoi contenuti ed il browser stesso. Ciò avviene sempre attraverso il linguaggio di scripting JavaScript.

Approfondisco ora questi cingue blocchi.

Event Handler

L'"Event Handler" svolge un ruolo centrale all'interno del client: il suo compito è quello di mediatore tra i vari blocchi del client e tra client e server. I moduli, infatti, collaborano tra loro nella maniera già esposta precedentemente, ovvero tramite lo scambio di eventi attraverso l'"Event Handler". All'interno del client gli eventi generati possono essere di due tipi:

Generati dall'utente: questo tipo di eventi viene lanciato in seguito all'interazione con il modulo GUI/DOM (click, spostamenti del mouse, pressione di tasti, ... in pratica gli eventi standard supportati dai browser). Essi permettono all'utente di utilizzare le funzionalità fornite dall'interfaccia, interagendo con essa tramite mouse e tastiera. In conseguenza di ciò il browser genera degli eventi, i quali ricevuti dall'"Event Handler" verranno gestiti nella maniera opportuna. Solitamente questo tipo di eventi viene gestito legando ai tag il codice per la gestione dello stesso. In questo caso, invece, la gestione viene centralizzata nell'"Event Handler" e ciò accade principalmente per tre motivi:

- **Comportamento differente browser:** la differenza di gestione degli eventi sta nel fatto che in un caso il browser inoltra l'evento dal componente più esterno del DOM verso quello più interno fino a che non viene trovato un listener in grado di gestirlo, nell'altro caso l'evento viene invece passato dall'interno verso l'esterno;
- **Separazione logica nel client:** essendo l'architettura orientata a suddividere dati, logica e presentazione, la centralizzazione dell'"Event Handler" permette di raccogliere tutti i listener per gli eventi in un unico luogo estrapolandoli così dall'html;
- **Unica copia codice gestione oggetti:** questo punto è meglio comprensibile tramite un esempio: supponiamo di avere una pagina che visualizza una serie di elementi tra di loro uguali attraverso Widget uguali, questi avranno anche lo stesso codice per la gestione. Quindi nel caso di una gestione degli eventi decentralizzata avremo il codice caricato per N volte una per ogni dato visualizzato. Una gestione centralizzata al contrario consente di mantenere un'unica copia del codice, utilizzabile da tutti gli elementi, con un conseguente risparmio di memoria.

Generati dai moduli: questo tipo di eventi viene invece generato per richiedere qualcosa, per collaborare con gli altri o per informare di un avvenimento (ad esempio: scadenza di un timer). Esso oltre che dai mo-

duli "Event Handler" è utlizzato anche da "Data Manager", "Workflow Manager" e "Presentation Manager" per una serie di necessità:

Recupero dal server: qualsiasi cosa un modulo necessiti di richiedere al server esso invia un evento contenente le informazioni necessarie per l'individuazione di ciò che va scaricato (esempio: richiesta di un dato da visualizzare al "Data Manager");

Invio al server: nel caso di necessità di invio al server un modulo spedisce un evento contenente le informazioni tramite le quali è possibili individuare il luogo in cui va posto il contenuto inviato (esempio: invio di un dato modificato per il salvataggio su DB);

Cambiamento di stato: nel caso in cui un modulo abbia necessità di informare un altro modulo di un avvenimento invia un evento contenente le informazioni raccolte all'"Event Handler" il quale si occuperà di gestirlo nella maniera corretta (esempio: scadenza di un timer);

Comunicazione altri moduli: nel caso in cui un modulo abbia necessità di comunicare ad un altro modulo invia un evento contenente le informazioni all'"Event Handler" il quale si occuperà di gestirlo nella maniera corretta (esempio: annullamento di un timer).

I compiti dell'Event Handler, come accennato precedentemente, sono tre e vengono eseguiti nell'ordine in cui sono qui elencati:

Intercettazione degli eventi: questo blocco interno al client per poter fornire i servizi necessari allo stesso dev'essere in grado di catturare gli eventi provenienti dai vari componenti (indipendentemente dal fatto che questi provengano dai blocchi del client, dal server o dall'interfaccia). Gli eventi provenienti dai blocchi del client verranno probabilmente notificati tramite una chiamata JavaScript all'Event Handler. Quelli provenienti dall'interfaccia verranno notificati legando una funzione per la loro gestione ai punti di inserimento forniti dal DOM. Infine gli eventi provenienti dal server passeranno attraverso il canale XMLHTTPRequest, fornito dal browser, sotto forma di XML (o JavaScript per aumentare le prestazioni);

Gestione eventi: alcuni eventi, come quelli dell'interfaccia, vengono elaborati in maniera diretta dal gestore che si occuperà di eseguire il codice

JavaScript necessario al loro trattamento. Questi vengono intercettati, analizzati, elaborati e se necessario smistati;

Smistamento eventi: altro compito svolto dall'Event Handler è quello di smistatore: è infatti un nodo centrale tra i blocchi, l'interfaccia ed il server. Esso deve quindi essere in grado di passare gli eventi o i risultati (i quali sono sempre eventi) al giusto componente. Agisce quindi come una sorta di "router" di eventi.

