UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Facoltà di Ingegneria di Modena

Corso di Laurea Specialistica in Ingegneria Informatica

Progetto e realizzazione di Servizi Web basati su tecnologia Portlet

Relatore:
Char.mo Prof. Sonia Bergamaschi

Candidato:
Roberto Pellegrino

Anno Accademico 2005/2006

Parole chiave:

Web Services

Portals

Portlet

Portal Bridging

Struts Bridge

INDICE

Indice delle Figure

1. Introduzione

Lo Scopo della tesi è duplice. In primo luogo lo studio e la sperimentazione del Framework Struts e della tecnologia di Bridging per lo sviluppo di applicazioni nell'ambito dei Portali. In secondo luogo lo studio e la valutazione dello standard WSRP (Web Services for Remote Portlet) (con particolare attenzione a tutte le componenti che intervengono in questa specifica) per lo sviluppo di Portlet.
La sperimentazione ha portato allo sviluppo di due applicazioni e alla loro valutazione in termini di robustezza e facilità d'uso.
La tesi si articola nei seguenti capitoli.
Nel Capitolo 2 verranno introdotte le caratteristiche delle Portlet facendo riferimento alla specifica JSR-168. Successivamente, verrà affrontato lo studio dello standard WSRP preceduto da una breve descrizione dell'architettura SOA (Service-Oriented Architecture) e dagli standard classici associati ai Web Services quali: WSDL (Web Services Description Language), SOAP (Simple Object Access Protocol) ed UDDI (Universal Description Discovery and Integration). Un paragrafo a parte verrà riservato per il progetto WSRP4J, il quale è stato integrato in Pluto per costituire il Portale "Producer" così da permettere a Portali "Consumer" di accedere alle Portlet messe a disposizione mediante lo standard WSRP. Seguiranno poi tre paragrafi dedicati ai Portali utilizzati. Per primo si tratterà Pluto mettendone in luce le principali caratteristiche e i limiti riscontrati. In secondo luogo si avrà la descrizione di uPortal e Liferay utilizzati come Portali "Consumer" evidenziandone le limitazioni incontrate durante il loro utilizzo.
Nel Capitolo 3 verranno introdotte le caratteristiche principali del Framework Struts che permette lo sviluppo di Web-Application seguendo il pattern MVC (Model-View-Controller). Successivamente sarà affrontato l'argomento dello Struts Bridging il quale permette l'utilizzo di una normale Web-Application sviluppata tramite Struts anche come Portlet. Di questo "dual mode" di funzionamento ne verranno approfondite le caratteristiche positive ma anche le limitazioni incontrate.
Nel Capitolo 4 verranno presentate le due applicazioni sviluppate. Nel primo paragrafo verrà descritto lo schema E/R e relazionale del Database utilizzato come supporto. Verrà poi illustrata l'applicazione di Amministrazione (sviluppata con Struts e lo Struts Bridge) con particolare riferimento ai DataView e alle Azioni. Successivamente

si avrà la descrizione della Portlet di Pubblicazione che verrà resa disponibile mediante lo standard WSRP.
Nel Capitolo 5 verranno esposte le conclusioni e gli sviluppi futuri.
Nel Capitolo 6 verrà riportato il glossario composto dai termini e dagli acronimi più significativi, seguiti da una breve descrizione.
Nel Capitolo 7 verrà riportata la bibliografia. Ogni articolo o libro inserito sarà seguito da una breve descrizione che ne giustifica la citazione nell'ambito del lavoro svolto.

2. WSRP

2.1 Specifica JSR-168

2.1.1 Introduzione

Le Portlet sono un componente definito dalla specifica JSR-168 (Java Specification Request) come:
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 (vedi bibliografia**[6]**).
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 (browser) 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 l'esecuzione di 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 reindirizzarle 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 autenticato 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 verranno inclusi nella pagina risultante.
- Il portale aggrega i vari markup prodotti dalle Portlet e spedisce al browser la pagina di risposta.

Per chiarire i concetti appena descritti viene riportata una figura la quale mostra il passaggio di consegne tra Portlet Container (o contenitore), Portale e Client Device (o browser).

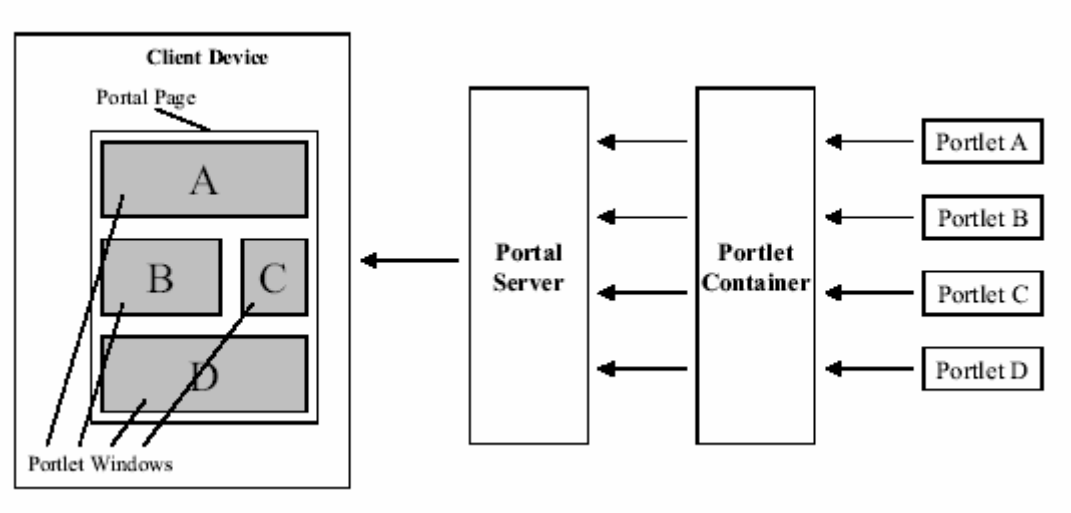

Figura 1: Creazione di una pagina del portale

Le Portlet non vanno confuse con le servlet, per questo vengono riportate le somiglianze e le differenze.

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 Portlet Container
- 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 che successivamente il portale si occuperà di gestire l'aggregazione dei vari frammenti 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 delle servlet comprendente `ActionRequests` e `RenderRequests`.
- Le Portlet hanno delle modalità predefinite (`view`, `edit`, `help` mode) e degli stati delle finestre che ne definiscono la dimensione(minima,massima,normale).
- 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 (vedi bibliografia**[8]**).

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 utilizzando la *request dispatcher* (vedi bibliografia **[7]**).

Per capire meglio il concetto di Portlet viene mostrato 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 qua sotto.

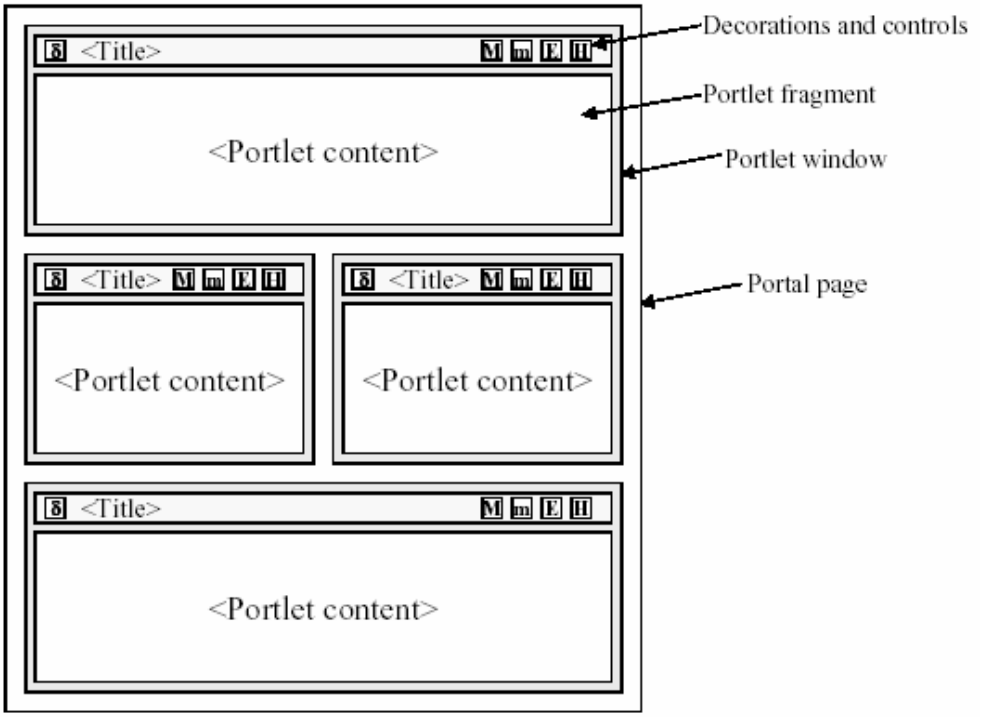

Figura 2: Componenti Portlet

Come esempio viene riportata una pagina di Liferay (Portale Open Source descritto nei paragrafi seguenti) con la presenza di alcune Portlet.

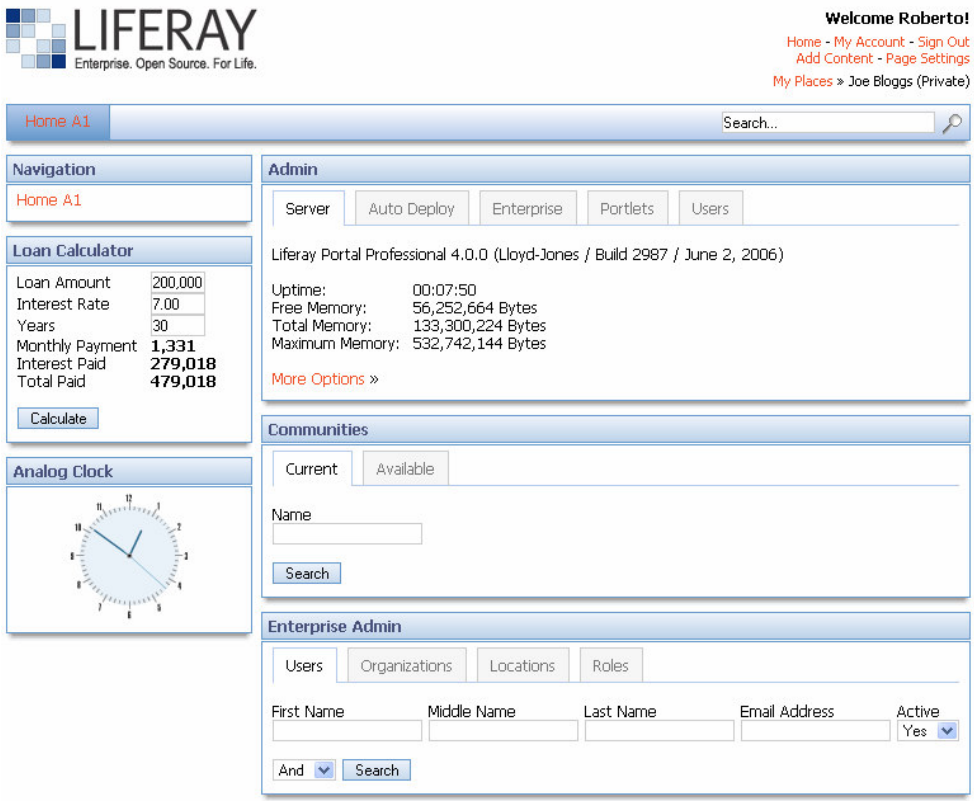

Figura 3: Esempio di Portal page

2.1.2 Portlet e GenericPortlet

L'oggetto principale che caratterizza la nuova specifica è l'interfaccia `Portlet` la quale è la principale astrazione delle Portlet API. Tutte le Portlet implementano questa interfaccia direttamente o estendendo una classe che la implementa.
Le Portlet API includono la classe `GenericPortlet` che implementa l'interfaccia appena descritta e mette a disposizione le funzionalità di default. Solitamente gli sviluppatori estendono direttamente o indirettamente la classe `GenericPortlet` per l'implementazione delle proprie Portlet.
Il ciclo di vita della Portlet è gestito principalmente da quattro metodi che vengono chiamati direttamente dal Portlet Container, questi metodi sono:

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

Vediamone l'utilizzo:
La `init()` viene chiamata quando la Portlet viene instanziata 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 `doHelp()` 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.

2.1.3 Portlet Mode e Stato Finestra

Ogni contenitore deve gestire per ogni Portlet il *portlet mode* ed il *window state*.
Il *window state* indica 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 potrà decidere di utilizzare questa informazione per decidere quante informazioni renderizzare (vedi bibliografia**[9]**).
Come abbiamo già accennato, esiste una modalità specifica a seconda della funzione eseguita in quel momento dalla Portlet. La specifica JSR 168 definisce tre modalità standard di funzionamento delle Portlet:

- `View`
- `Edit`
- `Help`

La disponibilità di queste tre modalità può cambiare a seconda del ruolo posseduto dall'utente del portale. Per esempio potrebbe essere permesso solamente all'amministratore di accedere in `edit` mode, mentre per i normali utenti rendere disponibile solamente la modalità `view` ed `help`.

View Mode

La funzionalità che si attende per una Portlet in `view` mode è quella di generare il markup che riflette lo stato corrente della stessa.
Per esempio, una Portlet in `view` mode potrà includere uno o più screens che l'utente potrà navigare ed interagire, oppure includere contenuti statici che non richiedono alcuna interazione con l'utente.
Gli sviluppatori di Portlet dovranno implementare la modalità `view` sovrascrivendo il metodo `doView` della classe `GenericPortlet`.

