Università degli Studi
del Piemonte Orientale

Facoltà di Scienze
Matematiche, Fisiche, Naturali
Sede di Alessandria

Corso di Laurea in
Informatica

Tesi di Laurea

Realizzazione di un Portale Web Aziendale con Tecnologia JSR 168

Tutore interno
Prof. Luigi Portinale

Tutore esterno
Ing. Mauro Gagni

Candidato
Guido Vicino

Anno Accademico 2003/2004
Sessione di Dicembre

A tutti coloro
che mi hanno aiutato
a compiere
questo primo passo.

Indice

Capitolo 1
IBSP ed Enterprise Portals

Negli ambienti lavorativi moderni è diventato sempre più importante disporre di mezzi tecnologici tramite i quali velocizzare e gestire la ricerca d'informazione.

Se un'azienda può disporre più velocemente di dati, documenti e altri fonti informative può migliorare il suo guadagno e le prestazioni fornite al cliente.

Uno strumento sempre più presente è il Portale Enterprise, che si propone come valido assistente per la risoluzione dei problemi precedentemente elencati.

La IBPS (ICON Business Process Solution) è un azienda specializzata da oltre dieci anni nell' Information and Communication Tecnology e nella fornitura di tecnologie hardware e software innovative.

Il gruppo ha come missione quella di proporsi come preparato ed esperto interlocutore per la realizzazione di soluzioni orientate al business supportanti i processi aziendali.

L'azienda è presente sul territorio nazionale attraverso le proprie sedi di Vimercate (MI) e di Roma.

Le sue competenze comprendono la realizzazione di portali enterprise, siti dinamici orientati al *business to business* (B2B) e al *business to customer* (B2C), soluzioni per il supporto del *customer relationship management*, soluzioni per il *document & knowledge management.*

Ultimamente si stanno anche realizzando soluzioni avanzate per i processi aziendali tramite sistemi *mobile* e tramite l'uso di *personal digital assistant* (PDA).

Il progetto che dovevo realizzare durante il tirocinio era quello di una migrazione di un *Portale Web Aziendale* verso il nuovo standard tecnologico Java denominato dalla Sun con la sigla *JSR-168*.

Il portale esistente era realizzato tramite l'utilizzo del framework Struts e si proponeva per l'utilizzo intranet all'interno di piccole aziende.

Prima, dopo e durante questo processo di migrazione ho condotto uno studio sulle varie tecnologie Open Source e commerciali per la realizzazione di portali enterprise tramite tecnologia Java.

Ma cosa intendiamo per *portali enterprise* o *enterprise information portal (*EIP)?

Un portale è l'aggregazione di dati provenienti da fonti differenti, che può essere consultato su Internet.

Normalmente si forniscono servizi di integrazione delle applicazioni, servizi di collaborazione, ricerca, business intelligence e gestione dei contenuti.

Nell'ultimo periodo diverse software house hanno proposte delle soluzioni software per EIP più o meno complete.

Uno sviluppatore che debba scegliere su che architetture lavorare si troverà a dover decidere tra molti prodotti commerciali:

- Sun Java System Portal Server 6.2
- Hummingbird Enterprise 2004 Enterprise Webtop
- BEA WebLogic Portal 8.1
- IBM WebSphere Portal 5.0
- Oracle AS 10q Portal
- Sybase Enterprise Portal
- Microsoft Office Share Point Portal Server 2003

oppure tra diversi prodotti Open Source:

- JetSpeed – Apache J2EE Open Source Portal
- uPortal - J2EE Open Source Portal for Universities
- Cocoon 2 – Apache Web Communication Framework
- PHPNUKE – PHP based Portal/Community System

Nei mesi di tirocinio abbiamo avuto modo di provare alcuni di questi portali e di assistere e parlare con dei rappresentanti delle varie ditte.

Il pacchetto software a cui affidarci non era ancora ben definito all'inizio, se non per il fatto che avrebbe dovuto supportare pienamente la specifica JSR 168 Portlet, che come vedremo in seguito si propone come la più recente tecnologia per la realizzazione di EIP.

Questa trattazione è stata suddivisa in vari capitoli, di cui ora descriverò l'argomento. Nella prima parte analizzeremo principalmente cosa si intende con il concetto di Portale, vedendone gli elementi caratterizzanti e le funzioni che ci si aspetta che ogni produttore offra tramite il suo prodotto.

Successivamente descriverò brevemente le tecnologie messe a disposizione per lo sviluppo di applicazioni Web tramite il linguaggio Java al fine di rendere maggiormente chiari i capitoli successivi.

Nella quarta parte si spiegano le specifiche inerenti alla tecnologia JSR 168 Java Portlet mostrandone sia i concetti teorici che quelli tecnici. Nel penultimo capitolo invece mostrerò le architetture che ho avuto modo di studiare e provare che vengono attualmente ritenute leader per la creazione di sistemi EIP.

Infine elencherò brevemente quali sono stati i problemi principali incontrati durante la realizzazione e lo sviluppo pratico del nostro lavoro di migrazione, al fine di mostrare un esempio pratico di *use case* di un portale aziendale.

Questa trattazione non vuole entrare nel dettaglio dei formalismi tecnici riguardanti le tecnologie discusse ma mettere in evidenza i vantaggi ed i difetti intrinsechi allo sviluppo di un portale Web realizzato con tecnologie Java.

Condurrò quindi un discorso più teorico mostrando alcuni esempi pratici al fine di rendere chiari alcuni concetti fornendo la mia analisi riguardante gli argomenti discussi.

Il mio scopo è quello di fare conoscere al lettore tutte le possibilità offerte da queste tecnologie in maniera che possa avere una visione più chiara nel caso si abbia intenzione di portare avanti progetti simili a quello realizzato da me e dall'altro studente durante il mio periodo di tirocinio.

Capitolo 2
Struttura di un Portale

Il portale si prefigura come lo strumento principale per la gestione e l'integrazione informativa all'interno di un impresa.

L'obiettivo principale è rendere l'informazione accessibile e pertinente aiutando l'azienda ad incrementare i guadagni tramite un più rapido ed efficiente accesso ai dati.

Le varie proposte commerciali comprendono ambienti molto complessi, spieghiamo in questo capitolo quali sono gli elementi standard che costituiscono un portale.

Principalmente un buon Enterprise Information Portal (EIP) deve fornire due tipologie di servizi:

- L'integrazione tra dati, applicazioni e sorgenti diverse e non necessariamente localizzate uniformemente.
- La personalizzazione dello strato di presentazione dei contenuti e delle interfacce dell'ambiente lavorativo.

In rete possiamo trovare molti esempi di portali: quello forse più conosciuto è *Yahoo.com*, partito come motore di ricerca questo sito fornisce ora servizi diversi quali email, newsgroup, forum e altri servizi d'informazione.

L'utente si potrà trovare ogni volta che si autentifica un ambiente familiare e configurabile secondo le sue esigenze, avendo rapido accesso alle informazioni che più gli interessano.

In un qualsiasi portale all'utente si darà la possibilità di operare modifiche su due aspetti:

- Personalizzazione
- Configurazione

A prima vista il livello di configurazione e di personalizzazione possono apparire analoghi, ma operano su due contesti completamente differenti. Procediamo ora a descriverne le caratteristiche.

Personalizzazione

Il concetto base è quello di differenziare i contenuti e l'aspetto presentativo a seconda del ruolo che il visitatore ricopre all'interno dell'ambiente di lavoro.

All'inizio si propone una richiesta di autentificazione tramite la quale

l'utente accede al portale inserendo uno *username* ed una *password* all'interno di un'area preposta per il Login.

Dopo l'autentificazione all'utente verranno presentati contenuti mirati e limitati dal suo status all'interno della gerarchia del portale.

Ad esempio, all'interno di un'azienda all'amministratore verrà presentato uno spettro informativo maggiore di quello che verrà presentato all'impiegato medio. Questo serve a fornire solo le informazioni veramente utili e d'interesse all'utente scartando quello che non gli è necessario.

Configurazione

Ad ogni utente verrà permesso di manipolare la propria area e adattarla alle proprie esigenze determinando la struttura del proprio profilo sul portale. È importante notare che il termine inglese utilizzato non è *configuration* bensì *customisation* che rende maggiormente l'idea di una riconfigurazione personale dell'aspetto presentativo dei contenuti.

Viene infatti fornita la possibilità di:

- Gestire i layout/template del proprio profilo, quindi i colori, i caratteri e tutti gli aspetti prettamente estetici del portale.
- Aggiungere e rimuovere i servizi e le fonti d'informazione messi a disposizione dal portale a seconda del ruolo riconosciuto all'utente dopo il Login.

Da questo punto di vista si parla sempre di personalizzazione, decisa dall'utente e non dall'amministratore del portale.

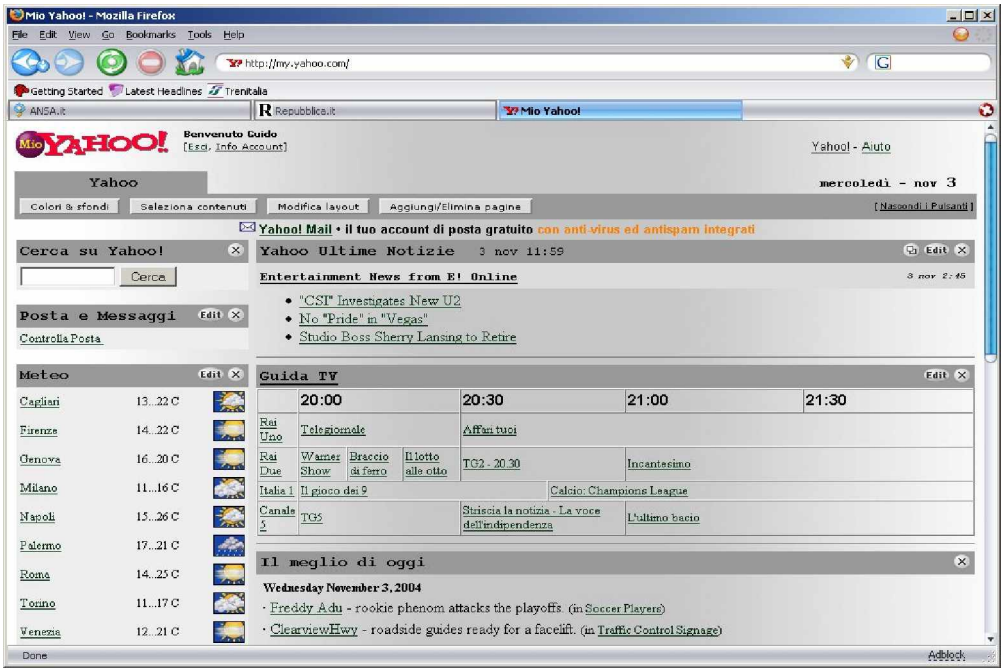

Figura 1: Esempio di portale sul Web.

Questi aspetti che abbiamo elencato sono relativi al rapporto tra il visitatore del portale e l'utilizzo che esso può farne.

È utile tenere conto che la vera capacità del portale è quella di proporsi come strumento conoscitivo, quindi come un oggetto utile per la ricerca e organizzazione dei contenuti.

Un portale concorrenziale deve essere in grado di recuperare informazioni tramite l'accesso a diverse sorgenti di dati.

La capacità di radunare i dati e proporli in maniera adeguata viene identificata sotto il nome di *aggregazione dei contenuti*.

Le sorgenti di dati possono essere suddivise in:

- Basi di dati e sistemi transazionali
- Documenti e contenuti non strutturati
- Servizi e fonti sul Web

È necessario quindi un sistema applicativo che riesca ad occuparsi del recupero intelligente di questi dati.

Con recupero intelligente intendiamo la capacità di riuscire a fornire contenuti essenziali e mirati recuperandoli da un enorme mole di dati.

Il software dovrà riuscire a capire le richieste dell'utente e a soddisfare le connessioni logiche presenti tra i vari documenti.

Si dovranno quindi fornire rapidamente ed efficientemente solo i contenuti che sono stati richiesti e gli eventuali collegamenti d'approfondimento al fine di velocizzare il processo informativo e conoscitivo dell'utilizzatore del nostro sistema.

La ricerca e l'organizzazione dei documenti e dei per scopi economici e aziendali tramite l'uso di tecnologia informatica non è sicuramente di recente interesse.

Due sistemi che vengono spesso proposti alle aziende sono quelli di *Data Warehouse* e *Enterprise Resource Planning*, *Customer Relationship Management*

Con *Data Warehouse (DWH)* si intende una base di dati utilizzata per mantenere tutte le informazioni sulle attività svolte nell'azienda al fine di poter servire come strumento di risorsa decisionale da parte della sezione di management.

Il software avrà quindi il compito di integrare dati provenienti dai vari sistemi transazionali e creare un "prospetto" dettagliato della situazione economica in tempo reale.

Da questo punto di vista il portale si configura come un'estensione dei Data Ware House in quanto estende la capacità di analizzare i collegamenti presenti tra vari processi aziendali.

Un altro strumento utile all'azienda è il *Customer Relationship Management (CRM)*.

Si può definire il *CRM* un insieme di strumenti, procedure organizzative, archivi, dati e modelli comportamentali creati per gestire le relazioni con il cliente. Il suo scopo è quello di migliorare il rapporto cliente-fornitore mantenendo traccia delle comunicazioni che vengono effettuate tra ogni membro dell'azienda e il *customer* al fine di poterle analizzare e studiare.