Data Manager

Il Data Manager è quel blocco che sul client si occupa della gestione dei dati. Vi sono vari casi in cui esso interviene:

Caricamento dati da server: il Data Manager interviene quando è necessario richiedere dei dati al server per qualsiasi motivo. Esso si occupa di preparare la richiesta ed inviarla nella maniera corretta;

Salvataggio dati su server: in questo caso il Data Manager raccoglie i dati da inviare e li "impacchetta" per l'invio a seconda dell'interfaccia utilizzata;

Riempimento interfaccia: una volta che i dati sono stati scaricati, il Data Manager ha anche il compito di riempire i campi presenti nell'interfaccia;

Lettura dati da interfaccia: nel momento in cui è necessario salvare i dati appena modificati il modulo Data Manager legge i campi dall'interfaccia facendo uso del DOM.

Possiamo separare logicamente alcune funzionalità del Data Manager:

Comunicazione: un sottoblocco all'interno del Data Manager si dovrà occupare delle comunicazioni le quali dovranno avvenire in maniera trasparente rispetto al trasporto. Infatti il blocco in analisi sarà in grado di richiedere i dati o tramite eventi od in maniera diretta;

Recupero/invio dati da/a sorgente remota: un secondo sottoblocco del Data Manager avrà l'incarico di gestire la richiesta e l'invio dei dati al server remoto. Per questioni di efficienza si potrebbe implementare all'interno del Data Manager una cache che questo blocco dovrebbe consultare prima di inoltrare la richiesta remota;

Mappa "Oggetto ↔ Widget": questo sottoblocco è necessario per ottenere un riferimento al componente grafico che visualizza l'oggetto che si sta cercando (e viceversa). È necessario per l'interazione tra DataManager e GUI;

Lettura/Scrittura dati da/a GUI: questo sottoblocco viene sfruttato nel momento in cui si devono visualizzare o leggere (per l'invio) dei dati dall'interfaccia grafica. L'interazione con l'interfaccia avviene tramite il DOM e la mappa e con questi strumenti è possibile, per questo sottoblocco, andare a riempire i vari campi che costituiscono l'oggetto e leggerli per poterli inviare al server;

Cache: questo componente ha la funzione di salvare temporaneamente i dati scaricati dal server. Esso infatti potrebbe mantenere un oggetto con il relativo timestamp ed andare a leggere dal server l'ultima data di modifica del dato. Se la cache risulta ancora aggiornata il Data Manager anzichè passare la richiesta al server può ripescare i dati in locale e visualizzarli risparmiando tempo e banda.

Per chiarire meglio i compiti del Data Manager vediamo un esempio suddiviso in fasi:

Richiesta dato:

- l'Event Handler inoltra un evento di "richiesta dato" al gestore dati accompagnato da un riferimento o dall'id del widget nel quale il dato verrà visualizzato;
- il sottoblocco preposto alla comunicazione riceve l'evento e vedendo che è di tipo "richiesta dati" lo inoltra al sottoblocco di recupero dati;
- successivamente questo scompone la richiesta e controlla nella cache per verificare di non averlo già:
  - se non lo trova prepara la richiesta e la passa al modulo di comunicazione il quale la serializzerà nella maniera necessaria alla forma di comunicazione prescelta;
  - se lo ha già non fa nulla.

Ricezione dato:

- dopo aver ricevuto il dato (dalla cache oppure dal server) il Data Manager passa questo e il riferimento al componente in cui dev'essere visualizzato al blocco che interagisce con la GUI;
- questo infine si prenderà cura di inserirlo nel widget corretto.

Invio dato:
- in questo caso dall'Event Handler arriva una richiesta di invio dati contentente il riferimento al widget nel quale è contenuto il dato da inviare;
- quindi il componente di comunicazione inoltra la richiesta al sottoblocco che interagisce con l'interfaccia;
- questo facendo uso dei dati contenuti nella mappa trova l'oggetto da inviare;
- legge i dati e li confronta con quelli che ha in cache:
  - se ci sono dati modificati soltanto questi verranno impacchettati;
  - se nessun dato è stato modificato, viene annullata la richiesta di invio;
- i dati impacchettati vengono serializzati dal componente che gestisce le comunicazioni e quindi inviati nella maniera adeguata.

Workflow Manager

Il blocco di gestione dei workflow ha una struttura piuttosto semplice. Infatti esso non svolge che un compito: eseguire i workflow (presentati successivamente). Nonostante la semplicità strutturale, le funzionalità svolte da questo blocco sono piuttosto complesse. Esso permette la definizione di comportamenti articolati controllati nel tempo.
Il Workflow Manager contiene una serie di "frammenti" di codice i quali collegati tra loro danno luogo ai workflow.
Il funzionamento del workflow è praticamente lo stesso sia sul client che sul server. Nel momento in cui viene intercettato un evento che si sa essere gestito tramite un workflow viene passato il controllo al blocco. Questo controlla lo stato in cui si trova il workflow in questione e secondo le possibilità esistenti verifica le condizioni per passare nello stato seguente. Solitamente è possibile associare alle transizioni delle azioni.
Un'altra caratteristica utilizzabile è la possibilità del server di essere informato al raggiungimento di un certo stato. Ciò permette di collegare un workflow del client con uno del server ottenendo quindi una versatilità notevole.
Questo blocco si suddivide in due sottoblocchi:

Comunicazione: ha la consueta funzionalità di comunicare con gli altri elementi dell'architettura in maniera trasparente rispetto al trasporto;

Esecuzione workflow: si occupa dell'esecuzione del workflow. Esso tiene traccia di tutto il codice necessario e dello stato in cui ci si trova.