Da notare che la modalità `view` è l'unica obbligatoria.

Edit Mode

Con la modalità `edit`, la Portlet dovrebbe offrire i contenuti e la logica che permetta all'utente di personalizzarne il comportamento. L'`edit` mode potrà includere uno o più screens attraverso i quali gli utenti navigheranno per inserire le proprie modifiche. Tipicamente in questa modalità vengono settate le `PortletPreferences`.

Gli sviluppatori di Portlet a seconda delle proprie necessità potranno implementare la funzionalità `edit` sovrascrivendo il metodo `doEdit` della classe `GenericPortlet`.

L'implementazione di questo metodo non è obbligatorio.

Help Mode

L'help mode permette di avere informazioni di aiuto sulla Portlet, le quali possibilmente dovranno essere simili ad un semplice help che spieghi gli obiettivi della Portlet stessa.

Gli sviluppatori potranno implementare le funzionalità dell'help mode sovrascrivendo il metodo `doHelp` della classe `GenericPortlet`.

Come avviene per il metodo `doEdit`, che il `doHelp` non è obbligatorio quindi è a discrezione dello sviluppatore implementarlo o meno.

Custom Portlet Modes

I portal vendors possono definire delle modalità di funzionamento personalizzate sulla base delle funzionalità specifiche che intenderanno offrire. Le Portlet potranno utilizzare le modalità definite dal portale, non solo, è permessa anche la definizione di custom mode che verranno utilizzate previa definizione attraverso il tag `customportlet-mode` nel deployment descriptors (vedi bibliografia **[10]**).

A tempo di deploy le custom mode definite verranno mappate nei custom mode supportati dall'implementazione del portale.

Se un custom mode definito nel deployment descriptor della Portlet non è mappato in un custom mode offerto dal portale, la Portlet non potrà invocare quella particolare modalità.

A titolo esemplificativo viene riportato un deployment descriptor che definisce due diversi custom mode chiamati `clipboard` e `config`.

```
<portlet-app>
     <custom-portlet-mode>
          <description>
```

```
                Creates content for Cut and Paste
          </description>
          <name>clipboard</name>
     </custom-portlet-mode>
     <custom-portlet-mode>
          <description>
               Provides administration functions
          </description>
          <name>config</name>
     </custom-portlet-mode>
</portlet-app>
```

Da tenere in considerazione il fatto che se la Portlet mette a disposizione dei custom mode andrà sovrascritto il metodo `doDispatch` della classe `GenericPortlet`.

2.1.4 Definizione dei Portlet Mode supportati

Ogni Portlet potrà descrivere nella propria definizione attraverso il deployment descriptor le modalità di cui offrirà il supporto. Come già detto in precedenza tutte le Portlet dovranno supportare la modalità `view` e per questo potrà essere omessa nel deployment descriptor poiché ritenuta obbligatoria.
Tutte le altre modalità andranno definite in modo appropriato nel file di configurazione poiché se non venissero definite non sarebbero utilizzabili.
Per chiarire ciò che è appena stato descritto viene riportato un pezzo di deployment descriptor che definisce (oltre al `view` mode) due modalità : `edit` ed `help`.

```
<supports>
     <mime-type>text/html</mime-type>
     <portlet-mode>edit</portlet-mode>
     <portlet-mode>help</portlet-mode>
</supports>
<supports>
     <mime-type>text/vnd.wap.wml</mime-type>
     <portlet-mode>help</portlet-mode>
</supports>
```

Per il markup HTML questa Portlet supporta l'edit e l'help mode (in aggiunta al view mode che è obbligatorio), per quanto riguarda il markup WML supporta il view e l'help mode. Il Portlet Container potrà ingnorare tutti i riferimenti alle modalità custom che non sono supportate dall'implementazione del portale.

2.1.5 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.

2.1.6 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 (vedi bibliografia **[11]**). Questa è un aspetto molto utile nelle Portlet poiché se venisse utilizzata la sessione per la memorizzazione delle preferenze, queste ultime verrebbero perse irrimediabilmente al momento del logout dell'utente o dopo un restart del portale, mentre le `PortletPreferences` vengono memorizzate su dispositivi di memorizzazione permanenti (es: hard disk) superando le limitazioni che sarebbero legate all'uso della sessione.
I metodi forniti da questa interfaccia sono:

- `getNames`
- `getValue`
- `setValue`
- `getValues`
- `setValues`
- `getMap`

- isReadOnly
- reset
- store

Solitamente per settare sei singoli valori vengono utilizzati i metodi `getValue` e `setValue`. In più è possibile dichiarare un attributo come ReadOnly ovvero di cui non è possibile modificare il valore assegnato di default, nel caso in cui venga invocato un `setValue` / `setValues` o `reset` verrà lanciata un'eccezione del tipo `ReadOnlyException`.

Da notare che la Portlet potrà modificare le proprie preferenze solamente durante l'invocazione del `ProcessAction`, infatti se il metodo `store` venisse invocato durante un render verrebbe lanciata un'eccezione di tipo `IllegalStateException`.

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`. Un altro aspetto da considerare per questo metodo è che nel caso in cui gli attributi da memorizzare siano molteplici, il metodo offre la garanzia della transazione atomica per cui se il metodo ritorna in maniera corretta si ha la garanzia che tutti gli attributi sono stati resi persistenti.

Come esempio viene riportato un frammento di codice che utilizza le `PortletPreferences` appena descritte:

```
PortletPreferences prefs = request.getPreferences();
String[] symbols = prefs.getValues(”preferredStockSymbols”,
  new String[]{”ACME”,”FOO”});
String url = prefs.getValue(”quotesFeedURL”,null);
int refreshInterval = Integer.parseInt
  (prefs.getValue(”refresh”,”10”));
```

Il metodo reset come dice il nome provvederà a resettare gli attributi al valore di default associato, se quest'ultimo non è stato definito l'attributo verrà cancellato.

Di seguito viene riportato un frammento del file `portlet.xml` nel quale vengono specificati gli attributi delle `PortletPreferences` con i rispettivi valori.

```
<portlet>
     <portlet-preferences>
```

```
        <preference>
            <name>PreferredStockSymbols</name>
            <value>FOO</value>
            <value>XYZ</value>
            <read-only>true</read-only>
        </preference>
        <preference>
            <name>quotesFeedURL</name>
            <value>http://www.foomarket.com/quotes</value>
        </preference>
    </portlet-preferences>
</portlet>
```

Negli attributi in cui non è specificato il tag `<read-only>` viene assunto come default false (ovvero l'attributo può essere soggetto a modifiche rispetto al valore di default).
In ogni caso le Portlet possono utilizzare anche degli attributi che non sono stati definiti nel deployment descriptor.

2.1.7 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`.

2.1.8 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 JSR-168 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* (vedi bibliografia **[12]**).
La differenza tra i due metodi è il tipo di URL creato che può essere:

- `ActionURL`
- `RenderURL`

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, inoltre il `renderUrl` a differenza dell'`action URL` non passa attraverso il metodo `processAction`.
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 *PortletUrl*:

```
PortletURL url = response.createRenderURL();
url.setParameter( “user”, “foo.com” );
url.setParameter(“show”, “summary” );
writer.print( <A HREF=\ +url.toString()+ >Summary</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.

2.1.9 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 (A*ctionRequest*).
Quando il contenitore invoca il metodo `render` si dice che si è inoltrata una richiesta di renderizzazione (R*enderRequest*). Normalmente gli utenti comunicano attraverso degli URL appositi creati dalla Portlet, questi vengono chiamati *PortletURL* e si dividono in *ActionURL* e *RenderURL* che vengono tradotti se selezionati negli equivalenti messaggi di *ActionRequest* e di *RenderRequest*. 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.
Viene riportato di seguito la sequenza di gestione delle richieste nel caso di tre Portlet, in cui una richiede l'esecuzione del `processAction`.

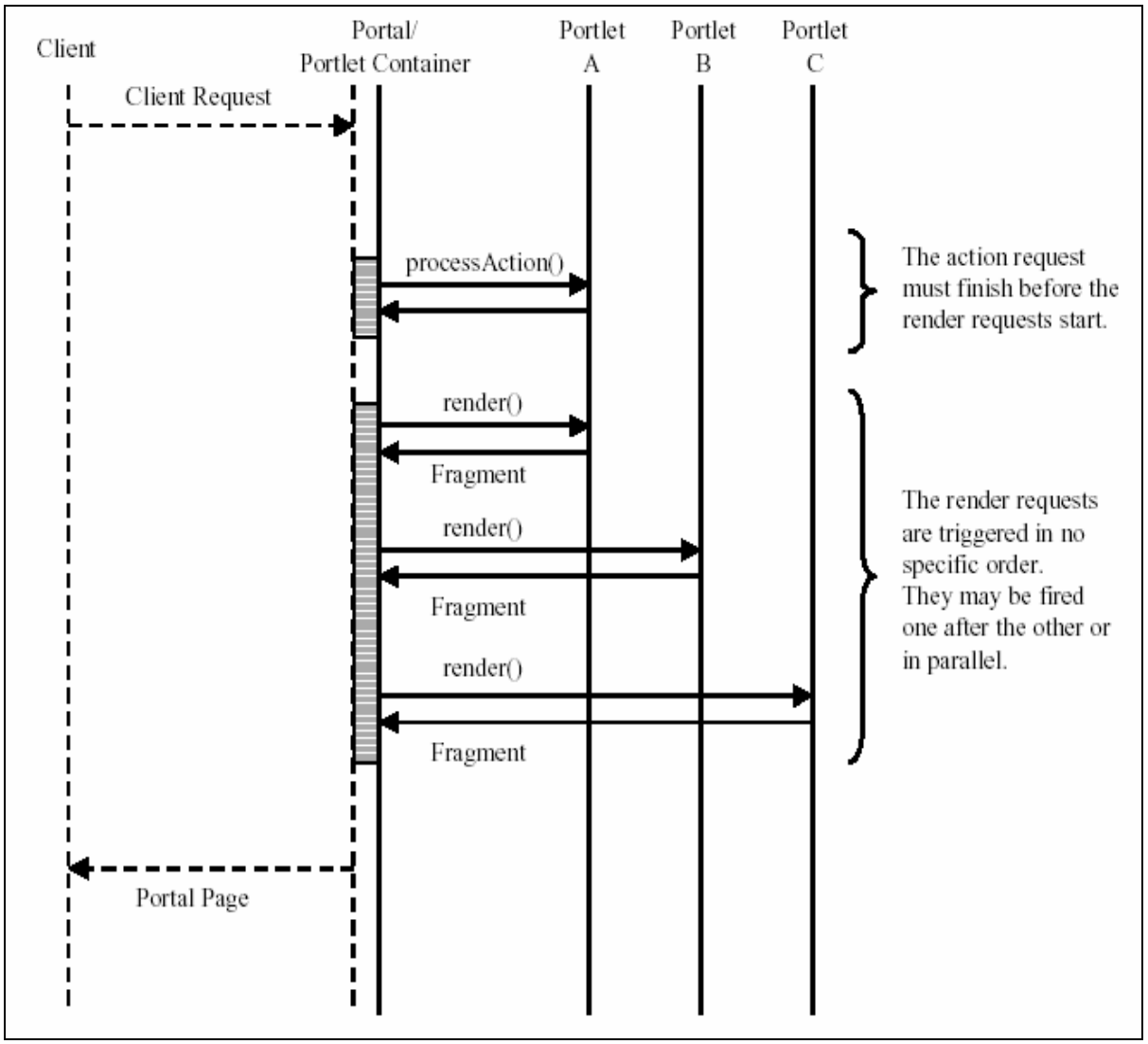

Figura 4: Sequenza di gestione delle richieste

2.1.10 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) quali nome, età, indirizzo, numero di telefono dell'utente autenticato. 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 del *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-app>
     <user-attribute>
         <description>User Given Name</description>
         <name>user.name.given</name>
     </user-attribute>
     <user-attribute>
         <description>User Last Name</description>
         <name>user.name.family</name>
     </user-attribute>
     <user-attribute>
         <description>User eMail</description>
         <name>user.home-info.online.email</name>
     </user-attribute>
     <user-attribute>
         <description>Company Organization</description>
         <name>user.business-info.postal.organization</name>
     </user-attribute>
 <portlet-app>
```

2.1.11 Portlet e Caching

Al fine di minimizzare il tempo di risposta del Portale verso gli utenti e 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 deployment descriptor

```
<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.
In realtà è possibile definire il tempo di validità in cache in modo programmatico settando la proprietà nell'oggetto `renderResponse` utilizzando la costante `EXPIRATION_CACHE` definita nell'interfaccia *PortletResponse*. Se la proprietà di expiration è settata a 0 il catching viene disabilitato mentre se viene settato a -1 le informazioni (se presenti) vengono prese dalla cache ritenendole sempre valide.
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 di risposta e diminuendo il carico.
Se durante l'invocazione del render la proprietà di expiration cache non è settata, verrà utilizzato il tempo di expiration definito nel deployment descriptor, se anche quest'ultimo non è presente l'expiration della cache il caching viene disabilitato.

2.1.12 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 `initparam` definiscono il nome la classe di partenza ed i parametri iniziali per la nostra Portlet (vedi bibliografia **[13]**).

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 descriptor della Portlet: `/WEB-INF/portlet.xml`
- Le classi della Portlet: `/WEB-INF/classes`
- Le eventuali Java Server Pages in `/WEB-INF/pages/*.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 và 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 difficoltoso.

2.1.13 Resource Bundles

Per permettere di utilizzare varie lingue nelle informazioni delle Portlet come il titolo, le keywords può essere utilizzato il *resource bundles*. Il nome della classe del resource bundle può essere settata nella definizione della Portlet nel deployment descriptor utilizzando il tag `resource-bundle`. La specifica delle Portlet definisce le seguenti costanti:

- `javax.portlet.title`: è il titolo che verrà visualizzato nella titlebar della Portlet.
- `javax.portlet.short-title`: è una versione ristretta del titolo che può essere usata da dispositivi con limitate capacità di visualizzazione
- `javax.portlet.keywords`: le keywords descrivono le funzionalità della Portlet. I portali che permettono agli utenti di cercare le Portlet in base a keywords possono utilizzare tale attributo. Possono essere specificate più keywords, basterà separarle da un virgola.

Di seguito vengono mostrati due resource bundles di esempio il primo in Inglese ed il secondo in Italiano.