L'acronimo *ERP* si identifica con *Enterprise Resource Planning* ossia un sistema informatico dedicato alla programmazione dei piani di produzione e gestione dei materiali all'interno dei cicli di lavorazione di un impresa.

Si configura quindi come un software gestionale formato da vari moduli integrati tra loro, la cui caratteristica principale è la gestione integrata delle risorse partecipanti alla creazione dei prodotti e dei servizi di un'azienda.

Nella pratica ogni modulo sarà assegnato ad uno specifico reparto (personale, logistica, finanza, etc) e ci sarà un modulo centrale creato con il fine di prelevare le informazioni dalla base di dati e visualizzarle o passarle

ad una astrazione superiore.

Un portale aziendale ben strutturato dovrà permettere lo sviluppo e la creazione di sistemi analoghi se vorrà essere utilizzato all'interno di un azienda, fornendo tutti quegli strumenti attui a soddisfare la *Business Intelligence* ossia la ricerca intelligente di dati, la produzione ed analisi in tempo reale di informazioni per il supporto delle attività decisionali e di controllo dei *Knowledge Information Workers* (ossia i manager o comunque il personale decisore).

Principalmente il cuore del portale dovrà manipolare queste tipologie di dato:

- *Dati strutturati:* Riguardano informazioni ripetitive e presentate in formato strutturato come file, tabelle, database etc. Vengono manipolati al fine di estrarre, trasformare e caricare dati secondo le logiche di business aziendale e crearci un analisi dettagliata sopra.
- *Dati non strutturati:* Una componente ulteriore è formata dalla gestione di quei dati meno sensibili quali newsletter, appunti, grafici ed email presenti nei processi aziendali.

Le varie software house dovranno occuparsi di creare quindi dei motori di ricerca affidabili, intelligenti e veloci al fine di soddisfare le richieste in tempo reale di ogni componente dell'impresa.

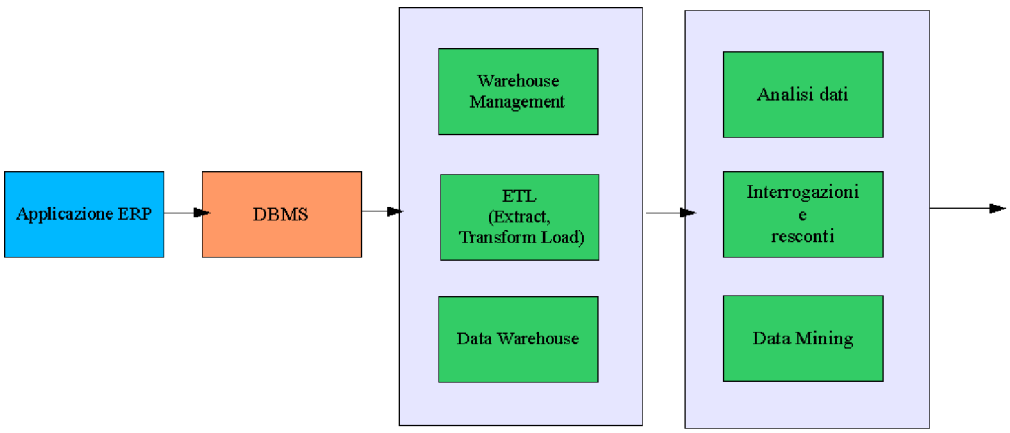

Figura 2: Gestione dei dati

Amministrazione del portale

Un buon ambiente EIP deve fornire strumenti di amministrazione che riducano il tempo di gestione di portali anche di grosse dimensioni. Un portale senza tool di amministrazione efficienti può non risultare d'interesse

per gli utilizzatori riducendone le potenzialità globali. Devono essere presenti strumenti *user friendly* per la gestione di utenti, gruppi di lavoro, autorizzazioni, impaginazione, sviluppo, caricamento dei Web service e molto altro ancora.

Architettura di un Portale

Al fine di garantire una massima scalabilità la Sun e l'IBM si sono assicurate che l'architettura di un portale sviluppato tramite tecnologia Java fosse più o meno standard, le caratteristiche base vengono infatti definite dalla *Java 2 Enterprise Edition (J2EE).*

Il portale prima di tutto viene visto come un applicazione Web, quindi sarà caratterizzato principalmente da due elementi costitutivi:

- Un *Application Server* che si occuperà di gestire nell'ambiente Java le nostre Servlet e Portlet tramite l'uso di opportuni Engine o Contenitori.
- Un *Web Server* che reinderizzerà le pagine generate e si occuperà della loro trasmissione tramite il protocollo HTTP.

In altri ambienti non prettamente Java troviamo più o meno la stessa struttura, basta pensare alle pagine PHP o ASP.

Queste nuove tecnologie mirano alla sostituzione del vecchio paradigma composto da pagine Web statiche e pagine dinamiche gestite da applicativi o script in Perl che venivano chiamati CGI (Common Gateway Interface). Queste soluzioni potevano andare bene per piccole applicazioni ma all'interno di ambienti fortemente dinamici si dimostrano inefficienti e scomode.

Altri applicativi tipici integrati in un portale sono:

- *Directory service:* Spesso legati ad Identity Server o altri sistemi per la gestione e autentificazione di utenti e processi.
- *Search Engine:* Al fine di permettere il recupero rapido d'informazione attraverso i numerosi dati di un portale.
- *Integrated Development Environment:* Vengono forniti strumenti di sviluppo mirati per la realizzazione di Web service, servlet, portlet etc.

Ovviamente questi sono i componenti interni al portale stesso, che verrà integrato con altri applicativi e sorgenti di dati esterni, si richiede quindi che un buon portale possa supportare l'interfacciamento con diversi applicativi e sorgenti informative.

Capitolo 3
Java Server Programming

In questo capitolo si presenterà brevemente un introduzione alla programmazione di rete tramite tecnologie Java. La programmazione moderna mira a separare i livelli di presentazione, business logic e le strutture dati al fine di rendere lo sviluppo più semplice e veloce. La Sun fornisce una piattaforma definita *Java 2 Enterprise Edition* che mira a soddisfare queste richieste di sviluppo definendo funzionalità standard che permettano di realizzare applicazioni distribuite utilizzanti tecnologie anche diverse tra loro.

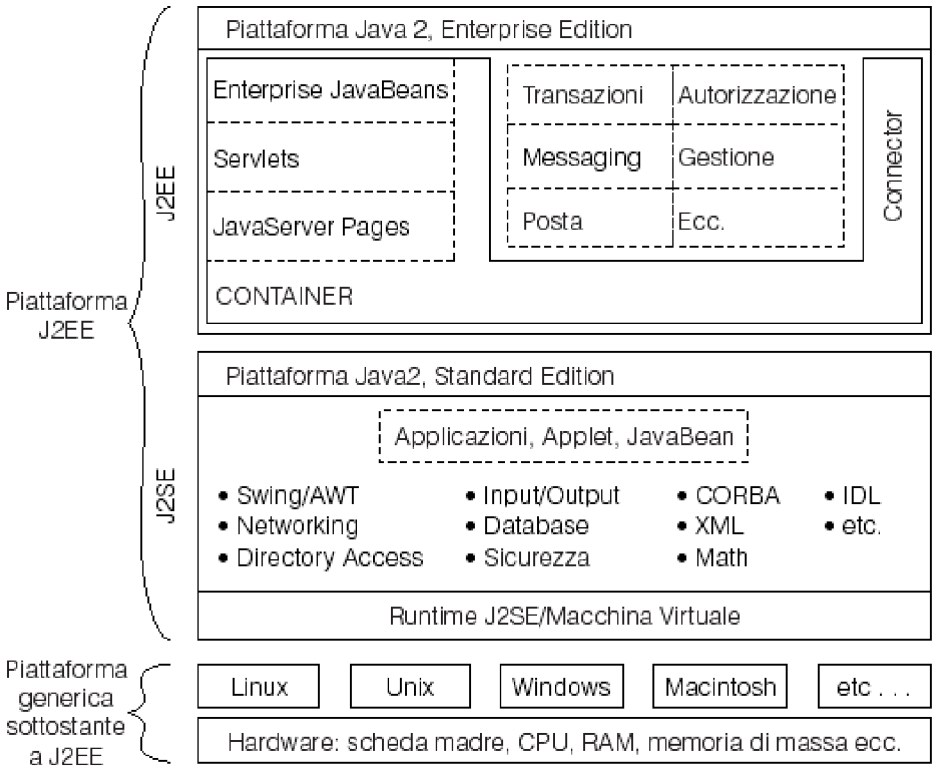

Figura 3: Il sistema di componenti J2EE [3]

La Sun ha voluto inserire nel nome della piattaforma il termine *Enterprise* per sottolineare il target di sviluppo mirato alle applicazioni distribuite.

Si presenta quindi la definizione di *architettura multi livello* caratterizzata

da un'applicazione composta da varie componenti logicamente distribuite su diversi livelli o su diversi strati dell'ambiente di elaborazione di rete.

Queste componenti possono essere distribuite su reti locali (LAN), su reti geografiche (WAN) e anche su Internet.

Si comprende quindi la difficoltà nel gestire un così vasto insieme di tecnologie, mantenendo comunque ordine ed una struttura pulita dal punto di vista architetturale.

I requisiti che questa piattaforma mira ad assicurare sono:

- *Stabilità*: Si attribuisce questa proprietà ad un applicazione che svolge il suo ruolo senza bloccarsi, senza subire crash e senza causare perdite di tempo all'utente a causa di mancanze e comportamenti non previsti. Se si pensa che le applicazioni Web come gli EPI devono svolgere transazioni critiche come possono essere quelle finanziarie o di *e-business* che hanno un impatto diretto sulle attività umane si può ben comprendere quanto questo requisito sia forse il più importante. La teoria dell'informazione ci spiega purtroppo che è impossibile scrivere software senza bachi di alcun tipo, J2EE vuole fornire una piattaforma in cui questo difetto intrinseco sia ridotto notevolmente.
- *Scalabilità:* Una buona applicazione deve poter essere facilmente ampliata soddisfacendo le esigenze di espansione degli utenti. Un' applicazione scalabile deve facilmente adattarsi all'incremento del carico lavorativo, rendendosi disponibile anche in situazioni di stress. Questa proprietà deve essere soddisfatta soprattuto a livello di piattaforma, se già questo livello è difettoso lo saranno anche quelli superiori, creando colli di bottiglia sempre più gravosi.
- *Sicurezza:* Questo aspetto è sempre più importante nell'ambito dell'Information Tecnology, come la stabilità può influire criticamente sulla vita degli utenti. Con sicurezza si intende infatti il grado di protezione da parte di un utilizzo non autorizzato e malevolo dell'applicazione. Il problema della sicurezza è l'elevato costo che richiede per essere mantenuta, ed il costante impiego di una mano d'opera specializzata. J2EE si propone come una piattaforma piuttosto sicura grazie al fatto di sfruttare le potenzialità intrinseche contenute in un linguaggio come Java (linguaggio fortemente tipato, nessun accesso a puntatori alla memoria, garbage collector, security manager, verificatore del codice byte code etc..)
- *Semplicità:* Gli utenti di un applicazione devono rispettare scadenze, un applicazione poco amichevole poco *user friendly* causa perdita di tempo e quindi perdita di denaro. Una buona applicazione deve essere semplice

sia da utilizzare sia dal punto di vista strutturale. La tecnologia Java aiuta questo processo tramite la classica tecnica del *divide et impera* caratteristica della programmazione Object Oriented.

Architettura di una applicazione Web

Una tipica applicazione Web può essere rappresentata da tre elementi di astrazione:

- Livello di Presentazione
- Livello di Applicazione
- Livello di trasmissione e memorizzazione dei dati

Il primo livello viene manipolato sia dal Web Browser ma anche dal Web Server che è responsabile di assemblare i dati tramite un aspetto presentabile.

Il secondo livello di cui abbiamo già discusso è il cuore pulsante del sistema, tutto quell'insieme di codice che gestisce e manipola i dati ossia il terzo livello.

Il terzo livello coinvolge quindi la *collezione dei dati* ossia la trasmissione di dati tra utente e server, normalmente questa viene gestita ad esempio da un form HTML in cui l'utente inserisce i propri dati e li riceve attraverso la generazione dinamica di una pagina di risposta.

Lo scambio di dati viene gestito tramite il protocollo HTTP, questo sistema fornisce tre metodi tipici per realizzare questa comunicazione:

- Il metodo HEAD che semplicemente recupera l'informazione riguardante un documento e non il documento stesso.
- I metodi GET e POST che si occupano in diversa maniera della gestione della richiesta di esecuzione di un applicazione web. Sebbene questi metodi possano essere utilizzati per realizzare gli stessi compiti il primo dovrebbe essere sfruttato per eseguire richieste di recupero d'informazione mentre il secondo per avvisare di un eventuale modifica o inserimento dei dati.

Tecnologie Server Side

Le tecnologie fornite da J2EE per lo sviluppo di applicazioni lato server sono principalmente due:

- Java Servlet
- Java Server Pages

La *Java Servlet API* fornisce degli oggetti che vengono spesso equiparati ad un altra tecnologia Java ossia le *Applet*. Queste due componenti sono infatti molto simili, l'Applet risulta una piccola applicazione o libreria scritta in Java che viene scaricata dal Web Browser sul sistema ed eseguita al fine di realizzare dinamicamente qualche operazione sui contenuti. La *Servlet* mantiene similmente la stessa costituzione ma viene eseguita dal lato Server.

La Servlet è quindi un programma server side che gestisce richieste HTTP e ritorna come risultato risposte HTTP.