Presentation Manager

Il Presentation Manager è quel blocco che si occupa della gestione di tutto ciò che riguarda la presentazione. Esso ha alcuni compiti e si suddivide in una serie di sottoblocchi:

Comunicazione: questo sottoblocco ha la stessa funzionalità che svolge negli altri blocchi: serializzare e deserializzare le richieste in maniera trasparente rispetto al trasporto;

Layout: il compito svolto da questo oggetto è quello di gestire la disposizione a video dei widget. Inoltre esso diviene utile nei casi in cui la dimensione della finestra viene modificata. In questa situazione, infatti, vi è la necessità di adattare le dimensioni dei componenti a video e magari di spostarli per mantenere la finestra utilizzabile (nei limiti del possibile). Il Layout Manager gestisce il layout tramite l'utilizzo delle tag *div* e delle tag *span* entrambe facenti parte dello standard XHTML. Questi elementi permettono di suddividere la pagina in blocchi. Con div questi vengono disposti in maniera verticale mentre con span si ottiene un allineamento orizzontale. Quindi tramite una composizione dei due tipi di elementi si riescono ad ottenere strutture anche piuttosto complesse. Il Layout Manager sfruttando queste tag ed il CSS per specificarne le proporzioni, e le dimensioni si pone l'obiettivo di mantentere la pagina chiara e leggibile;

Widget: questo sottobloccosi occupa di un compito molto importante: la gestione del caricamento e la visualizzazione dei widget. Esso utilizza principalmente due meccanismi per ridurre il più possibile il consumo di banda:

Composizione Top-Down: è un approccio nello scaricamento e quindi nella composizione dei widget. Essendo il widget costituito da una serie di altri widget più semplici è possibile procedere con lo scaricamento nella seguente maniera. Il server, anziché passare il codice XHTML per la visualizzazione del widget, fornisce al client l'identificativo del widget che dev'essere rappresentato. Il Widget Manager quindi verifica nella cache l'eventuale presenza del codice necessario alla visualizzazione del componente e procede in questa maniera:

- se il codice è già presente nel client il widget viene mostrato a video;

- in caso contrario il Widget Manager non conosce la struttura del componente e quindi chiede che gli venga descritto ad un livello di dettaglio maggiore. Questo accade fino a che non si arriva al livello di dettaglio massimo.

La composizione viene svolta in questa maniera per un semplice motivo: l'invio del solo identificativo o degli identificativi permette di utilizzare poca banda e allo stesso tempo massimizza l'utilizzo della cache. Il prezzo da pagare è un piccolo overhead dovuto allo scaricamento degli id. È tuttavia possibile abbassare questo costo. Un altro pregio di questo approccio deriva dal fatto che due o più widget possono avere come sottocomponenti oggetti visuali già scaricati dal server. Il vantaggio è quindi comune a tutti i widget.

Riutilizzo componenti: un'altra tecnica per ridurre il peso dello scaricamento è applicabile in quei casi in cui si debbano rappresentare delle liste di oggetti tutti uguali. Il vantaggio sta proprio nel fatto che questi siano uguali. Infatti, grazie a ciò, anziché scaricare nuovamente tutto il codice per la rappresentazione di questi oggetti, è possibile affidare al Widget Manager il compito di replicare l'XHTML un numero di volte pari alla lunghezza della lista;

Style: questo sottoblocco si occupa della gestione dello stile delle pagine, in altre parole di gestire i fogli di stile. Tramite questo componente, è possibile andare a modificare una qualsiasi caratteristica riguardante la visualizzazione. Per ottenere maggiore chiarezza vediamo un esempio: supponiamo che vi sia la necessità di enfatizzare dal punto di vista grafico il fatto che stiamo modificando un campo. Per fare ciò solitamente nelle applicazioni che siamo soliti utilizzare viene cambiato il colore di testo e/o sfondo. Anche in questo caso è possibile ottenere questo variazione, grazie all'utilizzo del gestore dello stile. Quindi ogni compito legato al "come" viene visualizzata la grafica (font, colori, bordi, ...) è gestito da questo sottoblocco;

cache: l'ultimo componente presente all'interno del Presentation Manager è la cache. Questa, alla stessa maniera di quella presente nel Data Manager, viene utilizzata per salvare un insieme di dati in maniera da evitare di doverli nuovamente scaricare.

Una caratteristica piuttosto importante è la capacità del Presentation Manager di visualizzare anche oggetti sprovvisti di widget. In questa maniera è possibile interfacciare il sistema direttamente ad una base dati ed ottenerne una visualizzazione. Questa possibilità deriva dal fatto che essi sono costruiti

in maniera composizionale: si parte da alcuni widget di base ed assemblandoli si ottiene qualcosa che rappresenta l'oggetto desiderato. Ciò è garantito dalla possibilità di leggere il tipo dei campi contenuti nell'oggetto e avendo queste informazioni è quindi possibile comporre un widget adeguato.