```
# English Resource Bundle
#
# filename: clock_en.properties
# Portlet Info resource bundle example
     javax.portlet.title=World Population Clock
     javax.portlet.short-title=WorldPopClock
     javax.portlet.keywords=World,Population,Clock

# Italian Resource Bundle
```

```
#
# filename: clock_it.properties
# Portlet Info resource bundle example
     javax.portlet.title=Orario Popolazione Mondiale
     javax.portlet.short-title=OraPopMond
     javax.portlet.keywords=Mondiale,Popolazione,Orario
```

2.1.14 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 *PortletUrl* all'interno delle JSP.
L'utilizzo di questi tag è molto semplice,basta inserire all'interno della pagina JSP un header dichiarante l'utilizzo della tag-library che ne specifica l'utilizzo:

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

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 basterà inserire 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 *PortletURL* 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:renderURL portletMode= “view” windowState= “normal” >
     <portlet:param name= “showUser”  value= “username”/>
</portlet:renderURL>
```

Questo esempio crea un collegamento specificando la visualizzazione dell'utente “username” 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 un *actionURL* o ad un *renderURL*, ad esempio:

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

2.1.15 Portlet e sicurezza

In un portale, soprattutto se effettua e-commerce, la sicurezza ha un ruolo decisamente importante: il nuovo standard JSR168 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 autenticato 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 autenticato 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`, come da 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 cifrata.

```
<portlet-app>
     ...
     <portlet>
         <portlet-name>accountSummary</portlet-name>
         ...
     </portlet>
     ...
     <security-constraint>
          <display-name>Secure Portlets</display-name>
          <portlet-collection>
               <portlet-name>accountSummary</portlet-name>
          </portlet-collection>
          <user-data-constraint>
               <transport-guarantee>
                         CONFIDENTIAL
               </transport-guarantee>
          </user-data-constraint>
     </security-constraint>
     ...
</portlet-app>
```

2.1.16 CSS Style Definitions

Per permettere un look comune delle Portlet nei vai portali, tutte le Portlet per la generazione dei contenuti devono utilizzare un CSS comune

Per quanto riguarda i tag <a> non è stata definita nessuna classe particolare per cui le entità potranno utilizzare la classe di default.

Font

Per quanto riguarda la definizione del carattere, la definizione interesserà solamente gli attributi ovvero: font face, size, color, style, ecc…

Style	Descrizione	Esempio
portlet-font	Utilizzato per la visualizzazione di informazioni "normali", non di particolare rilievo	Testo normale
portlet-font-dim	Simile al .portlet-font ma il colore dei caratteri è acceso	Dim Text

Se uno sviluppatore di Portlet vuole specificare la dimensione dei caratteri può farlo utilizzando la grandezza relativa.
Ad esempio:

```
<div class="portlet-font" style="font-size:larger">
     Important information </div>
<div class="portlet-font-dim" style="font-size:80%">
     Small and dim </div>
```

Messages
Lo stile dei messaggi coinvolge il rendering del paragrafo (alignment, borders, background color, etc) come succedeva per gli attribute del testo.

Style	Descrizione	Esempio
portlet-msg-status	Stato dell'operazione corrente	*Progress: 80%*
portlet-msg-info	Messaggio di aiuto, informazioni generali, ecc…	Info about
portlet-msg-error	Messaggi di errore	Portlet not available
portlet-msg-alert	Messaggi di attenzione	*Timeout occurred, try again later*
portlet-msg-success	Successo dell'operazione eseguita	**Operation completed successfully**

Sections

Lo stile della sezione interessa il rendering del markup come le tabelle, i div e lo span (alignment, borders, background color, etc) come gli attributi del testo.

Style	Descrizione
portlet-section- header	Stile da applicare all'intestazione della tabella
portlet-section-body	Testo normale nella cella di una tabella
portlet-section-alternate	Testo in ogni altra riga nella cella
portlet-section-selected	Testo nella cella selezionata
portlet-section-subheader	Testo sotto l'intestazione della tabella
portlet-section-footer	Note della tabella
portlet-section-text	Testo nella tabella che non rientra in nessuna delle altre categorie

Forms

Lo stile dei form definisce il look-and-feel degli elementi in un form HTML.

Style	Description
portlet-form-label	Testo utilizzato per le label descrittive
portlet-form-input-field	Testo per gli user-input
portlet-form-button	Testo nei bottoni
portlet-icon-label	Testo che appare accanto ad una icona
portlet-dlg-icon-label	Testo che appare accanto ad una icona standard (Ok, Cancel,…)
portlet-form-field-label	Testo utilizzato per separatore di fields (es: checkboxes)
portlet-form-field	Testo per field

Menus

Lo stile dei menù, definisce il look-and-feel del testo e del background della struttura dei menu. Questa struttura può essere integrata in pagine aggregate o apparire in un popup menù.

Style	Descrizione
portlet-menu	Impostazioni generali del menu come il background, il colore, I margini, ecc…

portlet-menu-item	Oggetto non selezionato del menù
portlet-menu-item-selected	Oggetto selezionato del menù
portlet-menu-item-hover	Oggetto non selezionato del menu quando il mouse ci passa sopra
portlet-menu-item-hover-selected	Oggetto del menu selezionato quando il mouse ci va sopra
portlet-menu-cascade- item	Oggetto del menu non selezionato che ha sotto menu
portlet-menu-cascade-item-selected	Oggetto di un sotto menu selezionato che ha sotto menù
portlet-menu-description	Testo descrittivo per il menu (es: help)
portlet-menu-caption	Menu caption

2.1.17 Window state

La specifica che riguarda le Portlet definisce tre differenti window state:

- `Normal`
- `Maximized`
- `Minimized`

Normal window state

Questa modalità di visualizzazione indica che la Portlet condividerà l'intera pagina in cui è deployata con le altre Portlet. Questa modalità può essere indicata soprattutto per i dispositivi con limitate capacità grafiche.

Maximized window state

Questa modalità di visualizzazione indica che la Portlet sarà l'unica ad essere visualizzata nella pagina in cui è stata deployata. Questa modalità può essere indicata nei casi in cui la Portlet generi contenuti molto ricchi.

Minimized window state

In questa modalità viene visualizzato solamente la barra del titolo.

Custom window state

I portal vendors possono definire dei custom window state, le Portlet potranno utilizzare solamente gli stati della finestra che sono stati definiti nel portale.

D'altro canto le Portlet potranno definire nel deployment descriptor attraverso il tag `custom-windows-state` dei custom windos state che intendono utilizzare.
A tempo di deploy, i custom window state definiti nel deployment descriptor verranno mappati nei custom window state supportati dall'implementazione del portale.

2.1.18 Sessione

Il container deve assicurare che ciascuna richiesta alle Portlet generata come richieste dal portale in seguito ad una richiesta di un client , abbiano tutte la stessa sessione. Se con queste richieste alle Portlet , più di una Portlet creasse una sessione, l'oggetto sessione deve essere lo stesso per tutte le Portlet della stessa Portlet application. La visibilità della sessione è a livello di contesto della Portlet application. Naturalmente poiché una Portlet è usata da più utenti in più modi, ciascuna Portlet ha il suo unico oggetto `PortletSession` per identificare la sessione di ciascun utente, sarà compito del Portlet Container garantire che l'oggetto `PortletSession` non venga condiviso tra Portlet application o tra diversi utenti.
Vi sono due tipologie di visibilità degli attributi in sessione:

- `APPLICATION_SCOPE`
- `PORTLET_SCOPE`

Per quanto riguarda l'`APPLICATION_SCOPE` gli attributi saranno visibili da tutte le Portlet facenti parte della stessa Portlet application.
Diverso è il discorso del `PORTLET_SCOPE` poichè gli attributi saranno visibili solamente a livello di singola instanza si Portlet e non verranno quindi condivisi con nessun altra applicazione.
Per conoscere la visibilità di una variabile nella sessione http (usata dalla *PortletSession*) va utilizzata la *PortletSessionUtil*.
Essendo una Portlet application una estensione di una Web-Application , essa può contenere Servlet e Jsp oltre alle Portlet.
La *PortletSession* immagazzina gli attributi nella sessione HTTP perciò i dati messi nella http-session dalle Servlet e dalle Jsp sono accessibili alle Portlet attraverso la *PortletSession*, e viceversa, quelli messi dalle Portlet nella *PortletSession* sono accessibili alle la Servlet e Jsp attraverso *HttpSession*.

Se la sessione http deve essere invalidata, il container invalida anche la *PortletSession* associata e viceversa.
Per settare un oggetto in sessione verrà utilizzato il metodo `setAttribute` appartenente all'interfaccia *PortletSession*. Viene riportato per chiarezza un esempio:

```
PortletSession session = request.getSession(true);
PortletURL url = new URL("http://www.foo.com");
     session.setAttribute("home.url",url,
          PortletSession.APPLICATION_SCOPE);
     session.setAttribute("bkg.color","RED",
          PortletSession.PORTLET_SCOPE);
```

Per ritrovare un oggetto in sessione bisognerà utilizzare il metodo `getAttribute` appartenente all'interfaccia *PortletSession*, mentre per l'eliminazione verrà utilizzato il metodo `removeAttribute`.

2.2 Standard WSRP

2.2.1 Introduzione

Il consorzio OASIS (Organization for the Advancement of Structured Information Standards) ha approvato lo standard Web Services for Remote Portlets (WSRP) 1.0.
Si tratta di una vera e propria rivoluzione per la gestione dei contenuti nei portali e nei siti a grande produzione di contenuti: forma, contenuti e piattaforma di un sito sono realmente separati e gestibili, permettendo così di curare separatamente le parti. L'obiettivo è ottenere l'interscambio dei contenuti tra portali i quali, ignorando le differenze di piattaforma e di programmazione, potranno importare ed esportare notizie, articoli, immagini e quant'altro da e verso qualunque sito compatibile affiliato.
Grazie al WSRP (Web Services for Remote Portlet) sarà incrementata l'interazione e lo scambio tra server e linguaggi di diversa natura, il tutto tramite l'interfacciamento con il linguaggio XML.
Lo standard WSRP è stato progettato per aggregare i contenuti dei portali in modo standard. I portali che implementano questo protocollo potranno accedere, visualizzare e utilizzare risorse (come le Portlet) che risiedono su Portali remoti.
Questa specifica definisce delle interfacce per l'interazione tra web services di tipo presentation-oriented.
Contrariamente a quello che si potrebbe pensare, la specifica JSR-168 e lo standard WSRP non sono in competizione poiché riguardano aspetti differenti delle funzionalità messe a disposizione dalle Portlet.
La specifica JSR-168 è una tecnologia specifica Java che mette a disposizione delle Portlet API, progettate per garantire interoperabilità tra le Java Portlet e i Java Portlet Containers. Queste Portlet sono locali al container il quale, provvederà alla loro gestione.
Il WSRP invece è un protocollo che permette di accedere a Portlet remote in modo standard. Ecco che allora questi due standard non sono in competizione ma piuttosto complementari (vedi bibliografia **[18]**).
Il WSRP mette a disposizione una piattaforma base per avere interoperabilità nella pubblicazione e consumo delle Portlet remote. In particolare definirà un protocollo comune e un insieme di interfacce per i web services presentation-oriented.

Lo scenario sarà composto da:

- Portal server (Producer) che mettono a disposizione Portlet in modalità remota accessibili mediante lo standard WSRP
- Portal server (Consumer) che consumeranno Portlet remote e aggregheranno tali contenuti nel proprio portale.

L'aspetto da tenere in considerazione è che Producer e Consumer potranno essere implementati in piattaforme differenti siano queste J2EE o .Net .

Questo particolare tipo di web services è costruito avvalendosi di tecnologie standard come:

- SSL / TLS
- URI / URL
- WSDL
- SOAP

Oltre a questi in futuro verranno anche utilizzati altri standard riguardanti i web services quali WS-Security e WS-Policy.

Tradizionalmente i web services sono data-oriented, quindi richiedono l'aggregazione delle applicazioni per fornire il livello di presentation logic. In più ogni applicazione di aggregazione comunica con il web services tramite un'interfaccia unica. Questo approccio non è adatto all'integrazione dinamica tipica dei portali.

La specifica WSRP, risolve questo problema introducendo delle web services interfaces di tipo presentation-oriented che permettono l'inclusione e l'interazione di contenuti tra web services (vedi bibliografia **[15]**).

Un web services di questo tipo fornirà sia la logica di applicazione che la logica di presentazione.

Nei paragrafi seguenti verrà illustrata l'architettura SOA con particolare riferimento ai Web Services. Conseguentemente verranno presentati gli standard maggiormente utilizzati nel WSRP come WSDL, SOAP e UDDI (vedi bibliografia **[17]**).