Le differenze con le applet sono principalmente inerenti all'assenza di un aspetto visuale ma vengono eseguite entrambe all'interno della *Java Virtual Machine (JVM).*

Queste applicazioni non si devono occupare direttamente di questioni di basso livello come la connessione alla rete, la cattura della richiesta e la formattazione delle risposte.

Questi dettagli vengono gestiti dal *Servlet Container* o *Servlet Engine*, che traduce le richieste HTTP in oggetti manipolabili dalle servlet.

Il Servlet Container gestirà anche il ciclo di vita delle servlet determinando quindi la creazione, la distruzione e il caricamento di queste componenti.

Un aspetto importante è che la Servlet viene caricata su richiesta una prima volta dopodichè rimane in memoria per servire successive chiamate diminuendo i costi di overhead.

Java Server Pages (JSP)

Questa tecnologia web combina HTML, scripting e componenti server side in un unico file chiamato Java Server Page (JSP).

Quando un utente richiede la visione di un file JSP il server prima di tutto genera una corrispettiva Servlet e ritorna un frammento di codice HTML al Web Browser.

In questa maniera è possibile realizzare pagine Web dinamiche integrando l'HTML al codice Java, riportiamo ad esempio una pagina JSP tratta dal libro *Professional Java Server Programming [1]*:

```
<HTML>
<HEAD>
<TITLE>
Demo of a JSP page</TITLE>
</HEAD>
<BODY>

<!-- Set global information for the page -->
```

```
<%@ page language="java" %>

<!-- Declare the character variable -->
<%! char c = 0; %>

<!-- Scriptlet - Java code -->
<%
     for (int i = 0; i < 26; i++)
     {
          for (int j = 0; j < 26; j++)
          {
          // Output capital letters of the alphabet,
          // and change starting letter
          c = (char) (0x41 + (26 - i + j)%26;
%>

<!-- Output the value of c.toString() to the HTML page
-->
<%= c %>
<%
     }
%>
<BR>
<%
     }
%>
</BODY>
</HTML>
```

Possiamo vedere come la pagina sia composta sia da codice HTML sia da codice Java: un esempio di tecnologie simili viene fornito dalle pagine *Microsoft ASP* e da quelle *PHP*.

Integrazione di sorgenti d'informazione

La tecnologia Java fornisce una serie di API standard per accedere alle varie sorgenti d'informazione esistenti in un ambiente tecnologico complesso.

Riportiamo un elenco di queste API:

- *JDBC:* Questo acronimo sta per *Java Data Base Connectivity* ed è l'API fornita per la connessione con le basi di dati relazionali. Viene fornita un'interfaccia comprensiva di funzioni indipendenti dalla base di dati utilizzata, attraverso l'utilizzo di un driver specifico che faccia da traduttore tra JDBC e il DBMS.

- *JTA*: Questa libreria (*Java Transaction API)* si occupa gestire il coordinamento delle transazioni attraverso sorgenti d'informazione transazionali eterogenee.
- *JMS:* La *Java Messaging Service* si occupa di fornire servizi di messaggistica nel software, più precisamente si tratta d'interfacce che permettono di accedere ed utilizzare i servizi di un sistema di *middleware* orientato ai messaggi.
- *JNDI:* La *Java Naming and Directory Interface* fornisce una struttura in grado di gestire la comunicazione tra moduli software la cui funzione principale è quella di associare nomi ad oggetti, similmente al meccanismo utilizzato per riferire i files nei File System. Verranno quindi recuperate informazione tramite driver colloquianti con LDAP, CORBA service ed altri servizi.
- *Java Mail*: È l'API standard per l'invio di messaggi email, e fornisce un interfaccia semplice che permette allo sviluppatore di non occuparsi dei dettagli di basso livello come la comunicazione tra socket.

Enterprise Java Beans (EJB)

Una recente tecnologia che ha riscosso molto successo è stata quella degli *Enterprise Java Beans,* queste specifiche definiscono un architettura per la realizzazione e l'esecuzione di componenti software distribuiti, riutilizzabili e portabili.

Questi componenti denominati *enterprise bean* vivono all'interno di un *EJB container* e attraverso di esso offrono servizi remoti al client, proponendosi quindi come *Web service*.

Durante il periodo di tirocinio non ho utilizzato questa tecnologia ma ho ritenuto importante inserirne un breve accenno in questa breve discussione sulla programmazione Web in Java perché ormai di grande interesse in questo campo.

Struts Framework

Il progetto *Apache Jakarta* nel duemila ha iniziato un progetto Opensource di successo denominato *Jakarta Struts*.

Questo progetto compatibile con la piattaforma J2EE propone un framework basato sul modello MVC (model, view, control) che aiuti a sviluppare applicazioni Web con facilità seguendo questa logica implementativa.

Gli sviluppatori preferiscono l'utilizzo di Struts per i seguenti motivi:

- Presenza di un disegno architetturale valido
- Riduzione dei tempi di progetto
- Semplificazione dello sviluppo

Uno dei principali problemi che si affrontano nello sviluppo di un applicazione Web è il dissociare i vari componenti applicativi in base al loro ruolo al fine di ottenerne vantaggio in riusabilità e manutenzione. *Struts* vuole sostituirsi al modello che in letteratura è conosciuto come “JSP Model One” ossia una logica implementativa in cui il livello di presentazione, controllo e business logic sono tutti devoluti alla pagina JSP.

Questo modello è inadatto a progetti di grosse dimensioni, e viene sconsigliato dalla stessa Sun.

Nel modello *MVC (o Model 2)* si affidano i ruoli di presentazione, controllo e business logic a componenti diversi e disaccoppiati. In questo paradigma si hanno tre livelli logici:

- *Controller:* determina il modo in cui l'applicazione risponderà agli input dell'utente.
- *Model:* serve a memorizzare i dati ed i contenuti.
- *View:* si occupa solo della visualizzazione dei dati.

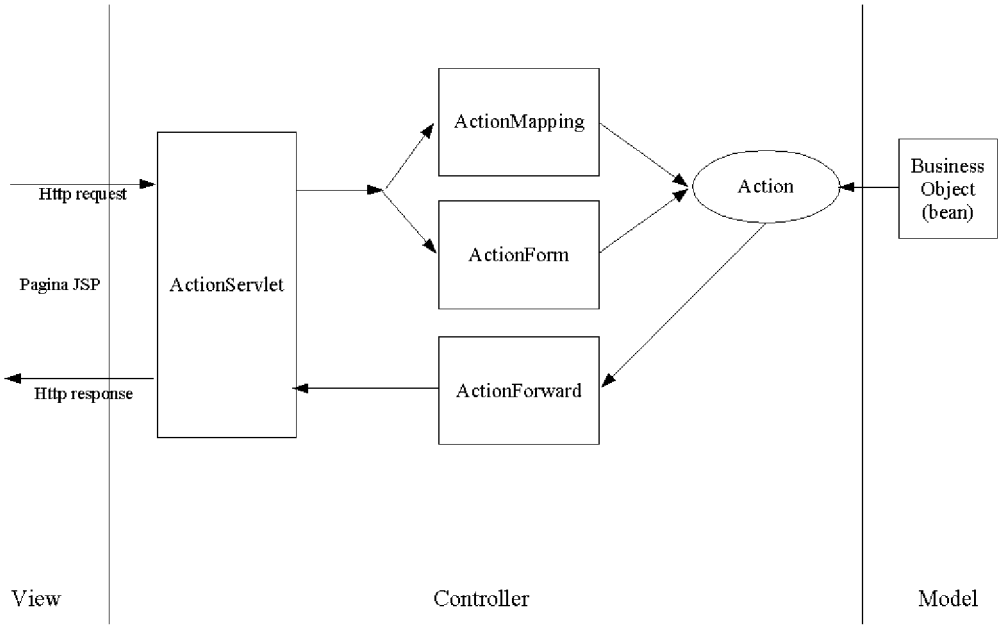

Figura 4: Implementazione MVC in Struts

Nell'implementazione dell'MVC di Struts entrano in gioco le seguenti classi ed interfacce:

- *struts-config.xml*: è il file di configurazione principale dell'applicazione in cui vengono gestiti gli elementi e le loro associazioni.
- *ActionServlet:* svolge il ruolo di controller gestendo tutte le richieste dell'applicazione.
- *Action*: Sono le classi alle quali la ActionServlet delega l'elaborazione della richiesta.
- *ActionMapping:* memorizzano i rapporti tra gli oggetti associati ad una Action nello struts-config.xml.
- *ActionForm:* servono a memorizzare i dati provenienti dalle richieste HTTP.
- *ActionForward:* contiene l'informazione del path della vista da fornire all'utente.
- *Tag Library:* vengono fornite librerie di tag al fine di semplificare compiti comuni nello sviluppo di pagine JSP.

I due elementi più importanti sono quindi *struts-config.xml* contenente tutta la configurazione riguardante l'applicazione e la *ActionServlet* che gestisce tutte le richieste del client smistando il flusso elaborativo in maniera

opportuna. In questa maniera si ha un unico punto di gestione del flusso applicativo riuscendo a sviluppare in maniera univoca le opzioni riguardanti sicurezza, logging e altro ancora.

Ultimamente alcuni portali come Websphere dell'IBM forniscono dei plugin per integrare il modello MVC di Struts con la nuova specifica JSR168 riuscendo ad ottenere un vantaggio da entrambe le tecnologie.

Capitolo 4
Portlet – Specifica JSR 168

Cos'è una Portlet? Dalle specifiche:

"La portlet è un componente web basato sulla tecnologia Java, gestito da un portlet container che processa richieste da parte dell'utente e genera contenuti dinamici. Le portlet sono usate dai portali come componenti d'interfaccia utente pluggabili al fine di provvedere un livello di presentazione per i sistemi informativi."

Le portlet generano dei frammenti di codice markup (es. HTML, XHTML, WML) che integrati ed aggregati secondo determinate regole formano il documento completo. Un portale sarà composto quindi da più portlet aggregate al fine di formare una pagina completa.

Il rapporto tra utente e portlet attraverso i web client è implementato tramite il classico paradigma di scambio di richieste e risposte tipico dei portali.

Le portlet possono generare contenuti diversi a seconda dell'utente del portale che le utilizza, e a loro volta gli utenti possono personalizzarsi il portale grazie all'alta modularità di questa tecnologia Java.

Le portlet analogamente alle *servlet Java* delle quali sono un estensione vengono gestite da un *Portlet Container*.

Questo contenitore fornisce l'ambiente runtime richiesto per fare girare queste componenti e ne gestisce l'intero ciclo di vita, oltre a fornire uno spazio persistente per memorizzare le preferenze relative ad esse.

Il contenitore si occupa si ricevere le richieste dal portale e di reinderizzarle alle portlet opportune.

Il contenitore non si occupa però dell'aggregazione dei dati, questa opzione è di responsabilità del portale.

Esempio:

- Un client dopo essersi autentificato effettua una richiesta HTTP al portale.
- La richiesta viene ricevuta dal portale.
- Il portale determina se la richiesta contiene un azione mirata per ciascuna delle portlet associate alla pagina del portale.
- Se esiste una portlet per quell'azione richiesta, il portale inoltra la domanda al *portlet container* che invocherà una chiamata ad essa.

- Il portale invoca le portlet, tramite il contenitore, al fine di ottenere dei frammenti di markup che potranno essere inclusi nella pagina risultante.
- Il portale aggrega i vari markup prodotti dalle portlet e spedisce al browser la pagina di risposta.

Per gli sviluppatori Java che già sviluppavano tramite l'uso delle tecnologie Servlet o delle Java Server Pages il salto verso le portlet sarà breve.

Andiamo ad esaminare somiglianze e differenze con la *Servlet Specification.*

Similitudini:

- Le portlet sono componenti web basate su tecnologia Java
- Le portlet sono gestiste da un contenitore specializzato
- Le portlet generano contenuti dinamici
- Il ciclo di vita delle portlet è gestito dal contenitore
- Le portlet interagiscono con il client web tramite un paradigma di richieste e risposte

Differenze:

- Le portlet generano soltanto frammenti di markup, non documenti completi. Il portale si occupa di gestire l'aggregazione dei vari pezzi di markup
- Le portlet non sono limitate ad un URL
- I client web interagiscono con le portlet attraverso il portale e non direttamente
- Le portlet hanno un sistema di gestione delle richieste più raffinato comprendente Action Requests e Render Requests.
- Le portlet hanno delle modalità predefinite e degli stati delle finestre che indicano la funzione la portlet sta eseguendo (view, edit, help mode).
- Più portlet coesistono nella stessa pagina di un portale.
- Le portlet possiedono mezzi preconfezionati per la memorizzazione permanente relativa alla loro configurazione e personalizzazione dei dati
- Le portlet hanno accesso ai profili utenti.
- Le portlet possiedono funzioni atte ad avere una gestione degli URL portabile indipendente dal portale.
- Le portlet non possono specificare l'insieme di codifica dei caratteri della risposta.

- Le portlet non possono specificare le intestazioni HTTP delle risposte.

Al fine di mantenere il massimo di livello di compatibilità con le servlet ed il massimo riutilizzo, si è cercato di mantenere invariate il maggior numero di metodologie.

Inoltre una portlet può facilmente gestire rapporti con servlet e Java Server Pages ed effettuare chiamate ad esse. è fornito anche il supporto per l'inoltro di richieste a Servlet o Jsp da parte delle portlet.

Per capire meglio il concetto di portlet bisogna mostrare il risultato che forniscono a livello d'interfaccia utente. Ogni portlet possiede un titolo, dei bottoni di controllo, ed altre decorazioni che la fanno assomigliare ad una classica interfaccia a finestra, come osservabile dalle Figure 1 e 2.