GUI/DOM

Questo componente è l'unico dei cinque presentati ad essere fornito nativamente dal browser. Il DOM è stato inserito nell'architettura in quanto:

- fornisce un'interfaccia per l'interazione con il browser e con il documento visualizzato:
  - permette l'implementazione dei comportamenti dinamici;
  - permette la modifica al volo delle pagine;
- è fondamentale per la visualizzazione del risultato prodotto dal server. In sua assenza ogni altro blocco dell'architettura sarebbe inutile;
- tramite questo componente viene mantenuto lo stato in cui si trova il sistema. Questo è necessario per l'utilizzo dei workflow, che come abbiamo visto, si basano sullo stato del sistema in un dato momento.

4.2.2 Server

Il server è da sempre la parte principale delle applicazioni web ma in questo caso esso gioca un ruolo un po' più marginale. Anche il server alla stessa maniera del client è suddiviso in cinque macroblocchi. Ogni blocco, a sua volta, ha una particolare architettura interna.
Il server è incaricato dello svolgimento di una serie di compiti:

Persistenza: appoggiandosi ad uno o più backend dati esso si occupa di mantenere i dati e tutto ciò che è necessario per il funzionamento dell'applicazione in uno stato persistente. Infatti nel modello presentato si può notare come ogni blocco sia collegato alla sorgente dati dalla quale vengono letti gli elementi necessari;

Coordinamento: è un compito piuttosto importante necessario nei casi in cui si ha a che fare con più client. Infatti in queste situazioni sarebbe facile incorrere in problemi dovuti a interventi concorrenti da parte di più client.
Supponiamo, ad esempio, che due client inviino una modifica nello stesso istante per lo stesso oggetto. Se l'azione non venisse coordinata

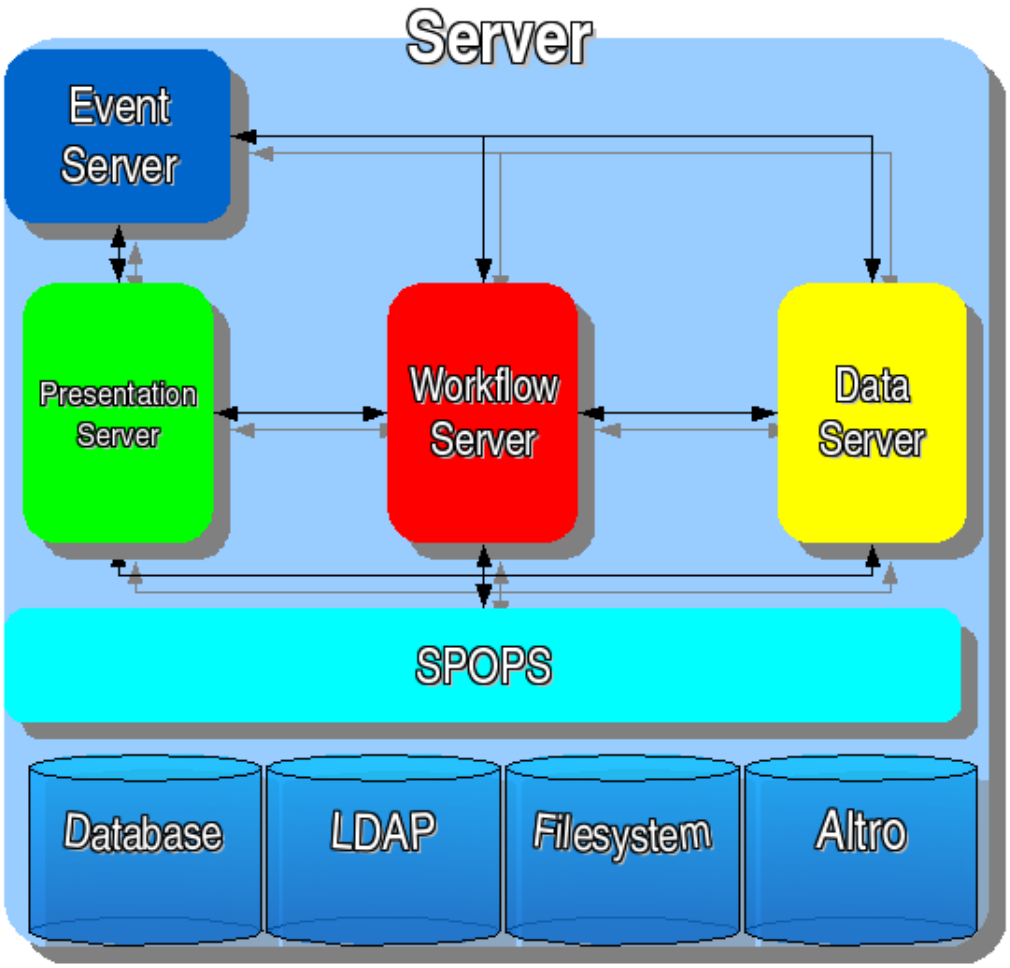

Figura 4.3: Struttura del server

porterebbe ad una corruzione dei dati. Coordinando gli interventi dei client è possibile evitare il problema.