Infine verranno illustrate le peculiarità proprie dello standard WSRP.

2.2.2 SOA

SOA (Service Oriented Architecture) è uno stile architetturale basato sul concetto di Servizio, che rappresenta quindi l'elemento strutturale su cui le applicazioni vengono sviluppate.
Un servizio è, in prima analisi, una funzionalità di business realizzata tramite un componente che rispetta un'interfaccia.
In passato, sono stati presentati spesso modelli di sviluppo incentrati su componenti con l'obiettivo del riuso. La finalità di queste tecnologie è stata di cercare di creare un mercato di componenti per favorirne la diffusione. Questo tipo di politica ha in parte fallito soprattutto perchè i modelli di programmazione proposti hanno portato alla definizione di componenti la cui possibilità di utilizzo era limitata ad un certo ambito tecnologico.
Inoltre, mentre oggi è chiaro che la necessità di sistemi informativi complessi è quella di avere componenti in grado di comunicare tra loro in maniera eterogenea e disaccoppiata, le tecnologie orientate al "mercato dei componenti" hanno spesso dato più importanza alla possibilità di riconfigurare i moduli sviluppati per essere utilizzati su applicazioni diverse piuttosto che a sottolineare le possibilità di integrazione.
SOA parte dalle considerazioni che hanno portato al fallimento del riuso dei componenti in ambito enterprise e definisce una serie di proprietà che i servizi devono soddisfare per essere realmente riusabili e facilmente integrabili in ambiente eterogeneo:

- I Servizi devono essere ricercabili e recuperabili dinamicamente. Chi necessita di un servizio deve essere in grado di ricercarlo sulla base dell'interfaccia e di chiamarlo a tempo di esecuzione. Questo tipo di meccanismo permette un forte disaccoppiamento tra chi richiede la funzionalità e chi la fornisce, permettendo inoltre di cambiare l'entità che esegue il servizio a tempo di esecuzione in maniera trasparente rispetto al chiamante.
- I Servizi devono essere autocontenuti e modulari.
  Per essere realmente riusabili, è importante che i servizi non siano legati al contesto o allo stato di altri servizi. Ovviamente le applicazioni richiedenti necessitano di avere stato persistente tra le invocazioni ma questo deve essere separato dal fornitore di servizio. In pratica non dovrebbe esserci uno stato conversazionale nelle chiamate tra chi richiede il servizio e chi lo fornisce, i servizi dovrebbero quindi essere stateless.

- I Servizi devono definire delle interfacce esplicite e indipendenti dall'implementazione

  Deve essere possibile invocare servizi in maniera indipendente dal linguaggio e dalla piattaforma. Questo si può ottenere definendo delle interfacce che possono essere invocate tramite protocolli generalmente ben supportati.
- I Servizi devono essere debolmente accoppiati (loosely coupled). L'accoppiamento si riferisce al numero di dipendenze tra i moduli. Ogni tipo di architettura ben definita è orientata ad avere accoppiamento debole, cioè un numero di dipendenze tra le entità basso e ben controllato. Un sistema formato da componenti fortemente accoppiati è più rigido e difficilmente modificabile.
- I Servizi devono avere un'interfaccia distribuita e devono essere accessibili in maniera trasparente rispetto all'allocazione.

  Un servizio con un'interfaccia distribuita può essere pubblicato sulla rete, diventando così disponibile per chi intenderà utilizzarlo. L'accesso tramite la rete permette inoltre di avere trasparenza rispetto alla reale allocazione del servizio.
- I Servizi devono avere preferibilmente un'interfaccia a "grana grossa" (coarse-grained).

  Un servizio che corrisponda ad un'unica chiamata e un'unica esecuzione complessa ha in genere dei vantaggi rispetto a una serie di chiamate a tanti servizi più piccoli. In questo modo infatti vengono fatte meno chiamate remote (tipicamente poco efficienti), non c'è bisogno di trovare un sistema per mantenere lo stato tra più chiamate ed è più semplice gestire problematiche legate al fallimento della comunicazione remota. Comunque può essere sensato definire servizi "sempici" soprattutto se questi servizi possono essere utilizzati per essere composti in altri servizi.
- I Servizi devono essere componibili.

  Dal punto di vista SOA le applicazioni sono aggregazione di servizi. È quindi importante disegnarne le interfacce in modo che corrispondano a funzioni di business riusabili, ovvero a servizi indipendenti e autocontenuti. La composizione di servizi per la produzione di applicazioni o di servizi più complessi viene indicata con il termine Service Orchestration.

Una architettura SOA è quindi una architettura software che definisce un modo di descrivere i componenti con caratteristiche ben specifiche orientate al riutilizzo e

all'integrazione. È importante notare che a livello implementativo la tecnologia utilizzata per lo sviluppo dei servizi non è determinante finché vengono rispettate le caratteristiche appena esposte.
Come già detto un servizio deve definire un'interfaccia pubblicabile sulla rete, ricercabile e invocabile in maniera indipendente dal linguaggio e dalla piattaforma. Per ottenere questi requisiti, le applicazioni SOA definiscono dei ruoli:

- *Service Requester*: l'entità che richiede il servizio; può essere un modulo di un'applicazione o un altro servizio.
- *Service Provider*: l'entità che fornisce il servizio e che ne espone l'interfaccia.
- *Service Contract*: definisce il formato per la richiesta di un servizio e della relativa risposta.
- *Service Broker*: Direttorio in rete dei servizi consultabili.

Poiché i servizi devono essere ricercati e recuperati dinamicamente, il Service Contract deve essere pubblicato su un Service Broker dal Service Provider. Il Service Requester deve richiedere al Service Broker il Contract relativo al servizio richiesto, che utilizzerà per eseguire il servizio tramite un protocollo di trasporto.

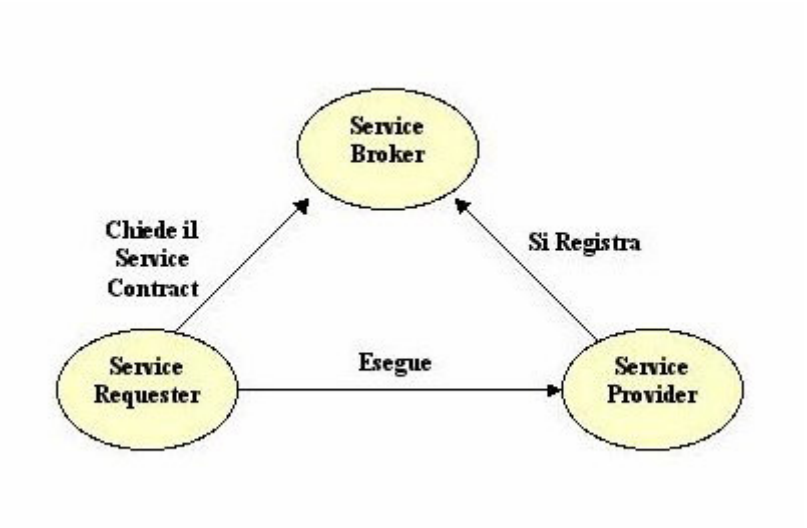

Figura 5: Interazione SOA

Sia il requester che il provider devono utilizzare un protocollo di comunicazione che sia comprensibile per entrambi. Questo protocollo deve essere deciso sulla base dell'ambiente di utilizzo e dei requisiti in termini di efficienza e robustezza. Ad

esempio nel caso di integrazione spesso la soluzione più interessante è utilizzare protocolli di messaging asincrono.
Nulla vieta inoltre che un servizio possa supportare la possibilità di essere invocato tramite più protocolli (Multiprotocol Service Invocation). In linea di principio, un servizio potrebbe essere definito una sola volta tramite l'interfaccia e avere molte implementazioni che supportano diversi protocolli di accesso.
I concetti definiti da SOA non sono legati ad una tecnologia particolare, tuttavia i Web Services come si potrà vedere dal paragrafo seguente rappresentano un modo di utilizzazione di tali concetti per l'esposizione di servizi.

2.2.3 Web Service

Prima di entrare nel merito dello standard WSRP, in questo paragrafo verranno introdotti i concetti che stanno alla base dei Web services.
Secondo la definizione data dal W3C un Web service (servizio web) è un sistema software progettato per supportare l'interoperabilità tra diversi elaboratori su di una rete; caratteristica fondamentale di un Web Service è quella di offrire un'interfaccia software (descritta in un formato automaticamente elaborabile quale, ad esempio, il WSDL) utilizzando la quale altri sistemi possono interagire con il Web Service stesso attivando le operazioni descritte nell'interfaccia tramite appositi "messaggi" inclusi in una "busta" SOAP: tali messaggi sono, solitamente, trasportati tramite il protocollo HTTP e formattati secondo lo standard XML.
Tramite un'architettura basata sui Web Service (chiamata, con terminologia inglese, *Service Oriented Architecture - SOA*) applicazioni software scritte in diversi linguaggi di programmazione e implementate su diverse piattaforme hardware possono quindi essere utilizzate, tramite le interfacce che queste "espongono" pubblicamente e mediante l'utilizzo delle funzioni che sono in grado di effettuare (i "servizi" che mettono a disposizione) per lo scambio di informazioni e l'effettuazione di operazioni complesse (quali, ad esempio, la realizzazione di *processi di business* che coinvolgono più aree di una medesima azienda) sia su reti aziendali come anche su Internet: la possibilità dell'interoperabilità fra diversi software (ad esempio, tra Java e Python) e diverse piattaforme hardware (come Windows e Linux) è resa possibile dall'uso di standard "open source".

Il consorzio OASIS (*Organization for the Advancement of Structured Information Standards*) ed il World Wide Web Consortium sono i principali responsabili dell'architettura e della standardizzazione dei Web Service; per migliorare l'interoperabilità tra le diverse implementazioni dei Web Service l'organizzazione WS-I sta inoltre sviluppando una serie di "profili" per meglio definire gli standard coinvolti.

La ragione principale per la creazione e l'utilizzo di Web Service è il "disaccoppiamento" che l'interfaccia standard esposta dal Web Service rende possibile fra il sistema utente ed il Web Service stesso: modifiche ad una o all'altra delle applicazioni possono essere attuate in maniera "trasparente" all'interfaccia tra i due sistemi; tale flessibilità consente la creazione di sistemi software complessi costituiti da componenti svincolati l'uno dall'altro e consente una forte riusabilità di codice ed applicazioni già sviluppate.

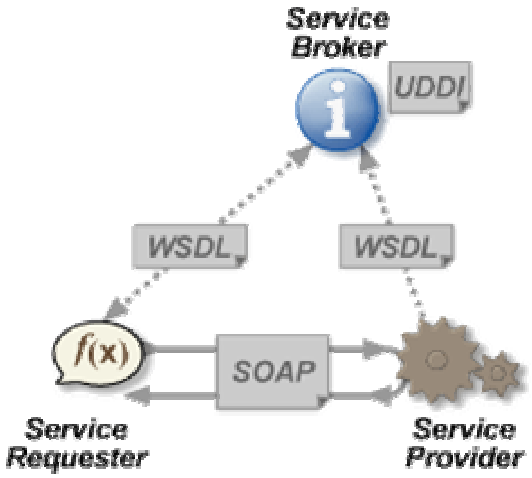

Figura 6: Interazione tra Web Services

La figura precedente rappresenta la pila protocollare dei Web Service che è l'insieme dei protocolli di rete utilizzati per definire, localizzare, realizzare e far interagire tra di loro i Web Services; è principalmente composta di quattro aree:

- Trasporto del servizio: responsabile per il trasporto dei messaggi tra le applicazioni in rete, include protocolli quali HTTP, SMTP, FTP, XMPP ed il recente *Blocks Extensible Exchange Protocol* (BEEP)
- XML Messaging: tutti i dati scambiati sono formattati mediante "tag" XML in modo che gli stessi possano essere utilizzati ad entrambi i capi della

connessioni; il messaggio può essere codificato conformemente allo standard SOAP, come utilizzare JAX-RPC, XML-RPC o REST

- Descrizione del servizio: l'interfaccia pubblica di un Web Service viene descritta tramite WSDL (*Web Services Description Language*) un linguaggio basato su XML usato per la creazione di "documenti" descrittivi delle modalità di interfacciamento ed utilizzo del Web Service
- Elencazione dei servizi: la centralizzazione della descrizione e della localizzazione dei Web Service in un "registro" comune permette la ricerca ed il reperimento in maniera veloce dei Web Service disponibili in rete; a tale scopo viene attualmente utilizzato il protocollo UDDI.

Ulteriori protocolli standard utilizzati sono:

WS-Security: il protocollo *Web Services Security protocol* è stato adottato come standard OASIS; tale standard permette l'autenticazione degli utenti e la confidenzialità dei messaggi scambiati con l'interfaccia del Web Service

WS-Reliability: si tratta di specifiche basate su SOAP ed accettate come standard OASIS che soddisfano la richiesta di messaggi "affidabili" (*reliable*), richiesta critica per alcune delle applicazioni che utilizzano i Web Service (come, ad esempio, transazioni monetarie o applicazioni di E-commerce).