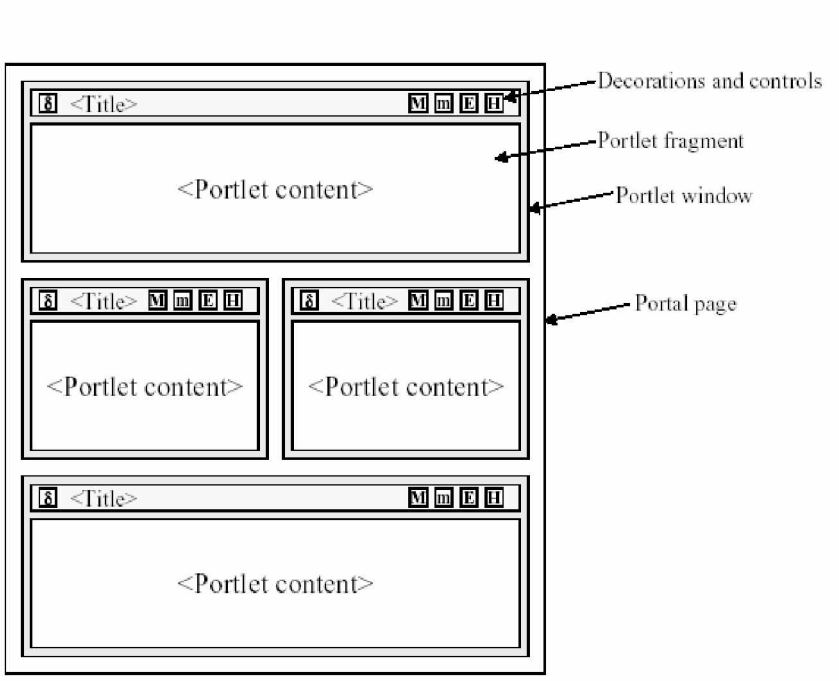

Figura 5: Elementi di una pagina di un portale

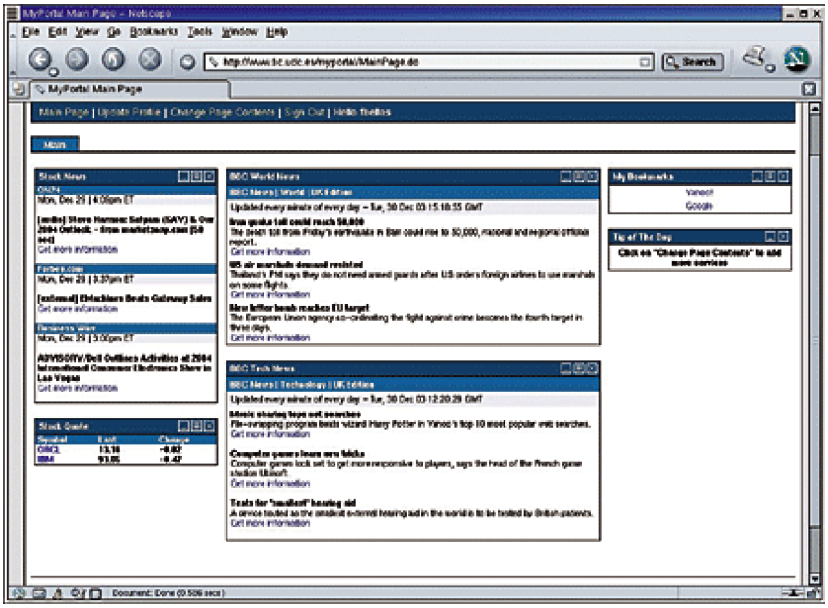

Figura 6: Esempio di pagina di un portale utilizzante le portlet

Portlet e GenericPortlet

L'oggetto principale che caratterizza la nuova specifica è l'interfaccia `Portlet`. Tutte le portlet implementano quest'interfaccia direttamente o estendendo una classe che la implementa.

La classe `GenericPortlet` è l'oggetto base da estendere per creare le proprie portlet, e viene fornito con delle funzionalità di default.

Il ciclo di vita della portlet è gestito principalmente da quattro metodi che vengono chiamati direttamente dal container, questi metodi sono:

- `init()`
- `destroy()`
- `processAction()`
- `render()`

Vediamone l'utilizzo:

La `init()` viene chiamata quando la portlet viene istanziata dal contenitore al fine di contenere la logica che preparerà l'oggetto a gestire le richieste.

La `destroy()` viene chiamata quando il contenitore distrugge una portlet al fine di far pulizia quando l'oggetto non viene più utilizzato o quando il server viene spento.

La `processAction()` viene chiamata dopo che l'utente ha effettuato una richiesta, serve a processare dei dati in input dall'azione dell'utente.

La `render()` viene chiamata ogni volta che c'è da ridisegnare o renderizzare un output dei dati.

La `GenericPortlet` possiede in aggiunta a questi metodi chiamati dal contenitore delle implementazioni specifiche del metodo `render()` che lo specializzano ulteriormente a seconda della modalità di utilizzo della portlet:

- `doView()`
- `doEdit()`
- `doHelp()`

In particolare:

La `doView()` è usata per renderizzare la portlet quando si trova in *View Mode*, ossia quella modalità d'utilizzo della portlet in cui l'utente interagisce con essa.

La `doEdit()` è usata per renderizzare la portlet quando si trova in *Edit Mode*, ossia quella modalità in cui è possibile specificare le opzioni di personalizzazione e configurazione della portlet.

La `doView()` è usata per renderizzare la portlet quando si trova in *Help Mode*, ossia per mostrare la pagina relativa alla guida d'utilizzo della portlet.

L'unico di questi metodi che è obbligatorio è il metodo `doView()` gli altri possono essere utilizzati a seconda delle preferenze dello sviluppatore.

Portlet Mode e Stato Finestra

Ogni contenitore deve gestire per ogni portlet il *portlet mode* ed il *window state*. Come abbiamo già accennato, esiste una modalità specifica a seconda della funzione eseguita in quel momento dalla portlet. I modi sono *View, Edit, Help.* Questi metodi vengono utilizzati dal metodo di default `render()` che decide quale metodo di visualizzazione di basso livello chiamare.

Il *window state* indica invece l'ammontare di spazio all'interno della pagina del portale che può essere assegnato per una portlet. Gli stati possibili sono *minimized, maximized* o *normal*. La portlet può decidere di utilizzare quest'informazione per decidere quante informazioni renderizzare.

Portlet Context

Al fine di poter condividere e comunicare dati ed informazioni per ciascuna portlet esiste una istanza della interfaccia *PortletContext* associata a ciascuna portlet application deployata nel container. Tramite questo oggetto è possibile accedere ai parametri di inizializzazione della portlet application, immagazzinare dati, recuperare risorse e ottenere un *requestDispatcher* per poter includere JSP o Servlet.

Preferenze della portlet

Ogni portlet prevede di poter supportare differenti viste e contesti per utenti differenti. Questo supporto ci viene fornito tramite l'interfaccia `PortletPreferences` che chiamata dal contenitore della portlet è responsabile del recupero e della memorizzazione di queste “preferenze” memorizzate in opportune coppie di nomi e valori.

I metodi forniti per il recupero e la memorizzazione delle preferenze saranno quindi dei classici `getValues()` e `setValues()`.

Ovviamente verranno eseguiti dei controlli di validazione per determinare al momento della memorizzazione tramite una funzione di `store()` se le preferenze rispettano i vincoli esistenti tramite l'uso di una classe opportuna definita `PreferencesValidator`.

Configurazione della Portlet

Attraverso il *PortletConfig* la portlet accede al suo deployment descriptor che contiene tutti i dati che le sono necessari durante il funzionamento, anche parti del resource bundle se necessario (viene usato dal metodo render della GenericPortlet).

I parametri iniziali inseriti nel descrittore della portlet possono essere richiamati tramite l'utilizzo dei metodi *getInitParameterName* e *getInitParameter*.

Portlet URL

All'interno del contenuto di una pagina si presenta la necessità di creare dei collegamenti, quindi degli *URL*, questi a volte possono riferirsi alla portlet come ad esempio quando si vuole attivare un opzione o inviare il testo di un form. Verranno quindi inviate delle richieste dal portale alla portlet, questa tipologia di URL viene definita *PortletURL*.

La specifica JSR168 fornisce delle librerie opportune chiamate *Portlet API*. Per inserire collegamenti alle portlet bisogna quindi creare degli oggetti *PortletURL* che verranno invocati tramite l'utilizzo dei metodi *createActionURL* e di *createRenderURL* dell'interfaccia *RenderResponse*.

La differenza tra i due metodi è il tipo di URL creato che può essere:

- Action URL
- Render URL

La differenza tra i due è che il secondo fornisce una versione più specializzata che viene utilizzata per assicurarsi che tutti i parametri vengano renderizzati nella seguente richiesta di renderizzazione della portlet.

Viene fornita la possibilità indispensabile della specifica di parametri all'oggetto *PortletURL* tramite l'utilizzo dei metodi *addParameter* e *setParameter* stando attenti a non avere nomi ridondati.

Forniamo qui in seguito l'esempio di un Portlet URL:

```
...
PortletURL url = response.createRenderURL();
url.setParameter(“utente”,”unipmn.it”);
url.setParameter(“show”,”summary”);
writer.print(“<A
HREF=\””+url.toString()+”\”>Sommario</A>”);
...
```

Un altra possibilità è quella di specificare la modalità portlet e lo stato della finestra della portlet che si andrà a richiamare ad esempio:

```
...
PortletURL url = response.createActionURL();
url.setParameter(“paymentMethod”,
“creditCardInProfile”);
url.setWindowState(WindowState.MAXIMIZED);
writer.print(“<FORM METHOD=\”POST\” ACTION=\””+
url.toString()+”\”>”);
...
```

Bisogna ricordare che è possibile permette la creazione di collegamenti sicuri tramite il protocollo *HTTPS* richiamato tramite il metodo *setSecure* di *PortletURL*.

Al fine di assicurare la massima portabilità è consigliato utilizzare questi metodi e non i metodi HTTP GET/POST che a seconda del portale potrebbero avere un'implementazione diversa.

Gestione delle richieste nella Portlet

L'interfaccia Portlet fornisce due metodi principali per la gestione delle richieste, il metodo `processAction` ed il metodo `render`.

Quando il contenitore invoca il metodo `processAction` di una portlet, si

dice che si è inoltrata una richiesta d'azione(*action request*).

Quando il contenitore invoca il metodo `render` si dice che si è inoltrata una richiesta di renderizzazione(*render request*).

Normalmente gli utenti comunicano attraverso degli URL appositi creati dalla portlet, questi vengono chiamati *portlet URL* e si dividono in *action URL* e *render URL* e vengono tradotti se selezionati negli equivalenti messaggi di *action request* e di *render request*.

Quando viene attivato un *action URL* il contenitore invoca la richiesta tramite la `processAction` che provvederà a gestire l'azione, in questo periodo il contenitore dovrà attendere la fine di questa procedura.

Il risultato dell'azione verrà trasmesso al portale tramite l'invocazione del metodo `render` a meno che non ci sia la possibilità di riutilizzare dei contenuti memorizzati in cache.

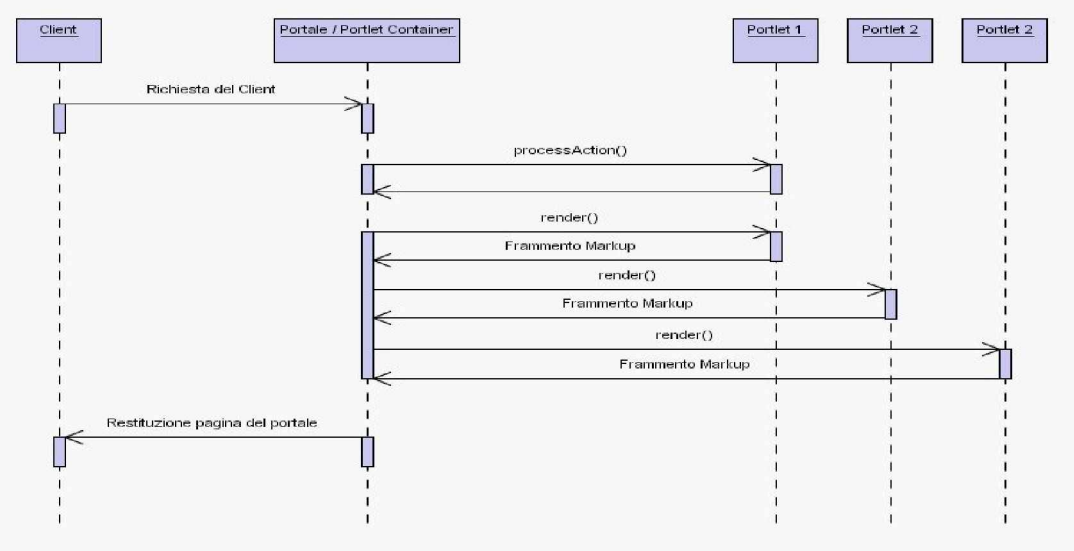

Figura 7: Sequenza di gestione delle richieste

Informazioni sull'utente

Un opzione sicuramente utile inserita in questa specifica è una gestione più strutturata rispetto alle *servlet* nel recupero delle informazioni relative all'utente che sta utilizzando le nostre pagine dinamiche.