I compiti integrati nell'architettura non sono molti. Si è infatti spinto il più possibile per creare un'architettura priva di tutto ciò che non è essenziale in modo da mantenere la maggiore versatilità possibile. Infatti funzionalità come la gestione degli accessi (pur essendo supportati dalla libreria di persistenza degli oggetti) non sono state integrate nell'architettura. In questa maniera chiunque può scrivere questi blocchi a seconda delle proprie necessità. È prevista in ogni caso la realizzazione di componenti general purpose da utilizzare per chi non ha esigenze particolari.
Internamente al server i moduli comunicano tra loro attraverso eventi come all'interno del browser. Il particolare disegno dell'architettura permette ai

vari blocchi di risiedere su macchine fisicamente separate.
Per quanto riguarda la parte server le tecnologie utilizzate sono:

Perl: la scelta del suo utilizzo nel progetto è derivata principalmente dal fatto che Leader.IT ha molto investito in esso e che la maggior parte dei sofware sviluppati dalla ditta sono scritti in questo linguaggio. Due motivi secondari per fare questa scelta potrebbero essere:

Versatilità: la breve presentazione fatta nella sezione 3.3.1 ha illustrato (in line di massima) le potenzialità di questo linguaggio che grazie alla sua ricchezza rende semplici compiti complessi;

Disponibilità di librerie: nell'archivio di Perl, il già citato CPAN, sono disponibili una moltitudine di librerie potenzialmente utilizzabili nel progetto.

Mason: questo framework viene utilizzato per la parte riguardante i widget. Esso ha la particolarità di poter comporre oggetti o pagine in maniera incrementale partendo da quelli che in Mason vengono definiti "componenti".
Esso è stato scelto in quanto utilizzato anche per la precedente implementazione della libreria e si è dimostrato un prodotto stabile e robusto.

POE: è un framework molto completo ed elaborato per la creazione di software concorrenti guidati da eventi. Esso fornisce una solida base ed è stato utilizzata anche in progetti piuttosto elaborati.
Questo era stato individuata precedentemente, in occasione di un altro progetto, da Leader.IT. Dall'analisi effettuata POE è risultato essere un prodotto completo e valido.

JSON: è una delle possibili forme di comunicazione tra client e server. Questo protocollo permette di comunicare in maniera "nativa" con il client (inviando codice JavaScript) e di inviare le strutture dati necessarie. È stato scelto perché uno degli obiettivi del progetto è la velocità. Infatti inviando i dati in altre forme spesso si hanno carenze in velocità anche se l'interoperabilità ottenuta è maggiore.
Il protocollo è stato individuato e scelto inoltre per la sua estrema semplicità e perchè, fatto questo molto importante, dispone del supporto per il linguaggio Perl (e per molti altri).

I componenti costituenti il server sono:

Event Server: ha il compito di raccogliere e gestire in maniera coordinata gli eventi provenienti da tutti i client e dai vari blocchi che risiedono nel server;

Presentation Server: si occupa degli aspetti riguardanti la presentazione, ovvero del recupero di widget, stili e layout per l'invio al client. Supporta inoltre il meccanismo per la composizione top-down dei widget illustrato nel client.

Workflow Server: permette di gestire comportamenti complessi e di codificare procedimenti elaborati in maniera semplice e schematica;

Data Server: questo blocco gestisce tutto ciò che riguarda i dati, il loro invio al client, la loro ricezione per la modifica e i permessi di accesso;

SPOPS: questo è il blocco, già esistente come framework, che gestisce la persistenza degli oggetti.

Verranno ora illustati in maniera più approfondita i vari componenti.

Event Server

L'Event Server gioca per il server un ruolo fondamentale come l'Event Hanler per il client. Esso è il gestore di tutte le comunicazioni che avvengono tra i blocchi del server e tra i client ed il server. Quindi il suo compito è quello di coordinatore e gestore di tutto ciò che accade sul server e degli eventi provenienti dal client. Esso fornisce anche un punto di entrata per la gestione degli eventi tramite workflow.
L'Event Server è suddiviso in alcuni sottoblocchi:

Comunicazione client: l' utilizzo di POE permette di gestire i vari client tramite il meccanismo delle sessioni ognuna delle quali fa affidamento su un certo driver per la comunicazione. In questa maniera è possibile, teoricamente, gestire nello stesso momento client che "parlano diverse lingue": POE ascolta le comunicazioni ed avvia in maniera automatica una sessione in grado di gestire il client la quale viene mantenuta fino alla disconnessione di quest'ultimo. In questo caso devono essere programmate le sessioni per la gestione della ricezione e dell'invio di eventi al client.
Un aspetto importante da tenere in considerazione in questo caso è quello della sicurezza: è necessario prevedere l'utilizzo di tecniche di

crittografia per proteggere l'accesso e garantire la riservatezza dei dati in transito.

Event Router: questo sottoblocco ha lo stesso scopo che aveva nel client: lo smistamento degli eventi tra i vari "attori" coinvolti. Esso sulla base del destinatario inoltra gli eventi in transito.

Comunicazione blocchi: da questo lato avvengono le comunicazioni da e verso i blocchi interni del server. Questo sottoblocco gestisce il traffico da, verso e tra i blocchi: per suo tramite gli eventi destinati ai client, quelli provenienti dai client e quelli scambiati tra i blocchi vengono gestiti e, se necessario, passati all'Event Router.

Come si può notare la struttura di questo blocco rispecchia da vicino quella del client. La maggiore differenza risiede nel fatto che lato server è necessario gestire le connessioni ai vari client mentre questi ultimi ne devono mantenere soltanto una.