I vantaggi introdotti dall'utlizzo dei Web Services, si possono riassumere nei seguenti punti:

- permettono l'interoperabilità tra diverse applicazioni software su diverse piattaforme hardware
- utilizzano standard e protocolli "open"; i protocolli ed i formato dei dati è, ove possibile, in formato testuale, cosa li che rende di più facile comprensione ed utilizzo da parte degli sviluppatori
- mediante l'uso di HTTP per il trasporto dei messaggi i Web Service non necessitano, normalmente, che vengano effettuate modifiche alle regole di sicurezza utilizzate come filtro sui firewall
- possono essere facilmente utilizzati, in combinazione l'uno con l'altro (indipendentemente da chi li fornisce e da dove vengono resi disponibili) per formare servizi "integrati" e complessi.
- consentono il riutilizzo di infrastrutture ed applicazioni già sviluppate e sono (relativamente) indipendenti da eventuali modifiche delle stesse

Vi sono però anche degli "svantaggi", quali:

- attualmente non esistono standard consolidati per applicazioni critiche quali, ad esempio, le transazioni distribuite
- le performance legate all'utilizzo dei Web Service possono essere minori di quelle riscontrabili utilizzando approcci alternativi di distributed computing quali Java RMI, CORBA, o DCOM
- L'uso dell'HTTP, permette ai Web Service di evitare le misure di sicurezza dei firewall (le cui regole sono stabilite spesso proprio per evitare le comunicazioni fra programmi "esterni" ed "interni" al firewall).

2.2.4 WSDL

WSDL (Web Services Description Language) è un linguaggio formale in formato XML utilizzato per la creazione di "documenti" per la descrizione di Web Service.
Mediante WSDL può essere, infatti, descritta l'interfaccia pubblica di un Web Service ovvero creare una descrizione, basata su XML, di come interagire con un determinato servizio: un "documento" WSDL contiene infatti, relativamente al Web Service descritto, informazioni su:

- *cosa* può essere utilizzato (le "operazioni" messe a disposizione dal servizio);
- *come* utilizzarlo (il protocollo di comunicazione da utilizzare per accedere al servizio, il formato dei messaggi accettati in input e restituiti in output dal servizio ed i dati correlati) ovvero i "vincoli" (*bindings*) del servizio;
- *dove* utilizzare il servizio (cosidetto *endpoint* del servizio che solitamente corrisponde all'indirizzo - in formato URI - che rende disponibile il Web Service)

Le operazioni supportate dal Web Service ed i messaggi che è possibile scambiare con lo stesso sono descritti in modo astratto e quindi collegati ad uno specifico protocollo di rete e ad uno specifico formato.
Il WSDL è solitamente utilizzato in combinazione con SOAP e XML Schema per rendere disponibili Web Services su reti aziendali o su internet: un programma client può, infatti, "leggere" il documento WSDL relativo ad un Web Service per determinare quali siano le funzioni messe a disposizione sul server e quindi utilizzare il protocollo SOAP per utilizzare una o più delle funzioni elencate dal WSDL.

2.2.5 UDDI

L'UDDI (Universal Description Discovery and Integration) è un *registry* (ovvero una base dati ordinata ed indicizzata), basato su XML ed indipendente dalla piattaforma hardware, che permette alle aziende la pubblicazione dei propri servizi offerti su internet.
Questo standard è un'iniziativa "open" sponsorizzata dall'OASIS (consorzio internazionale per lo sviluppo e l'adozione di standard nel campo dell'e-business e dei Web Services).
Una "registrazione" UDDI consiste, infatti, di tre diverse componenti:

- Pagine bianche (*White Pages*): indirizzo, contatti (dell'azienda che offre uno o più servizi), ed identificativi;
- Pagine gialle (*Yellow Pages*): categorizzazione dei servizi basata su tassonomie standardizzate;
- Pagine verdi (*Green Pages*): informazioni (tecniche) dei servizi fornite dall'azienda

L'UDDI è uno degli standard alla base del funzionamento dei Web Service: è stato progettato per essere interrogato da messaggi in SOAP e per fornire il collegamento ai documenti WSDL che descrivono i vincoli protocollari ed i formati dei messaggi necessari per l'interazione con i Web Service elencati nella propria *directory*.

2.2.6 SOAP

SOAP (Simple Object Access Protocol) è un protocollo leggero per lo scambio di messaggi tra componenti software, tipicamente nella forma di componentistica software. La parola *object* manifesta che l'uso del protocollo dovrebbe effettuarsi secondo il paradigma della programmazione orientata agli oggetti.
SOAP è una struttura operativa (*framework*) estensibile e decentralizzata che può operare sopra vari protocol stack per reti di computer. I richiami di procedure remote possono essere modellati come interazione di parecchi messaggi SOAP.
Questo standard può muoversi sopra tutti i protocolli di Internet, ma l'HTTP è il più comunemente utilizzato e l'unico ad essere stato standardizzato dal W3C. SOAP si basa sul metalinguaggio XML e la sua struttura segue la configurazione Head-Body,

analogamente ad HTML. Il segmento opzionale Header contiene meta-informazioni come quelle che riguardano il routing, la sicurezza e le transazioni. Il segmento Body trasporta il contenuto informativo (*payload*). Questo deve seguire uno schema definito dal linguaggio XML Schema.
Come esempio, viene mostrato come un cliente può formattare un messaggio SOAP per richiedere informazioni su un prodotto da un immaginario warehouse web service.

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

Quello che segue è il testo con il quale il warehouse web service potrebbe inviare il suo messaggio di risposta con le informazioni richieste.

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

2.2.7 Attori

Lo standard WSRP descrive le modalità di interazione tra Producer e Consumer. Come già accennato in precedenza, il Producer è un web services presentation-oriented il quale ospita delle Portlet che saranno utilizzabili attraverso questo standard. Il Consumer interagirà con il Producer per presentare il markup all'end-user.
Per permettere di accedere alle Portlet in modo remoto, il Producer metterà a disposizione un insieme di web services interfaces, quali:

- Service Description (richiesta)
- Markup (richiesta)
- Registration (opzionale)
- Portlet Management (opzionale)

Ne verranno spiegate le peculiarità nel paragrafo seguente.
Il Consumer è un sistema che si interpone tra il Producer e l'end-user. Il suo compito sarà quello di aggregare il markup prodotto dalle Portlet e presentarlo all'utente finale. In più dovrà anche gestire l'interazione che l'utente avrà con le Portlet remote. Per questo ruolo di intermediario, il Consumer viene spesso indicato come "message-switches" ovvero come sistema che effettua il routing dei messaggi tra le due parti.
L'ultimo attore che interviene nello standard è l'end-user il quale deciderà (secondo i propri diritti) l'utilizzo e l'interazione con le Portlet (remote) tramite il proprio portale (Consumer) messe a disposizione attraverso questo standard.
Viene riportato di seguito una figura rappresentate gli attori che intervengono nello standard WSRP e locale così da poterne cogliere le differenze.

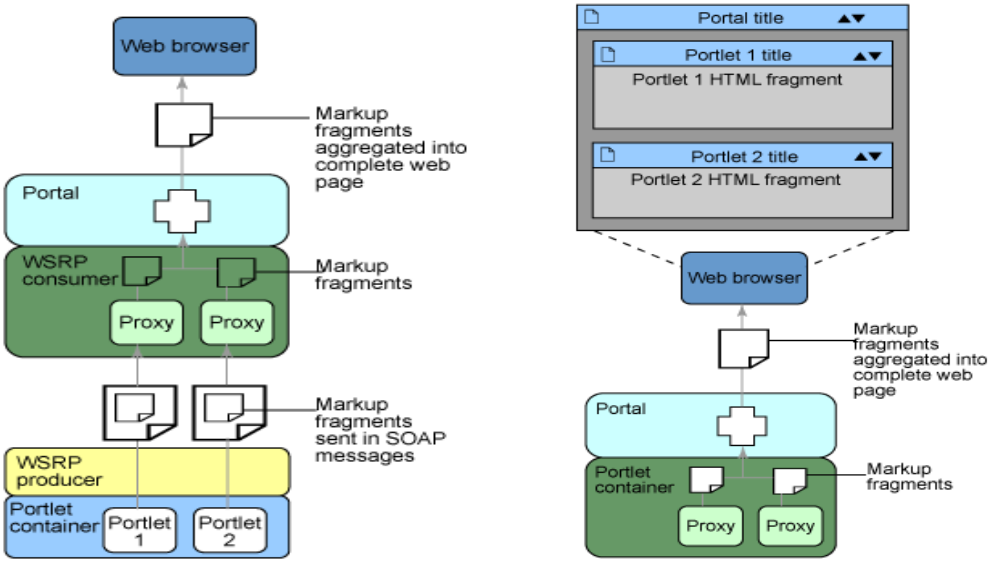

Figura 7: Differenze tra scenario WSRP e locale

Di seguito viene riportata una figura rappresentante l'interazione tra gli attori sopra descritti.

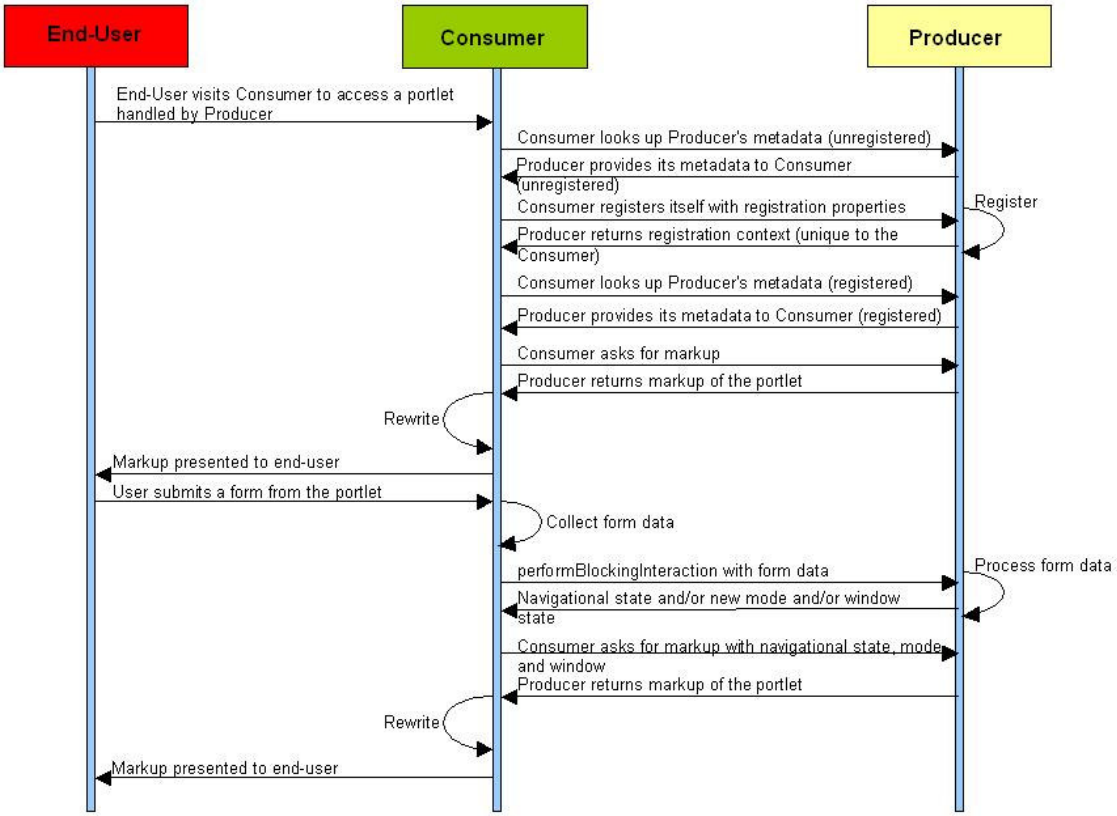

Figura 8: Interazione tra attori WSRP

Tipicamente il processo di interazione tra gli attori avviene tramite i seguenti step:

1. Il Consumer "scopre" il Producer, conoscendo gli Url associati alle interfacce standard che il Producer implementa per esporre i propri servizi si ha una prima interazione e tramite i metadata del Producer il Consumer viene a conoscenza delle Portlet che vengono messe a disposizione.
2. Creazione della relazione tra Consumer e Producer. Questo può comportare lo scambio di informazioni riguardo le capacità, le impostazioni di sicurezza e altri aspetti tecnici e/o di business.
3. Il Consumer viene a conoscenza di tutte le capacità e servizi del Producer basandosi sulla relazione stabilita.

4. Creazione della relazione tra Consumer ed end-user. Questo permette al Consumer di autenticare l'end-user, inoltre può permettergli di modificare le pagine che gli verranno presentate.
5. Produzione e aggregazione del markup per la composizione delle pagine da visualizzare da parte del Consumer.
6. Richiesta di una pagina. Questo capita solitamente quando l'end-user tramite un user-agent (es:browser) agisce su una Portlet remota.
7. Il Consumer passa la richiesta di interazione al Producer, il quale dopo le operazioni del caso invierà il markup di risposta. A questo punto si ritorna al punto 5.
8. Come ultima operazione si avrà la distruzione della relazione stabilita. Sia il Consumer che il Producer potranno in qualsiasi momento terminare la comunicazione.

2.2.8 Interfacce

In questo paragrafo verranno illustrate le peculiarità delle interfacce di cui ne è richiesta l'implementazione per poter fare uso dello standard WSRP.
Prendiamo in considerazione la `Service Description Interface`.
Questa interfaccia obbligatoria è richiesta per fornire i meta-data al Consumer così da poter interagire con ogni Portlet che il Producer ospita, inoltre fornisce delle informazioni aggiuntive sulle funzionalità offerte dal Producer.