Tramite questo sistema è possibile recuperare dati(memorizzati ad esempio durante la registrazione dell'utente al sito) quali nome, età, indirizzo, numero di telefono dell'utente autentificato. In questa maniera è possibile

gestire un accesso maggiormente personalizzato.

Questi attributi verranno recuperati ad esempio tramite il richiamo della costante `PortletRequest.USER_INFO` ad esempio:

```
...
Map userInfo = (Map)
request.getAttribute(PortletRequest.USER_INFO),
String nome = (userInfo != null) ? (String)
userInfo.get("nome.utente") : "";
String cognome = (userInfo != null) ? (String)
userInfo.get("conome.utente") : "";
...
```

La definizione di queste proprietà verrà definita all'interno di un *deployment descriptor* scritto in XML(un file *portlet.xml* all'interno di un archivio WAR) contenente dei nomi logici mappati sugli attributi utente forniti dall'ambiente d'esecuzione.

Portlet e Caching

Al fine di minimizzare il tempo di risposta del Portale verso gli utenti e al fine di ridurre il carico dello stesso, la nuova specifica permette il *caching* dei contenuti.

La specifica definisce un meccanismo di caching basato sull'espirazione delle pagine. È limitato al solo rapporto tra utente e portlet, questo significa che la memoria tampone non è condivisa tra gli utilizzatori della stessa portlet.

Tramite l'inserimento delle seguenti righe all'interno del file XML definente la portlet:

```
...
<portlet>
  ...
  <expiration-cache>240</expiration-cache>
  ...
</portlet>
...
```

si può specificare in secondi dopo quanto tempo la memoria verrà ripulita dalla portlet, nel nostro caso dopo quattro minuti.

Quando il contenuto di una portlet è nella cache e non è ancora scaduto, il *portlet container* potrà recuperarlo dalla cache evitando di chiamare la portlet e quindi aumentano la velocità e diminuendo il carico.

Nel caso il caching non sia specificato all'interno del descrittore o tramite funzioni, si considerà disabilitato

Packaging and Deployment

Con deploying si intende il caricamento della portlet sul portale, al fine di poter essere richiamata ed inserita all'interno di una pagina.

Analogamente ai sistemi utilizzati per servlet e JSP la *Portlet Specification* segue in parte lo standard *Web Application Archive(WAR)* ossia un semplice file compresso contenente le classi, le librerie ed i file di configurazione(scritti in XML di norma) utilizzati dalla nostra applicazione Web.

La `struttura` di un WAR Portlet non varia molto, se non per un file descrittore della portlet definito come `portlet.xml` contenente le varie informazioni e configurazioni, i vari attributi, i parametri iniziali, le preferenze legate all'utente, il titolo della portlet e cosi via'.

Esempio di `portlet.xml`:

```
<portlet-app>
  <portlet>
    <portlet-name>NewsPortlet</portlet-name>
    <portlet-class>
      sample.portlet.NewsPortlet
    </portlet-class>
   <init-param>
     <name>new.url</name>
     <value>java:/comp/env/NewsProvider</value>
   </init-param>
   <expiration-cache>7200</expiration-cache>
   <supports>
     <mime-type>text/html</mime-type>
     <portlet-mode>EDIT</portlet-mode>
     <portlet-mode>HELP</portlet-mode>
   </supports>
   <portlet-info>
     <title>News Aggregation Portlet</title>
     <short-title>News Portlet</short-title>
     <keywords>news,aggregator,rdf</keywords>
   </portlet-info>
   <portlet-preferences>
     <preference>
       <name>wired</name>
       <value>"http://www.wired.com/news/rss"</value>
       </preference>
       <preference>
         <name>slashdot</name>
         <value>"http://slashdot.org/index.rss"</value>
```

```
        </preference>
       <preferences-validator>
        sample.portlet.NewsPreferencesValidator
       </preferences-validator>
     </portlet-preferences>
   </portlet>
</portlet-app>
```

Come possiamo osservare `portlet-name`, `portlet-class` ed `init-param` definiscono il nome la classe di partenza ed i parametri iniziali per la nostra portlet .

Segue poi il tag `expiration-cache` quanto deve durare la permanenza di un contenuto creato dalla nostra portlet all'interno della cache, se non specificato il tempo durerà quanto la richiesta di render.

Il tag `supports` definisce le tipologie di contenuto supportate e le modalità gestite dalla portlet(ricordando che la VIEW è obbligatoria e può essere omessa).

All'interno del tag `portlet-info` possiamo inserire varie informazioni quali titolo, titolo abbreviato e parole chiave legate alla portlet.

L'ultimo campo di tag è legato alle preferenze di default, in cui posso inserire le coppie nome e valore contenute fin dal primo caricamento della portlet, all'interno di `portlet-preferences`.

Andando nel particolare, un tipico esempio di Web Archive per una portlet conterrà questi elementi:

- Il classico descrittore dell'archivio: `/WEB-INF/web.xml`
- Il file descrittore della portlet: `/WEB-INF/portlet.xml`
- Le classi della portlet: `/WEB-INF/classes`
- Le eventuali Java Server Pages in `/WEB-INF/jsp/*.jsp`
- Le varie librerie `/WEB-INF/lib/*.jar`
- Un eventuale manifesto /`META-INF/MANIFEST.MF`

Ovviamente questo schema può variare e contenere strutture differenti, in ogni caso i primi tre punti devono essere assolutamente rispettati, anche se il primo campo(`web.xml`) perde relativamente valore rispetto alle servlet e va' a contenere semplicemente il nome dell'applicazione:

```
<web-app>
<display-name>NewsPortlet Sample</display-name>
</web-app>
```

Questo per mantenere la compatibilità con il precedente standard, dato che il sistema di archiviazione delle portlet ne è semplicemente un estensione.
Una critica che può essere fatta è che attualmente il sistema di deploying delle portlet viene implementato non sempre in maniera efficiente e comoda dai diversi portali che, pur fornendo sistemi grafici per effettuare questo lavoro, spesso rendono il processo alquanto ardimentoso.

Portlet Tag Library

All'interno della specifica è stata inserita un opportuna *tag library* per permettere di inserire all'interno delle pagine *JSP*, incluse all'interno di una portlet, delle direttive create con lo scopo di permettere un accesso diretto a quest'ultima ed ai suoi elementi come ad esempio le *RenderRequest* e le *RenderResponse*. Questa libreria permette anche la creazione di *Portlet URL* all'interno delle JSP.

L'utilizzo di questi tag è molto semplice si inserisce all'interno della pagina JSP un header dichiarante l'utilizzo della library:

```
<%@ taglib uri=”http://java.sun.com/portlet” prefix=”portlet” %>
```

che ne specifica l'utilizzo.

Ovviamente è necessario che il contenitore preveda una implementazione di questa libreria, inoltre è possibile la definizione di tag aggiuntivi seguendo determinate regole imposte dallo standard JSP al fine di permettere la semplificazione di alcuni processi implementativi

Esistono diversi tipi di tag, ad esempio vengono forniti i *defineObjectTag* che permettono la definizione delle seguenti variabili che hanno lo stesso ruolo delle funzioni delle API:

- *renderRequest* richiama l'oggetto RenderRequest
- *renderResponse* richiama l'oggetto RenderResponse
- *portletConfig* richiama l'oggetto PortletConfig

Per utilizzare queste chiamate si inserirà quindi la seguente dicitura:

```
<portlet:defineObjects/>
```

in seguito si potranno inserire gli elementi dichiarati precedentemente.

Ad esempio volendo definire il titolo della risposta, si potrà scrivere:

```
<%=renderResponse.setTitle(“titolo della portlet”)%>
```

invocando quindi il metodo *setTitle()* dell'oggetto *RenderResponse* che andrà a cambiare il titolo della portlet.

Un altro tag utile è *actionUrl* che permette di richiamare dei Portlet URL all'interno di una pagina JSP ed è possibile ovviamente specificare i parametri da inviare con la richiesta.

È possibile l'inserimento di parametri aggiuntivi per specificare le varie opzioni fornite dallo stesso oggetto PortletUrl, questi parametri sono:

- Il parametro *windowState*
- Il parametro *portletMode*
- Il parametro *var*
- Il parametro *secure*

Un esempio di *renderUrl*:

```
<portlet>
<portlet:renderURL portletMode=”view”
windowState=”normal”>
<portlet:param name=”showUser” value=”gvicino”/> 20
<portlet:param name=”showUser” value=”mrossi”/>
</portlet:renderURL>
```

Questo esempio crea un collegamento specificando la visualizzazione degli utenti “gvicino” e “mrossi” tramite la specifica del parametro showUser, inseriti nell'utilizzo della modalità VIEW e con una visualizzazione della finestra “normale”.

I rimanenti tag sono:

- *namespace tag*
- *param tag*

Tramite l'utilizzo del primo è possibile definire un nome univoco all'interno associato ad elementi della portlet di uscita, come possono essere funzioni e variabili Javascript.
Ad esempio:

```
<A
HREF=”javascript:<portlet:namespace/>myFunc()”>Func</A
>
```

In questa maniera la funzione Javascript acquisisce un identificatore unico all'interno della pagina del portale.

Il secondo tipo di tag ossia quello *param* serve per specificare una coppia di nome e valore per definire un parametro da aggiungere ad una *actionURL* o ad una *renderURL*, ad esempio:

```
<portlet:param name=”myParam” value=”someValue”/>
```

Portlet e sicurezza

In un portale, soprattutto se effettua ecommerce, la sicurezza ha un ruolo decisamente importante: il nuovo standard 168 prevede diverse opzioni base per assicurare l'isolamento e la riservatezza dei dati e le transazioni degli stessi.

Innanzitutto il contenitore della portlet deve occuparsi di determinare quali utenti hanno accesso alla portlet (o a parti di essa) e la possono utilizzare, ad esempio una portlet può rispondere in maniera diversa a seconda che l'utente sia o meno autentificato nel portale. C'è quindi la possibilità di definire dei *ruoli* tramite l'utilizzo già supportato in J2EE per la sicurezza delle servlet.

Inoltre sono presenti i seguenti metodi:

- `getRemoteUser`
- `isUserInRole`
- `getUserPrincipal`

Tramite questi metodi possiamo determinare:

- Il nome dell'utente autentificato che sta utilizzando la nostra portlet.
- Se l'utente fa' parte di un determinato ruolo da verificare.
- Informazioni sull'utente principale loggato e la restituzione dell'oggetto `java.security.Principal` tramite il quale verranno effettuati dei controlli di business logic.

Le opzioni di sicurezza vengono definite nel file `web.xml` dov'è possibile inserire i ruoli che possono essere esaminati e controllati tramite l'uso dei tag `security-role-ref`.

Esempio:

```
...
<security-role-ref>
  <role-name>operator</role-name>
```

```
  <role-link>manager</role-link>
</security-role-ref>
...
```

Sempre nel file `web.xml` è possibile inserire un tag per specificare che la portlet deve girare esclusivamente utilizzando il protocollo *HTTPS* assicurando quindi che le informazioni confidenziali vengano trasmesse in maniera crittata.

Esempio di una Portlet

Al fine di comprendere meglio la specifica JSR 168, in questo capitolo mostrerò un esempio di semplice Portlet chiamata *ImageViewPortlet* che permette di recuperare delle immagini memorizzate sul portale tramite un interrogazione con un FORM e le visualizza all'interno della portlet.

Prima di tutto andiamo ad analizzare il codice della portlet che estende la classe predefinita *GenericPortlet*:

```
...
import javax.portlet.*;
import java.io.IOException;

public class ImageViewPortlet extends GenericPortlet
...
```

Dopodichè passiamo all'analisi dei vari metodi contenuti all'interno del codice, il primo dei quali è il metodo *init()* che nel nostro caso si limita ad eseguire una stampa su console per avvisare l'inizio dell'esecuzione:

```
...
public void init() throws PortletException
{
 System.out.println("ImageViewPortlet Avviata!!");

}
...
```

Ora vediamo la funzione *processAction* che si occupa della gestione delle varie azioni:

```
public void processAction(ActionRequest request,
ActionResponse response) throws PortletException,
java.io.IOException
{
  PortletMode pm = request.getPortletMode();
  if (pm.equals(PortletMode.VIEW))
  {

response.setRenderParameters(request.getParameterMap()
);
```

```
    }
    else if (pm.equals(PortletMode.EDIT))
    {
      String bgColor = request.getParameter("bgColor");
      PortletSession ps = request.getPortletSession();
      ps.setAttribute("bgColor", bgColor);
    }
  }
```

Come si può vedere si effettua un controllo sulla modalità della portlet e a seconda che si richiami la modalità VIEW o EDIT si recuperano i parametri oppure si specifica nella PortletSession i parametri di configurazione come ad esempio il colore dello sfondo.

Andiamo ad analizzare la *doView()*:

```
public void doView(RenderRequest request,
RenderResponse response)
throws PortletException, IOException
{
  PortletSession ps = request.getPortletSession();
  String bgColor = (String)
ps.getAttribute("bgColor");
  if (bgColor == null)
  {
    bgColor =
getPortletConfig().getInitParameter("bgColor");
    ps.setAttribute("bgColor", bgColor);
  }
  response.setContentType("text/html");
  String jspName =
getPortletConfig().getInitParameter("jspView");
  PortletRequestDispatcher rd = getPortletContext().
getRequestDispatcher(jspName);
  rd.include(request, response);
}
```

Questa funzione si occupa di recuperare i parametri di configurazione relativi allo sfondo(se non ne sono stati specificati nella modalità EDIT) e di includere il codice della pagina *view.jsp* che viene specificata nei parametri di configurazione iniziale contenuti all'interno del descrittore *portlet.xml*.