Presentation Server

Il Presentation Server si occupa di gestire il recupero dei dati necessari alle funzioni di presentazione. Esso supporta quindi le seguenti funzionalità:

Comunicazione: presente nei vari moduli questo sottoblocco permette di interagire con gli altri blocchi;

Widget: il sottoblocco fa utilizzo di Mason e delle sue caratteristiche per la composizione delle varie parti. Una delle particolarità per cui viene sfruttato Mason è la capacità di combinare vari componenti per ottenere un risultato unico. Questa funzionalità, combinata alla possibilità di utilizzo dell'ereditarietà tra componenti, garantisce la versatilità richiesta dal meccanismo dei widget.
I widget sono stati progettati in maniera da fornire molte possibilità:

Composizione: i widget possono essere composti a partire da elementi più semplici. Ciò consente, in una maniera simile a quella sfruttata dai RAD, (Rapid Application Developement) di creare dei widget i quali potranno poi essere base per altri e così via. Questo meccanismo permette la generazione di widget molto complessi, anche conoscendo soltanto i tipi di dato degli attributi di un oggetto. Le informazioni rigurdanti gli oggetti vengono estrapolate dal sistema di persistenza ed in questa maniera è possibile

creare un widget adatto alla loro visualizzazione. Tutto ciò unito al meccanismo di caching integrato in Mason, permette di creare widget particolari in maniera piuttosto rapida;

Specializzazione: Mason come abbiamo visto permette di sovraccaricare i componenti creati. Questo fattore favorisce fortemente il riutilizzo dei widget. Il motivo di ciò è presto spiegato: avendo a che fare con un'architettura che gestisce i dati sotto forma di oggetti è facile dedurre che i widget stessi non sono altro che oggetti visuali che ricalcano quelli contenenti i dati. Quindi, avendo questo tipo di organizzazione, è possibile sovraccaricare non soltanto gli oggetti contenenti i dati ma anche quelli grafici. Attraverso questi meccanismi anche i widget ereditano tutti quelle caratteristiche che rendono la programmazione ad oggetti un'idea vincente;

Separazione ruoli: un altro fattore che aumenta fortemente versatilità e riutilizzo è la separazione dei ruoli mantenuta a livello progettuale (contenuto/presentazione/logica). A prima vista potrebbe sembrare una cosa di poca importanza ma l'esperienza avuta dall'adozione delle tecniche di separazione contenuto/presentazione evidenzia i vantaggi di questa scelta. Solitamente essa era applicata a due componenti (contenuto e presentazione) mentre in questo caso mirando il progetto a rendere il client fortemente interattivo, viene introdotto un terzo componente: la logica. La separazione degli aspetti, quindi, abbraccia anche questa parte e da essa derivano vantaggi notevoli:

Versatilità: essendo separati i vari pezzi che compongono un widget possono essere combinati nelle più svariate maniere. Gli unici vincoli per la realizzazione di ciò sono la necessità di avere un'interfaccia comune ed, in caso di sostituzione del comportamento, compatibilità con la struttura dell'oggetto di visualizzazione;

Riutilizzo: anche a livello dei widget la separazione incrementa le possibilità di riutilizzo. Infatti mantenendo la stessa interfaccia è possibile riutilizzare oggetti, o loro parti, più volte risparmiando conseguentemente tempo e fatica;

Manutenibilità: un altro fattore migliorato dall'adozione della separazione è la manutenibilità. Ciò deriva dal fatto che non ci si deve preoccupare di modificare le altre parti quando se ne modifica una. Infatti mantenendo l'interfaccia uguale non ci si deve curare di aspetti appartenenti ad altri elementi;

Adattabilità output: una caratteristica che è possibile fornire sempre grazie a Mason è l'adattabilità dell'output. Questo concetto verrà meglio illustrato grazie ad un esempio. Supponiamo di avere la necessità di visualizzare lo stesso widget in due formati differenti: XHTML e XUL (nativo di Mozilla di cui è stato fatto cenno poco sopra). Descrivendo i widget attraverso un linguaggio intermedio è possibile, grazie alle capacità dinamiche ed ai filtri di Mason, adattarlo in base alla destinazione. Questa capacità è utile anche nel caso in cui si abbia a che fare con dispositivi differenti (dispositivi mobili, ...);

Lettura dati: i dati vengono prelevati da SPOPS che li mantiene persistenti in uno dei backend che supporta. In questo caso i dati sono gli oggetti che rappresentano stili, widget e layout.