Figura 9: Interazione tramite la Service Description Interface

Come si può notare dall'immagine riportata, il Consumer invierà tramite il metodo `getServiceDescription` una richiesta per ottenere la descrizione dei servizi messi

a disposizione dal Producer. Quest'ultimo risponderà con dei meta dati contenenti una serie di informazioni quali:

- Portlet offerte
- Tipi MIME supportati
- Modalità supportate
- Stati della finestra supportati

In questi meta dati potrà anche essere specificata la richiesta di registrazione da parte del Consumer.

La seconda interfaccia che prendiamo in considerazione è la `Markup Interface`: anche questa è obbligatoria; viene utilizzata per l'interazione con le richieste di utente e la generazione dei frammenti di markup.

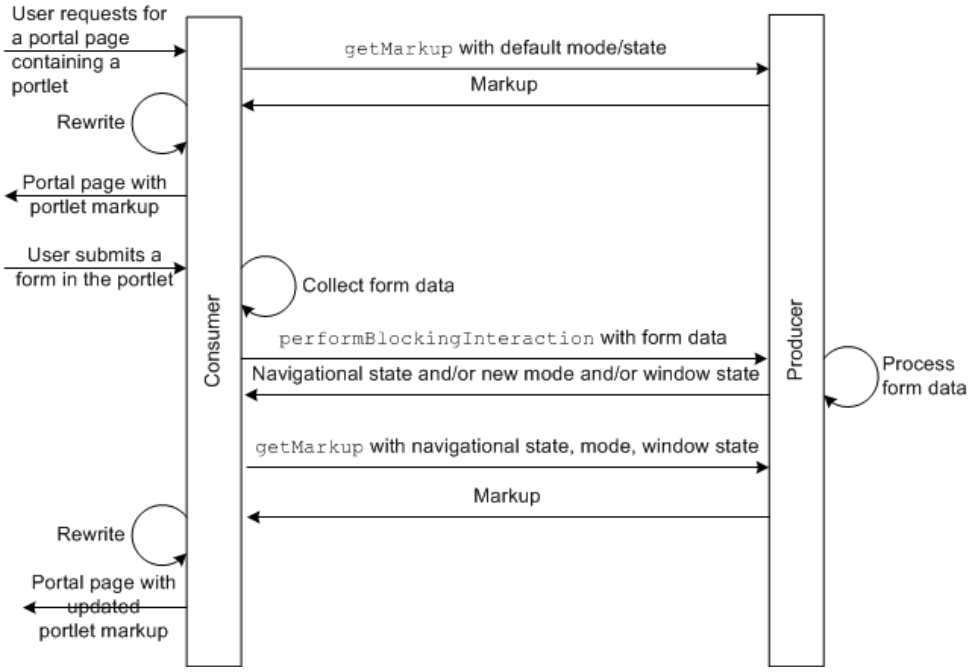

Figura 10: Interazione tramite la Markup Interface

Come si può notare quando l'end-user richiede una pagina del portale contenente Portlet remote il Consumer invierà la richiesta di markup al Producer che le ospita. Quest'ultimo risponderà con il markup richiesto che verrà aggregato dal Consumer per generare la pagina del portale e mostrarla all'end-user.

Prendiamo ora in considerazione la `Registration Interface`: un'interfaccia opzionale che definisce le operazioni per stabilire, aggiornare e cancellare la

registrazione del Consumer verso il Producer. Ogni registrazione riflette una particolare relazione tra queste due entità.

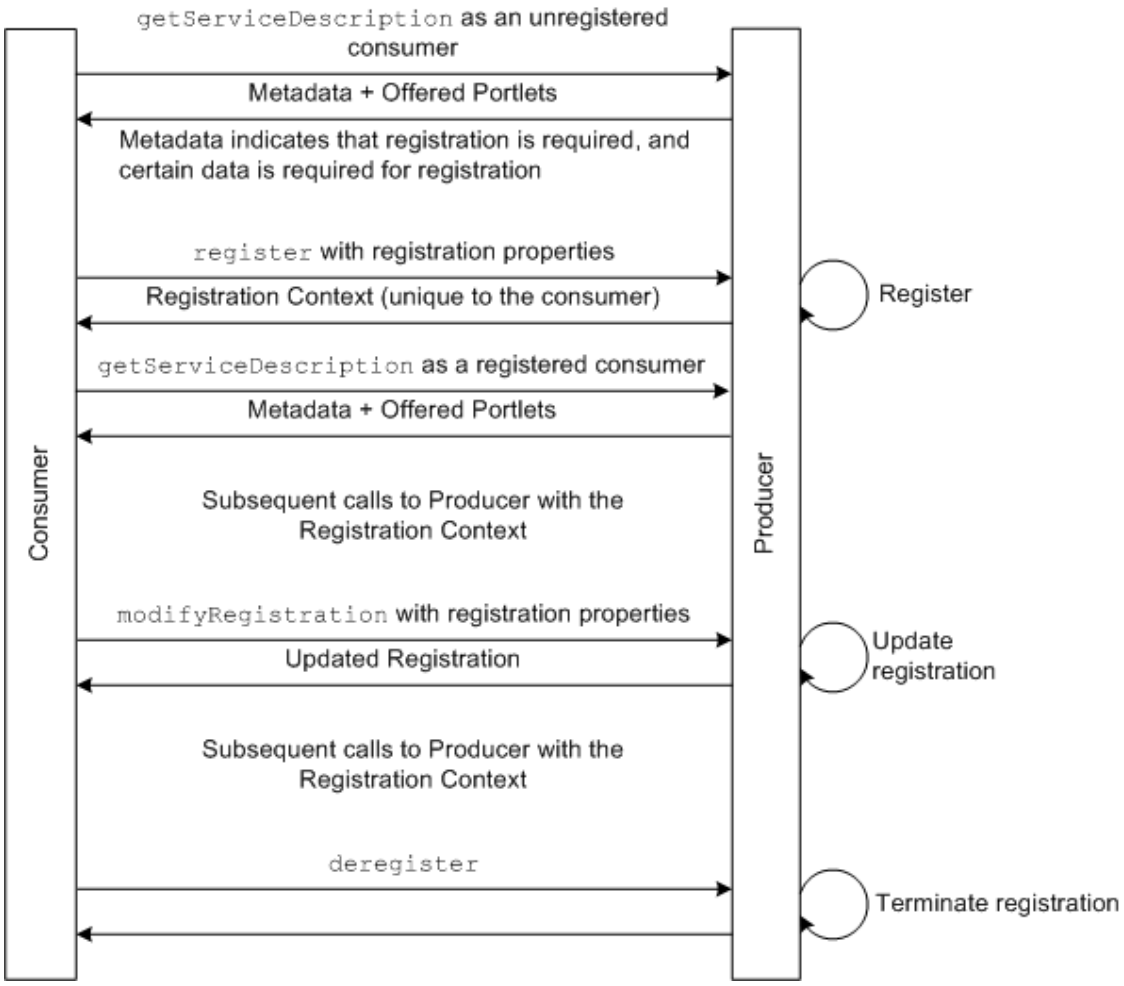

Figura 11: Interazione tramite la Registration Interface

Come si può notare dalla figura, nei meta dati inviati dal Producer è possibile richiedere la registrazione del Consumer. A questo punto il Consumer si registra ricevendo un `Registration Context` il quale è composto da:

- `RegistrationHandle:` è un identificatore univoco assegnato dal Producer al Consumer che rimane inalterato per tutta la durata dell'interazione tra le due entità.
- `RegistrationState:` è opzionale, se presente contiene informazioni associate alla registrazione da rendere persistenti per essere disponibili in futuro.

A questo punto la registrazione è stata effettuata, successivamente il Consumer potrà richiedere la modifica di alcune proprietà e alla fine dell'interazione verrà effettuata la terminazione della registrazione (tramite l'operazione di `deregister`).
Come ultima prendiamo in considerazione la `Portlet Management Interface`: un'interfaccia facoltativa usata per il controllo del life-cycle dello stato persistente delle Portlet

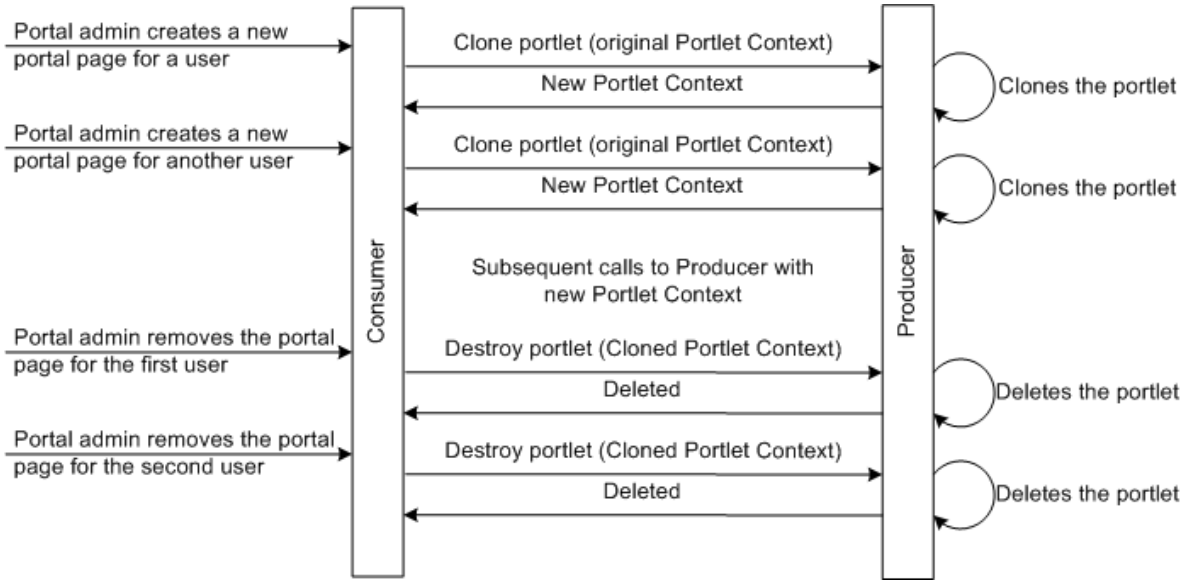

Figura 12: Interazione Portlet Management Interface

Come si può notare dalla figura riportata, quando l'amministratore (o un utente con i diritti per farlo) aggiunge una Portlet remota ad una pagina del portale Consumer, viene richiesta una clonazione della Portlet al Producer il quale invierà il nuovo contesto associato alla Portlet appena clonata. Quando tale Portlet verrà eliminata dalla pagina, verrà richiamato il metodo `destroy` che la cancellerà. Da notare che la Portlet che verrà eliminata è quella clonata non quella originale poiché se venisse cancellata quest'ultima, non potrebbe più essere utilizzata da alcun Consumer.

2.3 WSRP4J

2.3.1 Introduzione

L'integrazione di contenuti e applicazioni nei portali fino a questo momento hanno richiesto notevoli sforzi di programmazione. I venditori di portali o organizzazioni che li utilizzano, hanno dovuto scrivere speciali adapter Portlets per permettere ai portali di comunicare con applicazioni differenti e ai content provider per supportare una larga varietà di interfacce e protocolli.
Lo standard definito dall'OASIS ovvero i Web Services for Remote Portlets (WSRP) (vedi bibliografia **[14]**) semplifica l'integrazione di applicazioni remote e di contenuti dentro i portali. Gli amministratori dei portali potranno scegliere da una ricca lista di servizi decidendo quali integrare nel proprio portale senza eccessivi sforzi di programmazione.
Apache WSRP4J è un progetto open source che è stato iniziato dall'IBM per facilitare la veloce adozione dello standard WSRP dai content e application providers e dai portal vendors (vedi bibliografia **[16]**).
Essi potranno utilizzare WSRP4J come una piattaforma per lo sviluppo e l'hosting di WSRP compliant web services.
Nel momento di stesura della tesi, il progetto è in incubazione sotto la sponsorizzazione dell'Apache Software Foundation's (ASF). L'incubazione è richiesta per ogni progetto recentemente accettato, essa terminerà quando le indicazioni, le infrastrutture, le comunicazioni e le decisioni si saranno stabilizzate in modo consistente.
WSRP4J, non è un portal framework ma è l'implementazione di riferimento della specifica WSRP 1.0 la quale prevede l'esistenza di due soggetti distinti:

- Producer: il quale metterà a disposizione delle Portlet remote accessibili mediante lo standard WSRP.
- Consumer: il quale potrà utilizzare delle Portlet messe a disposizione da vari Producer.

In accordo con la specifica, il WSRP4J prevederà l'implementazione delle web services interfaces introdotte precedentemente.

Nel periodo di tirocinio, questo progetto è stato integrato in Pluto (vedi bibliografia **[1]**) così da poter essere utilizzato come portale Producer.