Vediamo ora il codice della *doEdit:*

```
public void doEdit(RenderRequest request,
RenderResponse response) throws PortletException,
IOException
{
```

```
  response.setContentType("text/html");
  String jspName =
getPortletConfig().getInitParameter("jspEdit");
  PortletRequestDispatcher rd =
getPortletContext().getRequestDispatcher(jspName);
  rd.include(request,response);}
```

La *doEdit()* risulta più semplice e come nella precedente funzione di rendering si stabilisce il tipo di contenuto e si include la pagina *edit.jsp*.

Il codice della *ImageViewPortlet* finisce qui, ora andiamo ad analizzare le due semplici pagine JSP che vengono incluse:

- view.jsp
- edit.jsp

la prima contenente il codice per la renderizzazione della modalità VIEW e la seconda contenente il codice di personalizzazione della portlet (EDIT mode).

Vediamo la prima parte del codice della *view.jsp* contenente l'importazione delle librerie e l'inclusione della tag library:

```
<%@ page session="false" %>
<%@ page import="javax.portlet.*"%>
<%@ page import="java.util.*"%>
<%@ taglib uri='/WEB-INF/tld/portlet.tld'
prefix='portlet'%>
```

Si prosegue con la creazione del PortletUrl e la definizione degli oggetti che verranno utilizzati all'interno della pagina specificanti il nome del file dell'immagine che vogliamo vedere e il colore di sfondo attualmente configurato per la visione della pagina:

```
<%
PortletURL url = renderResponse.createActionURL();
String filename =
(String)renderRequest.getParameter("filename");
PortletSession ps = renderRequest.getPortletSession();
String bgColor = (String)ps.getAttribute("bgColor");
%>
```

Il codice seguente presenta il FORM che interroga l'utente sull'immagine richiesta da visualizzare, mostra lo sfondo colorato secondo le opzioni e il codice HTML che recupera il path dell'immagine e la mostra:

```
color = <%=bgColor%>
<form method="post" action="<%=url.toString()%>"
style="background-color: <%=bgColor%>">
<p>Inserisci il nome dell'immagine che desideri
```

```
vedere:</p>
<input type="text" id="filename" name="filename"/>
<br />
<button type="submit">Mostra Immagine</button>
</form>
<br />
<%
filename =
renderResponse.encodeURL(renderRequest.getContextPath(
))+"/images/"+filename;
%>
<img src="<%=filename%>"
alt="L'immagine verrà visualizzata qui">
```

Ora andiamo a vedere il codice della pagina di personalizzazione che tramite una lista di opzioni ci permette di specificare quale colore di sfondo utilizzare nella pagina. Il codice d'intestazione è identico alla pagina precedente:

```
<h1>Edit Mode</h1><br>
Seleziona il colore di sfondo che preferisci:<br>
<%
  PortletURL url = renderResponse.createActionURL();
  PortletSession ps =
renderRequest.getPortletSession();
  String bgColor = (String)ps.getAttribute("bgColor");
%>

<form method="post" action="<%=url.toString()%>">
<input name="bgColor" type="radio" value="white"
<%=bgColor.equals("white") ? "CHECKED" : ""%>/>Bianco
<input name="bgColor" type="radio" value="black"
<%=bgColor.equals("black") ? "CHECKED" : ""%>/>Nero
<input name="bgColor" type="radio" value="blue"
<%=bgColor.equals("blue") ? "CHECKED" : ""%>/>Blu
<input name="bgColor" type="radio" value="red"
<%=bgColor.equals("red") ? "CHECKED" : ""%>/>Rosso
<input name="bgColor" type="radio" value="green"
<%=bgColor.equals("green") ? "CHECKED" : ""%>/>Verde
<br/>

<button type="submit">Cambia colore</button>
</form><br>
Current Portlet Mode:
<%=renderRequest.getPortletMode()%><br>
Current Window State:
<%=renderRequest.getWindowState()%><br>
```

Infine ci resta da esaminare il file descrittore della portlet contenente i parametri iniziali che sono:

- *jspView*: che contiene il path della pagina view.jsp
- *jspEdit*: che contiene il path della pagina edit.jsp
- *bgColor*: che contiene il colore di sfondo predefinito

Vediamo il codice:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app
xmlns="http://java.sun.com/xml/ns/portlet/portlet-
app_1_0.xsd" version="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/portlet
/portlet-app_1_0.xsd">

<portlet>
  <description>Image View Portlet</description>
  <portlet-name>ImageViewPortlet</portlet-name>
  <display-name>Image View Portlet</display-name>
  <portlet-
class>com.sedna.imageviewportlet.ImageViewPortlet</por
tlet-class>

  <init-param>
    <name>jspView</name>
    <value>/jsp/view.jsp</value>
  </init-param>

  <init-param>
    <name>jspEdit</name>
    <value>/jsp/edit.jsp</value>
  </init-param>

  <init-param>
    <name>bgColor</name>
    <value>white</value>
  </init-param>

  <expiration-cache>-1</expiration-cache>
  <supports>
    <mime-type>text/html</mime-type>

    <portlet-mode>VIEW</portlet-mode>
    <portlet-mode>EDIT</portlet-mode>
  </supports>
```