Workflow Server

Il Workflow Server è il blocco che permette di codificare e quindi svolgere compiti piuttosto complessi. In generale tramite workflow è possibile descrivere due tipi di processi:

Necessari all'utente: questa tipologia di processo permette di aiutare l'utente a svolgere un compito nella maniera corretta (ad esempio è possibile imporre la revisione di un articolo prima della sua pubblicazione);

Necessari all'applicazione: questo tipo di processo, a differenza del precedente, è necessario al funzionamento dell'applicazione e serve per modellare comportamenti complessi in maniera da renderne il riutilizzo in altri ambiti molto semplice. (in questo caso l'esempio potrebbe essere la gestione di un timeout oppure quella di una sessione)

Dopo aver illustrato i tipi di processi modellabili tramite workflow vediamo in che maniera è disegnata l'architettura interna al modulo:

Comunicazione: come in tutti gli altri casi questo sottoblocco si occupa della comunicazione verso l'esterno;

Esecuzione workflow: si occupa di eseguire il workflow. In questo sottomodulo possono esistere ed girare contemporaneamente diverse macchine a stati, ognuna delle quali si occupa di un workflow.
Per poter comprendere meglio come avviene l'esecuzione di un workflow bisogna conoscerne la struttura: un workflow non è altro che un sistema il quale permette di modellare un comportamento attraverso un grafo. In pratica esso è costituito da tre elementi di base:

Stati: gli stati non sono altro che le situazioni in cui il worflow si può venire a trovare. Detto in maniera più rigorosa gli stati sono i nodi del grafo che rappresenta il workflow;

Condizioni: le condizioni vengono poste sui rami che connettono i vari stati tra loro. Esse regolano il percorso seguito dall'esecuzione del workflow. Se una condizione è verificata all'interno di uno stato è possibile passare in un altro stato seguendo la transizione che li connette;

Azioni: queste vengono associate alle transizioni, quindi il passaggio da uno stato all'altro può significare l'esecuzione di una certa azione;

Ricapitolando il workflow non è altro che un grafo i cui nodi rappresentano gli stati in cui il processo descritto si può trovare. I nodi sono connessi tra loro per mezzo di transizioni (archi) sui quali possono essere poste delle condizioni che favoriscono od impediscono il passaggio. Inoltre alle transizioni è possibile legare delle azioni eseguite nel passaggio da uno stato ad un altro.
Esistono anche altre caratteristiche non essenziali ma comunque utili nel campo dei workflow:

Validatori: essi permettono la verifica della validità dei dati durante le azioni;

Osservatori: danno la possibilità di "osservare" l'accadimento di un particolare evento nel workflow. In pratica l'osservazione funziona alla rovescia ovvero: non è l'interessato che guarda ma è il motore di esecuzione del workflow che informa quello che si è registrato come osservatore.

Un particolare su cui vale la pena di fare luce è il meccanismo con il quale è possibile collegare tra loro eventi e workflow. Questo può essere suddiviso in due tipi di interazione:

Evento → Workflow: in questo caso l'arrivo di un evento è gestito dal workflow. Il meccanismo tramite il quale viene gestito è piuttosto semplice: a livello di programmazione viene stabilito che un certo evento deve scatenare il proseguimento dell'esecuzione del workflow. Quindi la macchina a stati viene informata al momento dell'arrivo dell'evento in maniera che venga gestito nel modo stabilito;

Workflow → Evento: in questo caso è possibile distinguere due situazioni:

Generazione evento: l'esecuzione del workflow, in particolare un'azione, genera un evento che è semplicemente inviato per svolgere un compito particolare;

Connessione workflow: rappresenta in realtà un caso particolare del punto precedente. Nell'architettura è stata prevista la possibilità di collegamento tra workflow di server e client realizzata facendo uso di un evento e di un osservatore. In pratica, nel punto di connessione tra i workflow viene registrato un osservatore. Questo genererà un evento che informerà il client del passaggio di controllo da parte del server (o il contrario).

Lettura dati: questo sottoblocco si occupa delle operazioni di lettura dei workflow serializzati nel backend. Oltre al workflow stesso esso legge anche lo stato in cui si trova in maniera da poterne correttamente riprendere l'esecuzione.

Data Server

Questo blocco è infine quello che si occupa della ricezione e dell'invio dei dati e della loro lettura dal db. Esso ha il compito di scrivere e leggere i dati dal backend (sempre passando per SPOPS).
Come gli altri blocchi presentati anch'esso si suddivide in sottoblocchi:

Comunicazione: si occupa di comunicare con gli altri blocchi. Offre anche le interfacce di interrogazione alternative, ad esempio, attraverso SOAP o attraverso l'uso di un URL.

Lettura/Scrittura dati: si occupa di effettuare le richieste al backend, siano queste di scrittura o di lettura. Inoltre ove necessario si preoccupa di adattare il formato della richiesta al linguaggio utilizzato dal backend prescelto. Esso sfrutta le ACL fornite da SPOPS per regolare l'accesso ai dati, informando nel caso in cui vi siano dei problemi. Appoggiandosi su SPOPS non necessita di effettuare operazioni complesse e da questo deriva la sua semplicità.

SPOPS

SPOPS è il framework utilizzato per la persistenza dei dati. Esso fa da interfaccia per la serializzazione in maniera persistente e successivo recupero

dei dati. Nell'architettura viene utilizzato per salvare gli elementi necessari a tutti i blocchi, difatti ogni modulo ha il proprio sottomodulo per la consultazione dei dati. Grazie a SPOPS si ha la possibilità di utilizzare in maniera trasparente vari backend, si ha la gestione integrata dei permessi e la versatilità del framework.

Capitolo 5

Conclusioni

Gli obiettivi di questo lavoro di tesi sono stati:

- l'analisi dello stato dell'arte dei CMS;
- l'analisi degli strumenti utilizzati;
- la definizione di una nuova architettura per CMS basata sugli eventi.