```
  <supported-locale>it</supported-locale>
  <portlet-info>
    <title>ImageView Portlet</title>
    <short-title>ImageView</short-title>
    <keywords>Images, View</keywords>
   </portlet-info>
</portlet>
</portlet-app>
```

Nei primi tag vengono definiti i nomi di libreria e i nomi della portlet, poi successivamente troviamo i parametri iniziali e i loro valori definiti tramite i tag *init-param* ed infine le modalità disponibili in questa portlet. Per ultimi vengono inseriti i parametri legati alla localizzazione e al titolo della portlet.

Come si può vedere la realizzazione di una portlet è molto semplice e presenta molti vantaggi, il primo dei quali la portabilità. Questo nostro esempio potrà essere utiizzato su qualsiasi portale compatibile, e nel caso successivamente ampliato ed esteso per un futuro utilizzo.

Capitolo 5
Analisi dei vari portali

All'inizio del mio periodo di tirocinio non erano ancora a disposizione dell'azienda i vari portali commerciali su cui si sarebbe voluto fare il porting del precedente sistema, perciò si decise di sviluppare la migrazione del primo modulo utilizzando la piattaforma di riferimento *Jakarta Pluto.*

Jakarta Pluto

Jakarta Pluto è un contenitore per portlet Open Source sviluppato dal progetto Apache Jakarta. Questo progetto permette di sviluppare e testare il ciclo di vita della proprie portlet, compresi anche gli aspetti legati alle preferenze, e qualsiasi altro aspetto delle portlet. Dalle ultime versioni è possibile anche l'integrazione con altri frameworks e per l'uso come portale, ma queste funzioni non sono ancora pienamente supportate.

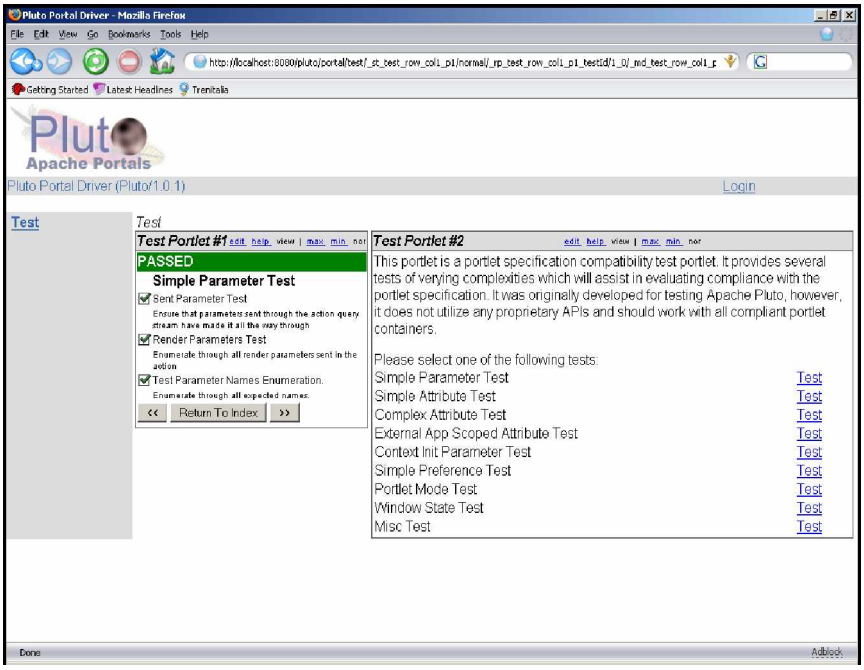

Figura 8: Pluto Portal Driver

L'utilizzo di Pluto e il deploying delle portlet è decisamente comodo e veloce e sfrutta altri due progetti Open Source, il primo, *Maven*, per il

deploying delle applicazioni e il secondo, *Tomcat*, come *Application Server*.

La community del progetto e gli stessi sviluppatori sono molto disponibili e si può trovare facilmente aiuto, questo aiuta a sopperire la mancanza di documentazione che al momento del mio tirocinio caratterizzava il progetto. Al momento attuale la documentazione sta decisamente aumentando e viene fornito molto più supporto.

Jakarta Jetspeed

Sebbene Pluto fornisca dei rudimentali supporti per l'integrazione e l'uso come contenitore per portale, il progetto consigliato dal gruppo *Jakarta* è *Jetspeed*. Questa valida alternativa ai portali commerciali attualmente esistenti è interamente controllata da Java e XML e gestisce con grande efficacia l'aggregazione da fonti disponibili sulla rete.

La specifica JSR 168 è pienamente supportata ed è possibile gestire pagine di portali create dai frammenti *portlet*. Vengono anche forniti diversi tool di supporto, per facilitare la creazione dei portali da parte dell'utente finale.

Figura 9: Jetspeed Portal Server

Il layout e il design della pagina vengono gestiti tramite file di template XSL permettendo una facile manipolazione dell'aspetto di *Presentation Layer*.

Jetspeed può essere integrato con diversi framework di pubblicazione contenuti, CMS e supporta progetti quali Coocoon, Velocity, Turbine, Struts etc.

Viene fornito il supporto per tecnologie *mobile*, è quindi possibile fornire diverse versioni del portale a seconda del tipo di *hardware* utilizzato per vedere la pagina. Il portale sarà visualizzabile sia con un classico browser per PC, che con quello del PDA o via WAP con il cellulare.

Figura 10: Jetspeed su emulatore wap

Concludiamo con i difetti. Anche in questo caso il problema principale è la scarsa documentazione, mancanza grave perché provoca scarso interesse in ambiente aziendale. Il progetto comunque è in crescita e si prefigura come ottima soluzione per portali di media e piccola dimensione.

Sun One Portal Server

Durante il tirocinio abbiamo testato due portali commerciali, il primo di questi è *Sun Java System Portal Server* fornito con il ben più grosso prodotto *Sun One Portal Server* che fornisce diversi strumenti per pressochè ogni aspetto della gestione di un server portale.

La suite contiene un *Application Server*, un *Identity Server*, un *Web Directory Server* e una comoda suite di sviluppo ossia *Sun One Studio*.

Il pacchetto fornito dalla Sun è decisamente voluminoso e difficilmente si può riuscire ad esaminare ogni aspetto che lo compone: nonostante un tecnico Sun abbia cercato in una giornata di spiegarci tutte le sue possibilità, diversi lati ci sono rimasti oscuri dato che non erano utili ai fini della nostra migrazione.

Prima di tutto bisogna sottolineare che tutto l'ambiente è basato sull'utilizzo di *Sun One Directory Server*.

Ricordiamo che con Directory Server intendiamo un database di tipo *gerarchico* in cui i dati vengono memorizzati come *entry* strutturate in

directory (un esempio conosciuto è LDAP). Questo servizio viene utilizzato per la gestione dei dati riguardanti utenti e gruppi, importante notare che *Sun One* è facilmente configurabile al fine di replicare utenti e gruppi presi da altri Identity-Management Service.

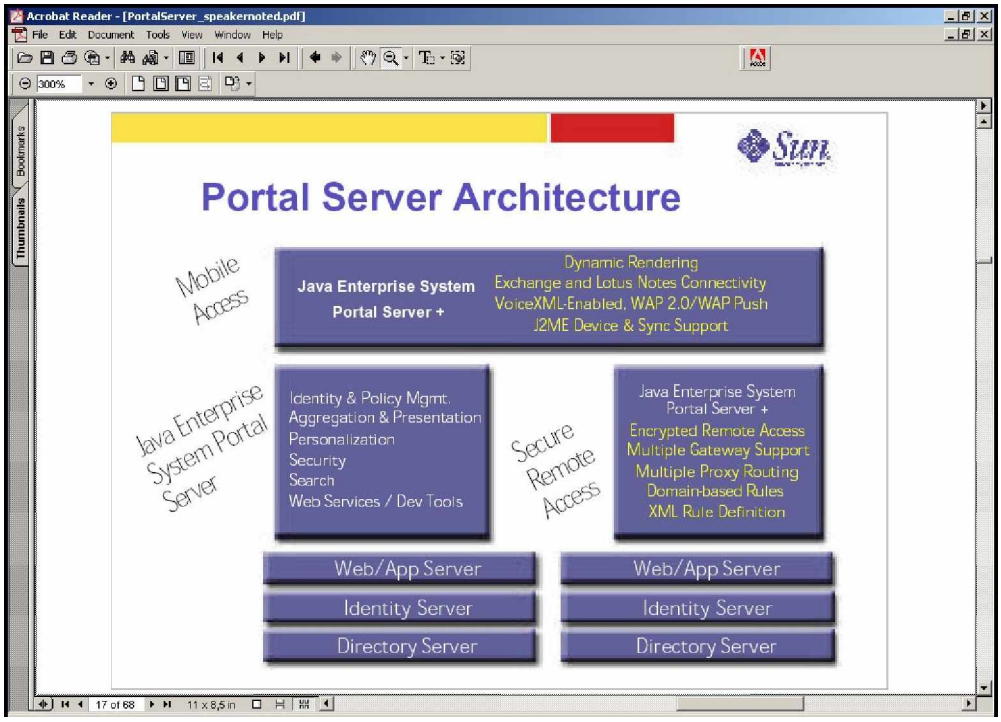

Figura 11: L'architettura di un sistema Sun Portal [5]

Il paradigma su cui si basa questo portale è la personalizzazione. Come amministratore possiamo tramite una classica interfaccia Web:

- Aggiungere ed eliminare portlet
- Personalizzare e definire l'aspetto del layout
- Gestire facilmente i profili utente
- Modificare le proprietà e i settaggi delle portlet.

Lo sviluppo delle portlet è facilitato se si decide di utilizzare lo strumento di sviluppo fornito con il package ossia *Sun Java Studio Portlet Builder*, purtroppo durante il tirocinio è stato solo testato dato che lo sviluppo iniziale era partito con l'utilizzo di *Borland Jbuilder X*.

Il *Builder* fornisce comodi strumenti di creazione guidata (*wizard*), strumenti per la modifica del deployer descriptor (*portlet.xml)*, strumenti per il debugging e l'esecuzione a caldo delle nostre portlet.

L'utilizzo ben integrato del sistema di Web Directory ci fornisce facilmente un modo per supportare diverse visualizzazioni delle nostre portlet a seconda del tipo di utente (meglio visto come client) e dell'hardware che utilizza, rendendoci molto semplice fornire contenuti per PDA o cellulari. Viene fornito anche un sistema di *Istant Messaging* che al di fuori del discorso delle portlet rendono l'utilizzo all'utente del portale decisamente comodo (ad esempio è possibile l'inoltro degli avvisi o dei messaggi su linea telefonica).

Un altro aspetto interessante è l'integrazione dei *Web service* con il portale, è possibile aggiungerli ed eliminarli tramite modifiche a file *XML* senza dover creare manualmente interfacce JSP o Java per ciascuno di essi.

Ultima funzionalità è la presenza di un sistema di ricerca molto avanzato, che permette di condurre indagini nei contenuti del portale senza problemi, fornendo servizi di recupero documenti, strumenti linguistici e navigazione avanzata.

In conclusione sicuramente un ottimo portale, il difetto principale è la sua dimensione che può risultare adatta a sistemi e progetti di grosse dimensioni ma a volte può confondere l'utilizzatore inesperto.

Hummingbird Portal

L'altro portale commerciale visto durante il tirocinio è stato quello della Hummingbird che però ha come target principale l'utilizzo non tanto sul Web ma come sistema informativo principale nelle aziende. Sintetizzando si può dire che le potenzialità di questo portale sono principalmente le idee di una gestione rispettante la *business logic*, si propone infatti come tecnologia di base per realizzare un *Enterprise Information Management System.*

Il portale dotato di molte funzionalità offre sistemi d'integrazione con diversi linguaggi compresi BML, Javascript, VbScript, Active Pearl, Jpython, JACL. Il supporto per la specifica JSR168 dalla versione Enterprise 2004 è reso decisamente facile ed intuitivo, e si propone come una delle soluzioni più “user friendly” per gli amministratori.

Viene inoltre fornito supporto non solo per la tecnologia Java ma anche per quella .NET tramite l'utilizzo di un sistema di plugin denominato e-Clip.
Il prodotto dispone di sistemi di ricerca avanzata, ricerche *concept based*, ricerche *full text*, ricerche su server multipli, ricerche parametriche e logica fuzzy.

Purtroppo il prodotto non può fornire suggerimenti di ricerca per informazioni aggiuntive che possano essere riferite alla ricerca originale.

Un difetto presente in questo portale è l'assenza di una interfaccia grafica per lo sviluppo applicativo, questo comporta il dovere sviluppare con due ambienti di produttori diversi.

Un altro punto forte di questo portale sono le diverse opzioni di sicurezza disponibili che comprendono gestione di indici e livelli di sicurezza per i documenti, gestione delle password, crittazione dei dati ed integrazione delle security list precedenti.

Il portale della Hummingbird si prefigura come sicuro, scalabile e di facile amministrazione. L'interfaccia è semplice sia per gli utenti che per gli amministratori. Sicuramente è un prodotto di tipo mirato ed un ottimo sistema informativo ma si prefigura appunto per un utilizzo aziendale e non per la realizzazione di generici portali su Internet.

IBM WebSphere

L'IBM ha sviluppato una suite di applicativi denominata WebSphere al fine di fornire un insieme di strumenti per la gestione di siti WEB ad alte prestazioni con la possibilità di integrare nuovi e vecchi sistemi.

Viene quindi fornito un Enterprise Information Portal denominato *WebSphere Portal Server* che fornisce un accesso centralizzato ad informazioni e servizi.

All'interno del portale si trovano i processi, le interazioni, le applicazioni e i contenuti tipici del ciclo di vita aziendale.

Il portale dell'IBM permette di organizzare e gestire template e aspetto editoriale del sito fornendo strumenti indispensabili quali sistemi di ricerca, gestione dei contenuti, sistemi di registrazione degli eventi, supporto per dispositivi mobili ed altro ancora.

La costituzione della suite IBM è molto eterogenea, vengono forniti diversi strumenti come per il portale della SUN al fine di provvedere ai bisogni di sviluppatori, amministratori ed utenti.

L'ambiente è così formato:

- *WebSphere Portal Server Framework:* L'intelaiatura portante di tutto il sistema IBM, vengono fornite le funzioni di integrazione tra applicazioni e fonti di dati. Si offrono anche sistemi per la registrazione e l'accesso al portale, per i servizi di connettività e per la presentazione.
- *WebSphere Application Server:* Si propone come il cuore applicativo del framework dove vengono gestite Enterprise Java Bean e gli altri Web

service.

- *WebSphere Portlet:* La specifica JSR 168 viene pienamente supportata e vengono forniti svariati strumenti per lo sviluppo e il caricamento delle portlet.
- *IBM Tivoli Directory Server:* Vengono forniti i servizi di LDAP e autentificazione sicura al portale.

L'intero sistema può essere ampliato tramite l'integrazione con sistemi di gestione contenuti quali Sametime, MindSpam, QuickPlace e Lotus Email.

A fianco del portale vengono forniti potenti strumenti di sviluppo riuniti all'interno del prodotto denominato WebSphere Studio che permette di sviluppare e gestire qualsiasi aspetto creativo, dalla pubblicazione WEB, allo sviluppo di Web service e all'integrazione con applicazioni J2EE. Questi strumenti sono stati ampliati a partire dal progetto OpenSource conosciuto come Eclipse, uno strumento di sviluppo Java che si sta imponendo molto velocemente creando una concorrenza rilevante ai prodotti Borland e Sun.

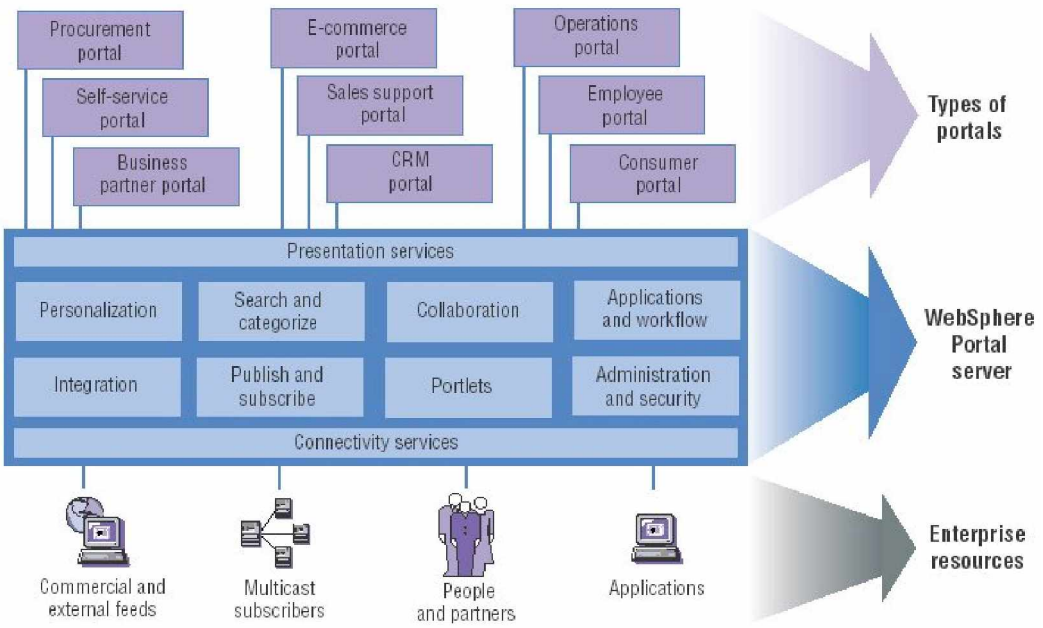

Figura 12: Architettura IBM WebSphere [7]

Websphere Portal offre un numero limitato di portlet pre integrate, la maggior parte riguardanti la collaborazione e la gestione dei contenuti e l'integrazione con servizi Lotus (Domino/Notes). Vengono fornite anche

alcune portlet per la gestione di documenti d'ufficio di prodotti Microsoft. Come molti suoi concorrenti fornisce sistemi di Istant Messaging come Lotus Sametime o MSN.

WebSphere Portal è un ambiente intuitivo e benchè le sue opzioni a disposizione siano limitate, l'utilizzo come portale base è realizzabile facilmente dopo poche ore a differenza di prodotti di taglia maggiore.

Confronti tra le varie proposte analizzate

Tra i vari portali sicuramente spiccano quelli della Sun e quelli della IBM, il primo adatto a portali di grosse dimensioni e di grande complessità, il secondo adatto per portali di medie dimensioni.

Questi due portali sono quelli che sicuramente supportano in maniera migliore la specifica JSR168 fornendo tutti i dettagli per sfruttarla al massimo.

I due prodotti OpenSource sono sicuramente di buona qualità purtroppo però essendo progetti recentemente nuovi soffrono di diversi difetti prima di tutto dal punto di vista della documentazione fornita, ma per piccoli progetti sono sicuramente performanti ma soprattutto gratis, permettendo di utilizzare la nuova specifica anche agli utenti della rete che non dispongono di grandi risorse economiche.

Un altro portale che ho potuto provare e di cui sarebbe interessante un analisi più approfondita è il portale uPortal, un prodotto Opensource mirato all'utilizzo in ambito accademico.

Capitolo 6
Realizzazione del portale Web

Il progetto di stage venne denominato *Sedna* e aveva come obiettivo la realizzazione di una migrazione di un portale aziendale pre-esistente con diverse funzionalità sviluppato tramite *Apache Struts* verso un sistema *Sun One Portal* utilizzante tecnologia *JSR 168*. In questo capitolo parlerò brevemente di come si è svolto questo progetto e di quali sono stati i problemi sia in fase di analisi che in fase di sviluppo.

La progettazione e il coordinamento della migrazione sono stati gestiti dal nostro tutor all'interno dell'azienda. Lo sviluppo e la realizzazione del portale sono state invece affidate al realizzatore di questa tesi e ad un compagno universitario di stage.

Durante il tirocinio abbiamo applicato in parte le metodologie di sviluppo conosciute sotto il nome di *Extreme Programming*. Abbiamo in prima analisi discusso del progetto con il tutore e si sono definite nella prima settimana le *specifiche dei requisiti* e le *specifiche di progetto*.

Subito dopo abbiamo realizzato uno studio preliminare sia della tecnologia Struts che di quella Portlet, sconosciute sia per me che per l'altro stagista in quel momento.

Infine abbiamo realizzato e sviluppato il porting dei vari moduli man mano procedendo al testing e alla revisione delle specifiche nel caso fossero avvenuti errori concettuali nella scrittura di queste ultime.

La realizzazione dell'intera migrazione è durata circa tre mesi, nei quali abbiamo lavorato spesso in coppia ma anche singolarmente su vari moduli.

Descrizione del portale esistente e delle aree applicative

Il portale aziendale della IBPS voleva fornire un comodo strumento intranet per la gestione di contatti, cataloghi, scadenze, progetti e documenti all'interno di una media o piccola azienda.

Il passo principale nella definizione delle specifiche è stato quello di suddividere quest'applicazione in cinque moduli o aree da sviluppare.

Il portale era scomponibile nelle seguenti aree:

- *Area Rubrica:* Questo modulo presentava una rubrica per la memorizzazione dei contatti aziendali, permettendone l'inserimento, la modifica, la cancellazione. Erano presenti ovviamente anche sistemi per la ricerca e l'ordinamento dei contatti.
- *Area Procurement:* Quest'area forniva un semplice sistema di gestione delle transazioni di *procurement,* ossia di tutta la gestione di acquisto e vendita di prodotti interni ed esterni all'azienda. Dal browser era quindi possibile aggiungere i vari prodotti, i fornitori, creare delle categorie per essi e formare dei modelli per effettuare le varie richieste d'acquisto alle aziende.
- *Area Progetti:* Tramite quest'area del portale era possibile creare delle aree di progetto al fine di tenere traccia delle varie comunicazioni tra l'azienda ed i partner relative alle transazioni in corso. Questo modulo fungeva quindi da semplice *Customer Relationship Management (CRM),* ed è strettamente legato ai due moduli successivi ossia allo scadenzario e al documentario.
- *Area Scadenze:* Legata all'area precedente, svolge il ruolo di scadenziario permettendo la creazione di filtri per categorie di scadenze mostrando le scadenze attive e quelle ormai concluse.
- *Area Documenti:* Questo modulo permette l'inserimento di dati non strutturati quali fogli MS Excel, documenti MS Word, note, email e ogni tipo di documento e poteva associare documenti ad aree progettuali.

La memorizzazione dei dati di rubrica, delle scadenze e di tutti gli altri dati erano effettuate tramite query SQL sul un database fornito insieme al portale.

L'accesso al portale avveniva tramite un apposito sistema di autentificazione realizzato dai precedenti sviluppatori, e non erano previste regole particolari relativi ai ruoli degli utenti registrati.

Nel processo di migrazione si è volutamente ignorato la gestione dell'autentificazione, in quanto questa sarebbe stata facilmente gestibile tramite i sistemi automatici del portale. Nel caso si fosse voluto fornire sistemi di personalizzazione e visualizzazione mirata di ciascuna portlet a seconda del ruolo dell'utente che la stava utilizzando non si sarebbe dovuto far altro che estendere il codice per ciascun modulo.

Caratterizzazione dell'architettura esistente

L'architettura del portale esistente era realizzata tramite componenti diversi ma non sfruttava ancora le possibilità offerte dai nuovi portali supportanti la specifica JSR168.

Principalmente si faceva utilizzo di tre elementi principali questi elementi erano:

- *Apache Tomcat 4.1*
- *Jakarta Struts*
- *MySQL*

Tutta la gestione delle operazioni di basso livello quali l'interfacciamento al database, l'esecuzione delle query, le classi riguardanti i vari oggetti e il data level erano implementati tramite l'uso di un insieme di librerie sviluppate dalla IBPS denominate *alchemi-mainlib.*

Il portale esistente era stato realizzato tramite l'utilizzo dell'ambiente di sviluppo della Borland denominato Jbuilder. Al fine di poter manipolare e studiare meglio l'architettura precedente abbiamo deciso di procedere con lo sviluppo del nuovo portale utilizzando anche noi questo ambiente.

Durante il periodo di tirocinio abbiamo provato a spostarci vero la suite della Sun, ma per questioni di tempo si è solo testato brevemente il suo funzionamento.

All'inizio durante l'analisi del codice già esistente c'eravamo interrogati su come gestire una così grande quantità di dati, dato che a livello accademico non avevamo mai gestito un progetto software così complesso.

Fortunatamente man mano che procedevamo nello studio ci siamo resi conto che le strategie e il codice precedente erano riutilizzabili e riadattabili per un nuovo progetto.

Problemi e sviluppo della migrazione

Il problema principale della migrazione è stato l'abbandono completo del framework Struts, questa decisione non era obbligata ma avrebbe permesso di realizzare un ambiente più pulito e di provare sul campo l'utilizzo della singola tecnologia Portlet.

Il problema è che abbandonando completamente questo ambiente si sono dovute effettuare molte modifiche strutturali.

All'inizio si è dovuto operare per decidere quali Portlet sarebbero state create e messe a disposizione del portale.

Inizialmente si era pensato di suddividere le varie aree del portale esistente

in più portlet.

Ad esempio il primo modulo Rubrica prevedeva la seguente suddivisione:

- *PortletRicerca:* Che avrebbe permesso la ricerca dei contatti nella rubrica.
- *PortletContatto:* Che avrebbe fornito le maschere per la modifica e gestione dei contatti.
- *PortletAzienda:* Che avrebbe fornito le maschere per la modifica e la gestione delle aziende.

Si pensava quindi di fornire per ogni *use case* una portlet singola, man mano che procedevamo nello sviluppo ci siamo però resi conto che questo avrebbe creato maggiormente problemi.

La nuova tecnologia JSR168 infatti per adesso non supporta un sistema efficiente di scambio messaggi tra i vari oggetti Portlet, abbiamo quindi deciso di realizzare soltanto cinque portlet ciascuna associata alle varie aree di progetto.

In questa maniera il nostro portale diventava anche più espandibile, un cliente poteva decidere quali portlet comprare o utilizzare nel proprio portale, si sono quindi realizzate le seguenti portlet:

- *RubricaPortlet*
- *ProcurementPortlet*
- *ProgettiPortlet*
- *ScadenzePortlet*
- *DocumentiPortlet*

Lo sviluppo della prima portlet è stato quello determinante perché è stato quello in cui abbiamo deciso e provato la nostra politica implementativa e di migrazione. La realizzazione del primo modulo portlet è stata fatta in coppia, successivamente ci siamo divisi individualmente nella attività di elaborazione, ciascuno realizzando la migrazione di una diversa area.

La seconda decisione che abbiamo preso è stata quella riguardante alla suddivisione logica dei vari strati, ossia quelli di *presentazione*, di *business logic* e di *data & service layer*.

Relativamente al primo strato abbiamo deciso al fine di riutilizzare più codice possibile modificando le pagine JSP già esistenti.

Purtroppo questa migrazione non è stata facilissima in quanto Struts utilizza per le proprie pagine JSP delle *tag library* altamente specializzate al fine di velocizzare lo sviluppo di questo ambiente rendendoci impossibile un

riutilizzo completo.

Nella nostra implementazione la portlet dopo aver analizzato la richiesta proponeva all'utente una pagina JSP di risposta, tramite l'utilizzo del metodo *doView,* di cui propongo un breve segmento:

```
public void doView(RenderRequest request,
RenderResponse response) throws
     PortletException, IOException
{
  _contentPage = getContentJSP(request);
  response.setContentType
   (request.getResponseContentType());
  PortletRequestDispatcher dispatcher =
    _pContext.getRequestDispatcher(_contentPage);
  dispatcher.include(request, response);
}
```

A seconda delle azioni che erano state effettuate determinavamo la pagina da visualizzare tramite un metodo *getContentJSP* e dopodichè richiedevamo al dispatcher di visualizzarci il frammento markup in questione.

Lo strato di *business logic* è quello che ci ha creato maggiormente problemi in quanto abbiamo dovuto realizzare delle classi logiche per la validazione e la gestione dei dati.

Struts come abbiamo spiegato nel secondo capitolo gestisce tutta la parte di MVC tramite l'utilizzo di classi specializzate che ruotano tutte attorno al sistema di gestione delle azioni.

Questo sistema di azioni non era facilmente portabile con la tecnologia Portlet, abbiamo quindi creato delle nostre classi logiche opportune che fossero facilmente utilizzabili con le portlet, eliminando tutti i riferimenti e tutti gli utilizzi alle classi specializzate del framework precedente.

La gestione delle azioni è stata delegata alla funzione *processAction* della portlet che dopo aver recuperato la richiesta andava a chiamare i vari tipi di azione:

```
portletession session = request.getportletession();
action = request.getParameter("action");

...

if(isActionValid(action) == true)
     if(action.equalsIgnoreCase("edit"))
```

```
        {
              archLoad(request, response, "edit", id,
                arId, yyId);
        }
        else if (action.equalsIgnoreCase("cancel"))
        {
              session.setAttribute("nextPage", _listPage);
              resetTempData(session);
        }
        else if (action.equalsIgnoreCase("file_upload"))
...
```

Il livello di *data level* è stato quello che ci ha posto meno problemi, dato che l'utilizzo delle librerie già esistenti svolgeva già i compiti di interrogazione e memorizzazione dei dati nel database.

Effettuate alcune modifiche al fine di integrare il codice precedente con quello nuovo, abbiamo dovuto realizzare delle espansioni di alcune classi in quanto ci sono state richieste delle funzioni aggiuntive ai vari moduli.

Testing dei vari moduli e dei vari portali

Man mano che procedevamo nella realizzazione dei vari moduli ci è stato richiesto di effettuarne il testing e tenere traccia dei vari risultati tramite dei fogli di calcolo, al fine di mantenere traccia dei nostri risultati nel tempo.

Il testing, quindi, doveva essere mirato ad ogni singolo *use case*: questa metodologia ci ha permesso di modificare repentinamente i vari errori e di poter lavorare e scambiarci il codice senza perdere tempo nel analizzare aspetti già visti.

Riflessioni finali

Alla fine siamo riusciti a completare la migrazione di tutti i componenti del portale, purtroppo non siamo riusciti a testare l'effettiva portabilità promessa dalla tecnologia JSR168 su vari ambienti. Il funzionamento è stato quindi testato limitatamente ai portali della Sun e all'ambiente Pluto con buoni risultati. Le varie aree risultano quindi portlet indipendenti che possono essere utilizzate a seconda del proprio bisogno.

Dal mio punto di vista questa migrazione non ha portato solo vantaggi quali portabilità ma anche alcuni aspetti negativi.

Prima di tutto realizzare dei moduli portlet leggeri che possano essere spostati tra varie architetture di produttori diversi è molto difficile in quanto comunque i livelli sottostanti a quelli di presentazione devono essere gestiti da diverse librerie e sono una commistione di diversi strati logici.

La gestione delle azioni relativamente alla portlet porta spesso un'appesantimento del codice.

Quella che forse sarebbe stata la soluzione ideale era il non eliminare l'utilizzo di Struts ma di integrare entrambe le tecnologie, al fine di poter rendere il codice più leggero e portabile.

Con questo intendo che per grossi progetti utilizzare solo la tecnologia Portlet per la gestione delle azioni, della presentazione non crea degli oggetti piccoli e semplici ma delle applicazioni forse troppo voluminose.

Capitolo 7
Conclusione e sviluppi futuri

Al momento in cui scrivo stiamo assistendo ad una sempre maggiore proliferazione della tecnologia JSR168, i vari produttori continuano a offrire proposte commerciali nuove e dall'altro lato nascono proposte OpenSource sempre più interessanti come ExoPlatform e GridSphere.

Sicuramente questa nuova tecnologia ha riscosso un enorme successo e presenta innumerevoli vantaggi pratici e per molto tempo ne sentiremmo ancora parlare.

Durante lo stage e nei periodi successivi ho potuto farmi un'idea sia sull'uso in azienda delle proposte commerciali sia sullo utilizzo di quelle OpenSource e sono arrivato ad alcune conclusioni che elencherò in seguito.

Le soluzioni commerciali offrono ambienti quasi sempre completi e studiati al fine di soddisfare qualsiasi esigenza degli utilizzatori, ma il difetto molto spesso sta in questo aspetto.

L'utilizzo di ambienti complessi come Sun One Portal non è quasi mai ottimale e ci si riduce ad utilizzare e a sfruttare una minima parte di quello che offrono. Questi grossi portali sono perfetti per grossi progetti, ma per realizzazione di portali di medie dimensioni risultano eccessivi.

Il vantaggio dei prodotti OpenSource è principalmente dovuto al fatto che i vari team di sviluppo non si devono preoccupare delle licenze, risparmiando quindi sui costi, d'altro canto però la mano d'opera dev'essere maggiore e più specializzata, causando problemi a chi voleva soluzioni veloci e facilmente gestibili.

Questo stage mi ha permesso di approfondire la mia modesta conoscenza del linguaggio Java e dall'altra parte mi ha dato la possibilità di capire e comprendere le metodologie e le tecniche utilizzate in campo lavorativo.

La realizzazione di portali Web è sicuramente un argomento di grande interesse nelle aziende, e anche in ambito accademico dato che la comunicazione di contenuti ed informazioni è un bisogno essenziale che ha dato vita forse all'informatica stessa.

L'utilizzo di questa nuova tecnologia semplificherà sicuramente la vita a molti sviluppatori permettendo di spostare maggiormente l'attenzione su quello che realmente conta ossia il servizio all'utente e non le problematiche di sviluppo.

Elenco Figure

Bibliografia

1. Danny Ayers, Hans Bergsten, Michael Bogovich, Jason Diamond, Matthew Ferris, Marc Fleury, Ari Halberstadt, Paul Houle, Piroz Mohseni, Andrew Patzer, Ron Phillips, Sing Li, Krishna Vedati, Mark Wilcox, Stefan Zeiger.*Professional Java Server Programming Word Press 1999*
2. Alejandro Abdelnur, Elain Chien, Stefan Hepper
   JSR-000168 Portlet Specification (Final Release)
   16 October 2004
3. Puliti, Venditti, Giovannini, Cerquetti, Bigatti, Morello, Rossini
   Manuale pratico di Java 2 – Seconda edizione
   Mokabyte 2003
4. Lori MacVittie
   Portal Power
   CMP Media LLC 2004
5. Sang Shin
   Java Portlet (JSR 168)
   Sun Microsystem 2003
6. Sun Microsystem
   Sun One Portal 2004
   http://www.sun.com
7. IBM
   WebSphere Software 2004
   http://www.ibm.com
8. Hummingbird
   Hummingbird Portal 2004
   http://www.hummingbird.com
9. Apache Portals Pluto
   Apache Jakarta Pluto 2004
   http://portals.apache.org/pluto/
10. Apache Jestspeed
    Apache Jakarta Jetspeed Overview 2004
    http://portals.apache.org/jetspeed-1/

11. Image Viewer Example
Kenneth Ramirez
http://www.sys-con.com/story/?storyid=46966&DE=1

12. JSR 168, WSRP, Portlets & Enterprise Portal
Autori Vari
http://portlets.blogspot.com/