Il lavoro di raccolta delle informazioni è stato molto impegnativo. Una delle difficoltà maggiori infatti è stata quella di individuare informazioni valide poiché, riguardo ai CMS, è possibile trovare molte fonti, ma spesso incomplete o non troppo approfondite.
La fase centrale del lavoro ha riguardato la definizione della nuova architettura. Tale fase ha visto un'interazione intensa con l'azienda Leader.IT per discutere e capire i requisiti e la struttura ottimale per la proposta di architettura basata sugli eventi. Nel breve termine il progetto verrà portato avanti ed implementato all'interno dell'azienda. Data la versatilità dell'architettura il risultato che si vuole ottenere è quello di costruire un web application server sul quale appoggiare un ambiente RAD (Rapid Application Development) utilizzabile via web per la costruzione di applicazioni web.
Nel medio termine si pensa anche di lavorare per l'integrazione all'interno dell'architettura delle tecnologie dei web service e del supporto necessarrio per l'introduzione del semantic web.

Bibliografia/Riferimenti

[AJA] AJAX: http://www.adaptivepath.com/publications/essays/archives/000385.php articolo di riferimento per la tecnologia ajax.

[Apa] Apache: http://apache.org/ sito ufficiale del webserver apache.

[BLCL+94] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world-wide web. *Commun. ACM*, 37(8):76–82, 1994.

[Bri] Bricolage: http://bricolage.cc/ sito ufficiale del cms bricolage.

[CMS] CMSMatrix: http://www.cmsmatrix.org sito che riporta le caratteristiche di molti cms permettendo anche di fare dei confornti.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387, 1970.

[CPA] CPAN: http://www.cpan.org/ sito dell'arhivio di moduli perl cpan.

[CSS] CSS: http://www.w3.org/style/css/ sito ufficiale per lo standard dei css (cascading style sheets).

[DOMa] DOM: http://www.w3.org/dom/ dom pagina di riferimento per il document object model.

[DOMb] Articolo sul DOM: http://xml.coverpages.org/dom.html articolo che tratta del dom.

[ECM] ECMA: http://www.ecma-international.org/ sito ufficiale della europead computer manufacturer association (ecma).

[Fed] Fedora Directory Server: http://directory.fedora.redhat.com/wiki/main_page sito ufficiale del directory server di fedora (ex netscape directory server).

[Fla] Flash: http://www.macromedia.com/it/software/flash/ sito ufficiale della piattaforma flash di macromedia.

[HP] Hewlett Packard: http://www.hp.com/ sito ufficiale di hewlett packard.

[IIS] IIS: http://www.microsoft.com/windowsserver2003/iis/default-.mspx sito ufficiale del webserver iis.

[Jav] Java: http://java.sun.com/ sito ufficiale del linguaggio java.

[JSO] JSON: http://www.crockford.com/json/xml.html introduzione a json un protocollo di interscambio dati leggero.

[Lat] LaTeX 2_ε: http://www.latex-project.org/ sito del linguaggio di markup per la produzione di documenti bibliografici di alta qualità.

[Masa] Mason: http://masonhq.com/ sito ufficiale del motore di templating mason.

[Masb] Libro Mason: http://www.masonbook.com/book/chapter-1.mhtml versione online del libro su mason.

[MSS] MS SQL Server: http://www.microsoft.com/sql/default.mspx sito ufficiale del dbms microsoft sql server.

[MyS] MySQL: http://www.mysql.com/ sito ufficiale del dbms mysql.

[Ope] OpenLDAP: http://www.openldap.org/ sito ufficiale del directory server openldap.

[Ora] Oracle: http://www.oracle.com/database/index.html sito ufficiale del dbms oracle.

[Per] Perl: http://www.perl.org/ sito ufficiale del linguaggio perl.

[Plo] Plone: http://plone.org sito ufficiale del cms plone.

[POEa] Introduzione a POE: http://www.perl.com/pub/a/2001/01/poe-.html articolo introduttivo all'uso di poe.

[POEb] POE: http://poe.perl.org/poedown/poe-whitepaper-a4.pdf whitepaper del framework poe.

[Pos] PostgreSQL: http://www.postgresql.org/ postgresql sito ufficiale del dbms postgresql.

[Pyt] Python: http://python.org/ sito ufficiale del linguaggio python.

[RDF] RDF Primer http://www.w3.org/tr/rdf-primer/ rdf primer documento di specifica di rdf.

[RDQ] RDQL: http://www.w3.org/submission/2004/subm-rdql-20040109/ proposta del linguaggio rdql come standard per l'interrogazione di database rdf.

[RW02] Dave Rolsky and Ken Williams. *Embedding Perl in HTML with Mason.* O'Reilly, first edition, 2002.

[Spi] SpiderMonkey: http://www.mozilla.org/js/spidermonkey/ pagina ufficiale dell'engine javascript di mozilla spidermonkey.

[SPO] SPOPS: http://spops.sourceforge.net/ sito ufficiale del sistema di persistenza di oggetti perl spops.

[TWi] TWiki: http://twiki.org sito ufficiale del cms twiki.

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl.* O'Reilly, third edition, 2000.

[Wik] WikiPedia: http://www.wikipedia.org/ enciclopedia multilingue distribuita online sotto la gnu fdl (free documentation license).

[Zop] Plone: http://zope.org sito ufficiale dell'appication server zope.