2.3.2 Installazione e configurazione

Il progetto wsrp4j è in una fase di incubazione, quindi in questo momento non sono state pubblicate versioni formali. Essendo un progetto Open Source è possibile ottenere i sorgenti utilizzando un Subversion Client digitando da linea di comando la seguente stringa:

```
svn co https://svn.apache.org/repos/asf/portals/wsrp4j/trunk
```

Requisiti per il funzionamento di WSRP4J:

- JDK 1.5.0 o maggiore
- Subversion Client
- Maven 1.0.2
- Tomcat 5.5 o come nel nostro caso Pluto (il quale per il proprio funzionamento utilizza Tomcat)

Il progetto WSRP4J si divide nelle seguenti componenti:

- producer
- consumer
- commons
- commons-producer
- commons-consumer
- consumer-swingconsumer
- testportlet (Portlet di esempio)

Per prima operazione andrà rinominato il file `build.properties.example` (presente nella directory in cui sono stati scaricati i sorgenti di WSRP4J) in `build.properties`. Successivamente andrà editato impostando la directory in cui è installato Tomcat e la relativa versione, viene riportata qua di seguito la configurazione del nostro caso:

```
maven.tomcat.home=/pluto/pluto-1.0.1-rc4
maven.tomcat.version.major=5.5
```

Come si nota, la Tomcat home è settata sotto Pluto, poiché come già accennato in precedenza Pluto utilizza Tomcat per il proprio funzionamento.
Un altro file di configurazione molto importante è `ConfigService.properties` reperibile nella directory: [WSRP4J-HomeDir]\producer\target\wsrp4j-producer\WEB-INF\config\services. In questo file è possibile definire:

- l'host name che intendiamo associare al WSRP4J Producer
- la porta su cui è configurato Tomcat per il traffico HTTP
- la porta su cui è configurato Tomcat per il traffico HTTPS
- i Portlet Mode supportati
- i Windows State supportati
- proprietà rilevanti del Portlet Container

Ne viene riportato un esempio:

```
host.name=WSRPServer
host.port.http=@HTTPPORT@
host.port.https=@HTTPSPORT@

supported.portletmode=view
supported.portletmode=edit
supported.portletmode=help
supported.portletmode=config

supported.windowstate=normal
supported.windowstate=maximized
supported.windowstate=minimized

portletcontainer.uniquename=wsrp4j-producer
portletcontainer.entrance.impl=org.apache.pluto.
PortletContainerImpl
  portletcontainer.entrance.wrapper.impl=org.apache.pluto.
  portalImpl.core.PortletContainerWrapperImpl
  portletcontainer.supportsBuffering=no
```

Lo standard WSRP per il proprio funzionamento richiede l'implementazione di quattro interfacce, due delle quali sono facoltative. Il progetto WSRP4J le implementa tutte.
A questo punto vanno indicati tramite un file WSDL (Web Services Description Language) gli Url a cui saranno raggiungibili tali interfacce. Questo file con nome `wsrp_service.wsdl` è reperibile nella directory: [WSRP4J-HomeDirectory] \producer\target\wsrp4j-producer\wsdl.

Viene riportato come esempio:

```
<wsdl:definitions
      targetNamespace="urn:oasis:names:tc:wsrp:v1:wsdl"
      xmlns:bind="urn:oasis:names:tc:wsrp:v1:bind"
      xmlns="http://schemas.xmlsoap.org/wsdl/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
    <import namespace="urn:oasis:names:tc:wsrp:v1:bind"
     location="wsrp_v1_bindings.wsdl"/>
    <wsdl:service name="WSRPService">
        <wsdl:port binding="bind:WSRP_v1_Markup_Binding_SOAP"
         name="WSRPBaseService">
            <soap:address location="http://WSRPServer:8080/
              wsrp4j-producer/WSRP4JProducer/WSRPBaseService"/>
        </wsdl:port>
        <wsdl:port
         binding="bind:WSRP_v1_ServiceDescription_Binding_SOAP"
         name="WSRPServiceDescriptionService">
            <soap:address location="http:// WSRPServer:8080/
              wsrp4j-producer/WSRP4JProducer/
              WSRPServiceDescriptionService"/>
        </wsdl:port>
        <wsdl:port
         binding="bind:WSRP_v1_Registration_Binding_SOAP"
         name="WSRPRegistrationService">
            <soap:address location="http:// WSRPServer:8080/
              wsrp4j-producer/WSRP4JProducer/
              WSRPRegistrationService"/>
        </wsdl:port>
        <wsdl:port
         binding="bind:WSRP_v1_PortletManagement_Binding_SOAP"
         name="WSRPPortletManagementService">
            <soap:address location="http:// WSRPServer:8080/
              wsrp4j-producer/WSRP4JProducer/
              WSRPPortletManagementService"/>
       </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

Come si può notare, le interfacce avranno degli indirizzi del tipo : `http://WSRPServer:8080/wsrp4j-producer/WSRP4Jproducer/[NomeInterfaccia]`.

Per verificare che tali interfacce siano veramente raggiungibili tramite tali Url andrà per prima cosa installato il WSRP4J Producer.

Per fare ciò viene utilizzato Maven che permette di effettuare il build e l'installazione del Producer WSRP4J in Pluto attraverso i seguenti comandi (comandi digitati dentro la directory in cui è stato installato WSRP4J):

```
maven build
maven deploy-producer
```

Terminata questa operazione, per controllare l'effettivo raggiungimento delle interfacce tramite gli url definiti, basterà avviare Pluto, digitare l'indirizzo in un qualunque browser e se tutto è stato configurato a dovere si avrà una schermata che ci avverte che è in esecuzione un servizio AXIS (Apache eXtendable Interaction System) il quale è l'implementazione di SOAP per Java fornita dall'Apache Group.

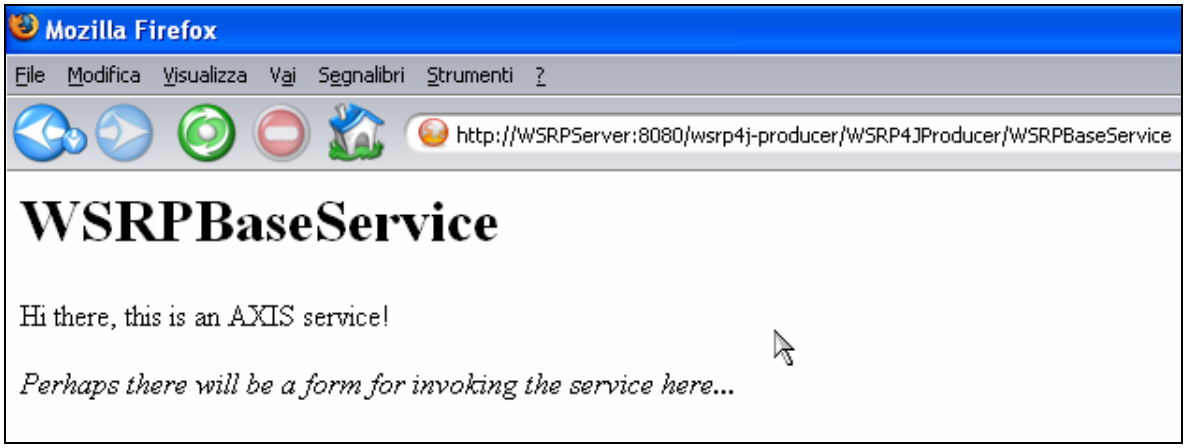

Figura 13: Verifica esecuzione interfacce WSRP

Il WSRP4J utilizza Pluto come Portlet Container, per aggiungere Portlet ospitate dal WSRP4J Producer, può essere utilizzata la funzione (di amministrazione) messa a disposizione da Pluto che permette il deploy delle Portlet come WAR file.
In alternativa può essere utilizzato il seguente maven goal (questo codice inglobato nel file [WSRP4J-HomeDirectory]/testportlet/maven.xml fa riferimento alla Portlet di esempio).

```
<goal name="tomcat:deploy">
  <java classname="org.apache.pluto.driver.deploy.CLI"
   fork="yes">
     <classpath>
        <path refid="maven.dependency.classpath"/>
```

```
        <pathelement path="${maven.build.dest}"/>
      </classpath>
      <arg value="-verbose"/>
      <arg value="-p"/>
      <arg value="${maven.tomcat.home}/webapps/wsrp4j-
       producer"/>
      <arg value="-d"/>
      <arg value="${maven.tomcat.home}/webapps"/>
      <arg value="${basedir}/target/${maven.war.final.name}"/>
    </java>
</goal>
```

2.3.3 Aggiungere Portlet al swingconsumer

1. Nel file `portletentityregistry.xml` ([Pluto-HomeDirectory]\webapps\pluto\WEB-INF\data) copiare l'entry associata alla Portlet che si vuole rendere accessibile tramite lo standard WSRP in [Pluto-HomeDirectory]\webapps\wsrp4j-producer\WEB-INF\data.
2. Aggiungere una entry della Portlet che si vuole rendere visibile dal swing consumer in [WSRP4J-HomeDirectory]\consumer-swingconsumer\target\persistence\pages\org.apache.wsrp4j.commons.consumer.driver.PageImpl@WSRP4JTestPortlets.xml come ad esempio:

```
<portlet-key xmlns:xsi=http://www.w3.org/2001/XMLSchema-
 instance
 xsi:type="java:org.apache.wsrp4j.commons.consumer.
 driver.PortletKeyImpl">
        <portlet-handle>9.0</portlet-handle>
        <producer-id>1</producer-id>
</portlet-key>
```

3. [WSRP4J-HomeDirectory]\consumer-swingconsumer\target\persistence\portlets aggiungere un file (in questo caso poiché la Portlet viene identificata con l'handle 9.0) con il seguente nome `org.apache.wsrp4j.commons.consumer.driver.WSRPPortletImpl@WSRP4J_9_0.xml` Come corpo del file:

```
<Portlet>
    <portlet-key>
```

```
        <portlet-handle>9.0</portlet-handle>
        <producer-id>1</producer-id>
    </portlet-key>
    <parent-handle>9.0</parent-handle>
</Portlet>
```

4. Successivamente basterà digitare il comando

```
maven run-swingconsumer
```

e se tutto è stato configurato correttamente si avrà la visualizzazione delle Portlet che sono state configurate precedentemente.

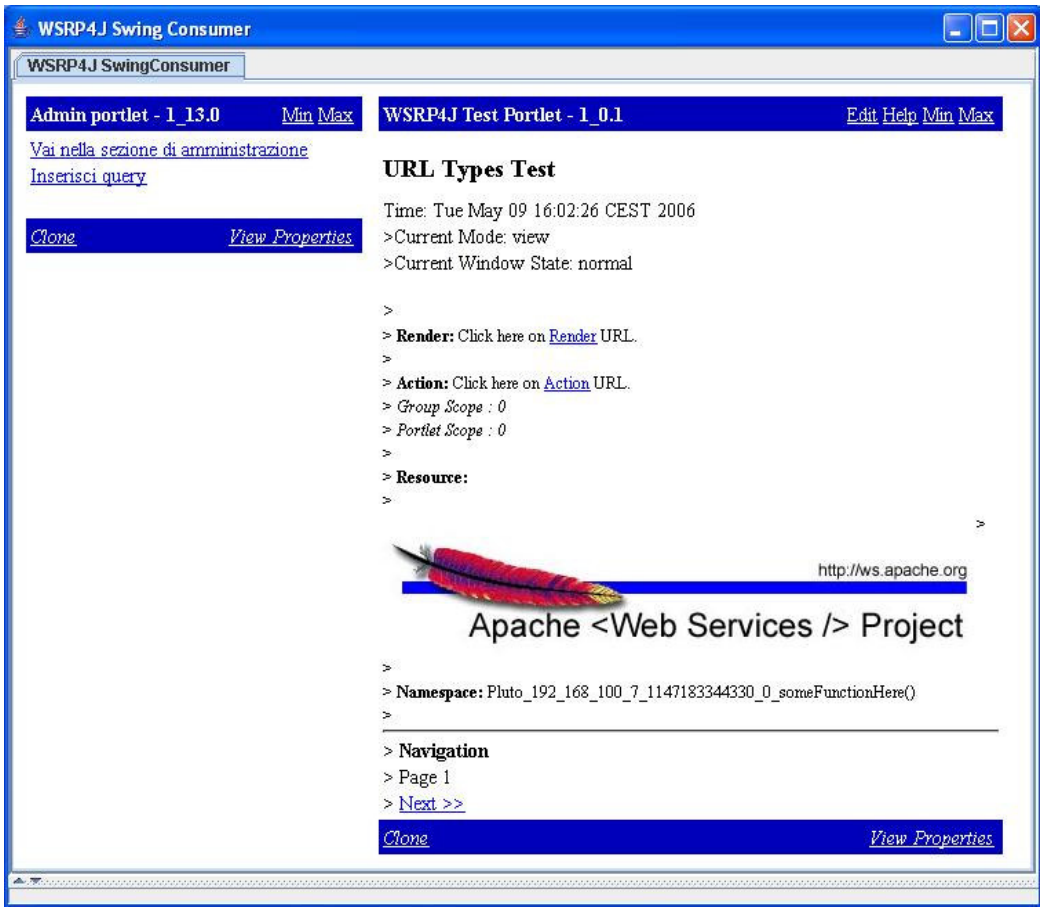

Figura 14: Esempio Swing Consumer

Il swing consumer viene utilizzato in particolare per testare la corretta configurazione delle Portlet che vogliono poter essere messe accedute in modalità remota tramite lo standard WSRP.

Un limite del swing consumer è quello che non permette l'invio e la ricezione dei dati inseriti in un form, questo è un aspetto che ne limita la propria utilizzazione.

2.3.4 Rendere una Portlet accessibile via WSRP

Nel periodo di tirocinio è stato utilizzato Pluto come portale Producer, mentre come portali Consumer Liferay 4.0 ed uPortal 2.5.1.
La prima operazione da eseguire è deployare la Portlet che si intende rendere disponibile via WSRP nel portale Producer (nel nostro caso Pluto). Come verrà ampiamente spiegato nel capitolo dedicato a Pluto, il deploy va ad aggiornare dei file di configurazione per specificare la presenza di una nuova Portlet.
Una volta terminata questa operazione, basterà aprire il file `portletentityregistry.xml` (nella directory [PlutoHomeDirectory]\webapps\pluto\WEB-INF\data) copiare l'entry relativa alla nuova Portlet e copiarla nel file `portletentityregistry.xml` nella directory [PlutoHomeDirectory]\webapps\wsrp4j-producer\WEB-INF\data. A questo punto la Portlet è accessibile tramite lo standard WSRP.
Un esempio di entry associata ad una Portlet è la seguente:

```
<application id="0">
        <definition-id>wsrp4j-testportlet</definition-id>
        <portlet id="1">
            <definition-id>
                   wsrp4j-testportlet.WSRP4JtestPortlet
             </definition-id>
        </portlet>
 </application>
```

2.4 Pluto

2.4.1 Introduzione

Apache Pluto (vedi bibliografia **[1]**) è un subproject Open Source appartenente all'insieme di progetti Apache Portals.
Pluto è l'implementazione di riferimento della specifica riguardante le Portlet (JSR-168). La versione che è stata utilizzata è la 1.0.1 .
Come già detto in precedenza le Portlet devono essere eseguite nel contesto di un portale che mette a disposizione un'infrastruttura che permette:

- l'accesso alle informazioni del profilo utente
- tramite interfacce standard di memorizzare e recuperare impostazioni persistenti

Le Portlet sono sviluppate attraverso l'utilizzo delle Portlet API le quali sono molto simili alle Servlet API anche se di norma le Portlet vengono gestite in modo più dinamico rispetto alle Servlet. Il Portlet Container implementato in accordo con le Portlet API fornisce l'ambiente a runtime per le Portlet, in cui ne viene gestito il ciclo di vita (instanziazione, utilizzazione ed infine distruzione).
Pluto Portal è una Web-Application che mette a disposizione un'implementazione base di un portale (vedi bibliografia **[2]**). Con il termine portale non si intende un enterprise portal ma un'applicazione che permette di:

- semplificare lo sviluppo di Portlet mettendo a disposizione un portale "leggero"
- mettere a disposizione un semplice esempio di come inserire ed invocare il Portlet Container dal portale

Pluto è un semplice portale costruito sopra il Portlet Container e offre agli sviluppatori una piattaforma di esecuzione di esempio da cui è possibile testare le Portlet.
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. Quest'ultimo viene utilizzato per avere un web container in cui fare eseguire il Portlet Container.
Di seguito viene riportata un'immagine che illustra le componenti fondamentali di Pluto:

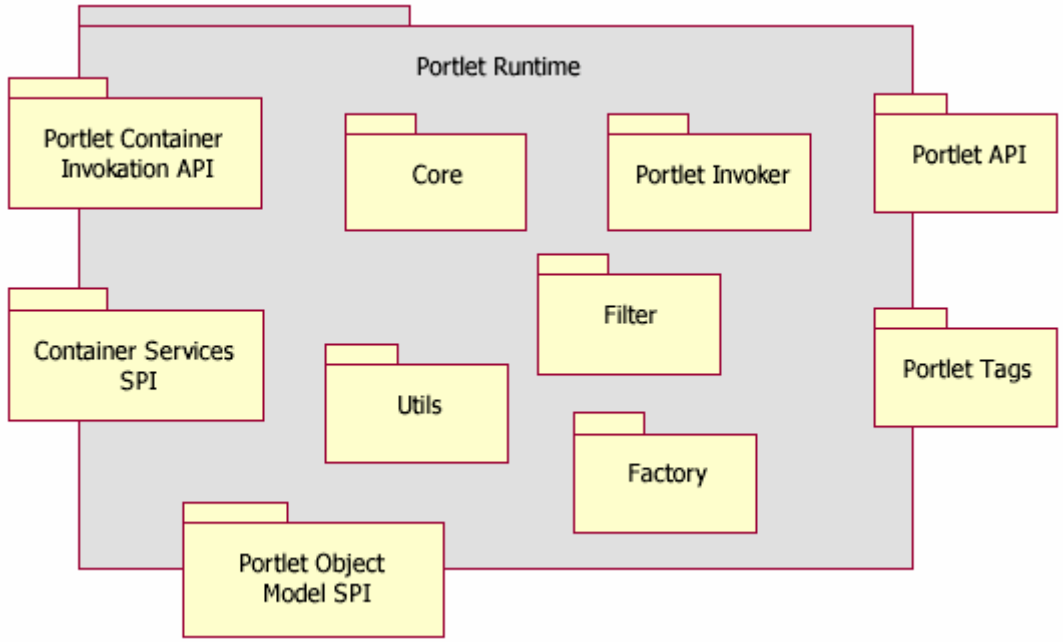

Figura 15: Componenti fondamentali di Pluto

Viene ora riportata una breve descrizione per ogni componente presente nella figura riportata di sopra (vedi bibliografia **[10]**).

- Portlet Container Invoker API: è il punto di ingresso del Portlet Container. È chiamato dal portale quando si deve effettuare il render o un'action sulla Portlet.
- Container Services SPI: includono le interfacce che arricchiscono il Portlet Container di funzionalità tipiche del portale. Il portale implementa queste interfacce e provvede a instanziarle per renderle disponibili al Portlet Container.
- Portlet Object Model: rappresenta un livello di astrazione per i dati, p necessario per eseguire il Portlet Container. Il portale potrà decidere dove memorizzare i dati attraverso il sistema di backend.
- Core: questo package contiene l'implementazione delle Portlet API.
- Portlet Invoker: è l'equivalente del Servlet Request Dispatcher per il "mondo" delle Portlet, mette a disposizione i metodi per invocare action o render delle Portlet.
- Factory: è usato ampiamente affiancato a Pluto, tutti gli oggetti vengono creati utilizzando la factory così da avere massima flessibilità, libertà e miglioramento delle prestazioni.

- Portlet Tags: la specifica delle Portlet definisce i Portlet tags supportati dalle Portlet nelle pagine JSP, questo package ne contiene l'implementazione.
- Filter: Pluto mette a disposizione un Portlet Filter, le Portlet o il Portlet Container possono utilizzarlo per estendere o modificare il comportamento dei metodi delle Portlet.
- Utils:insieme di classi e utilities che rendono più facile il compito degli sviluppatori.
- Portlet API: questo package contiene le API associate alla specifica delle Portlet.

Un altro progetto dell'Apache Software Foundation è Jetspeed, quest'ultimo al contrario di Pluto pone maggiore importanza alle funzionalità del portale rispetto a quello del Portlet Container.

Nelle ultime versioni è anche possibile integrare Pluto come Portlet Container in altri Portali come ad esempio avviene in Jetspeed-2 (vedi bibliografia **[10]**).

Per chiarire i concetti sopra descritti viene riportata una figura che illustra la relazione tra Portale, Portlet Container e Portlet.

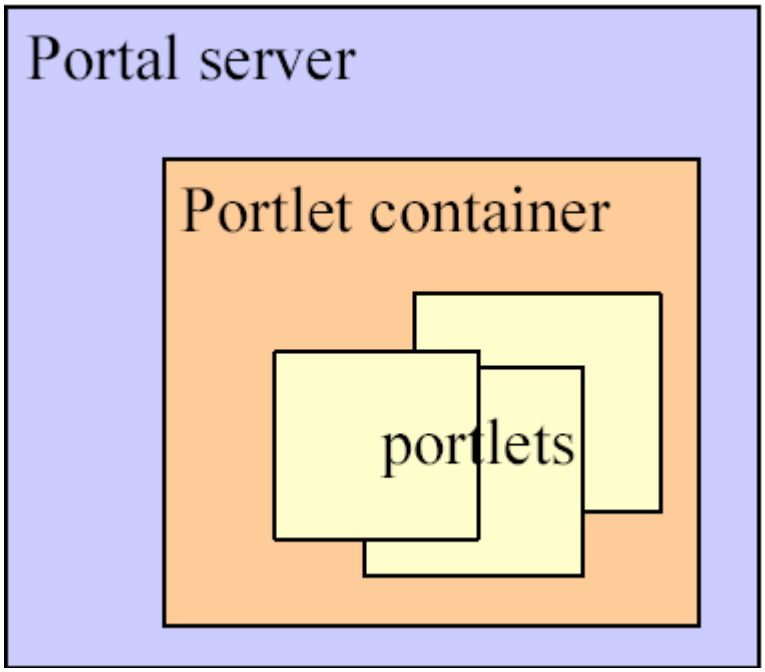

Figura 16: Relazioni tra Portale, Portlet Container e Portlet

2.4.2 Configurazione di Pluto

Come già detto in precedenza, per il proprio funzionamento, Pluto deve essere affiancato da un'Application Server come Tomcat.
Pluto è un progetto Open Source, sono disponibili tre distribuzioni:

- Binari: in cui oltre a Pluto si trova già Tomcat. In questa distribuzione vengono anche inclusi: il Pluto Portlet Container, Portal Driver, Pluto Testsuite e l'Admin Portlet application.
- Library: questa distribuzione è pensata per permettere l'integrazione con un portal server esterno.
- Source: l'installazione è quella che richiede maggiore sforzo, per questo è raccomandata solo per i casi in cui si intende modificare il container. Questa distribuzione è praticamente uno snapshot del codice sorgente

Nel nostro caso è stata utilizzata la distribuzione binaria.
Per testare il corretto funzionamento di Pluto basterà lanciare lo script *startup.bat* (presente nella directory bin di Pluto) e successivamente digitare su un qualunque browser il seguente url:

```
http://localhost:8080/pluto/portal
```

Supponendo:

- di avere Pluto in locale altrimenti si sarebbe dovuto specificare l'indirizzo IP o l'host name del nodo in cui è installato
- che la porta su cui è stato configurato Tomcat sia quella di default ovvero la 8080 (quest'ultima è modificabile nel file `server.xml` presente nella directory conf).

Se tutto è andato a buon fine dovremmo vedere la schermata iniziale di Pluto come mostrato in figura.

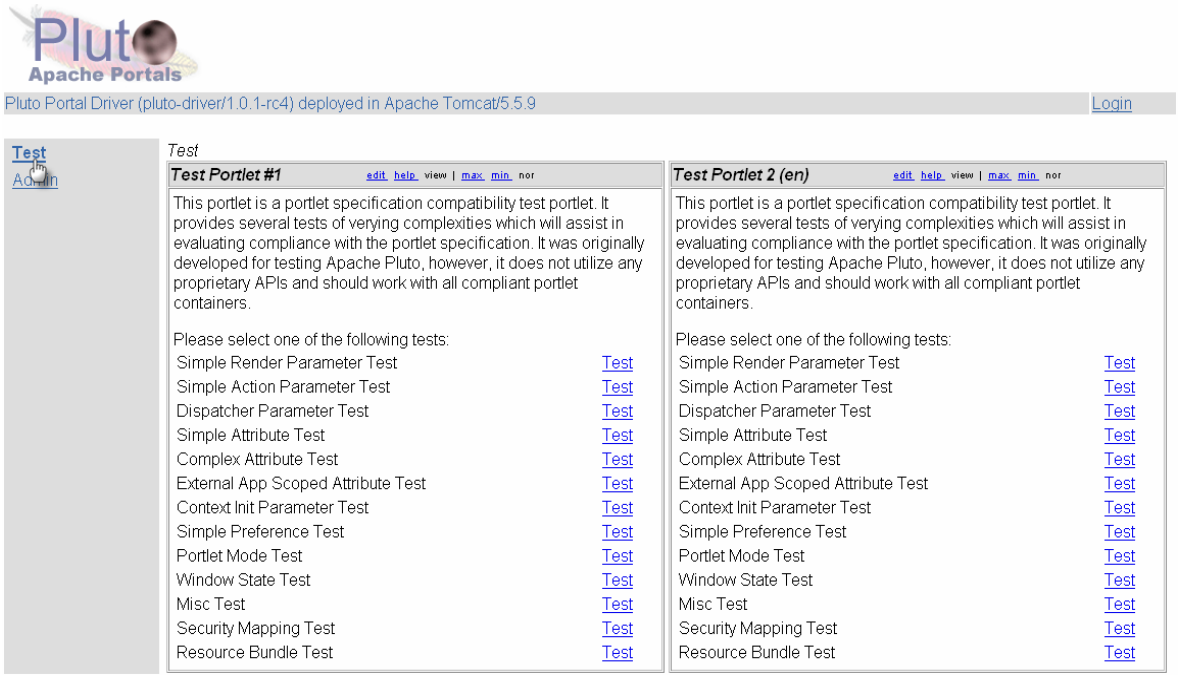

Figura 17: Home page Pluto

2.4.3 Deploy di una Portlet

Per effettuare il deploy di una Portlet si hanno due possibilità: utilizzare Maven o l'Admin Portlet Application.

Per effettuare il deploy di una Portlet con Maven bisogna seguire i seguenti passi:

- Assemblare la Portlet application in un file .war valido
- Eseguire il seguente comando nella directory in cui è installato Pluto:

```
maven deploy -Ddeploy=/MyPathToMyPortlet/target/
  MyPortlet.war
```

- Modificare i seguenti file:
  - [portal-home]/WEB-INF/data/`portletentityregistry.xml` in cui andrà specificato l'ID associato all'applicazione ed un ID univoco per la nuova Portlet. Questi due ID servono poiché un'applicazione potrà contenere al suo interno più Portlet. È necessario anche conoscere il nome della Portlet definito nel file `portlet.xml`. Viene riportato di seguito un frammento del file `portletentityregistry.xml`.

```
<application id="6">
  <definition-id>MyPortlet</definition-id>
   <portlet id="1">
          <definition-id>
                MyPortlet.MyPortlet
          </definition-id>
   </portlet>
</application>
```

- [portal-home]/WEB-INF/data/`pageregistry.xml` questo file viene utilizzato da Pluto per le informazioni di layout della Portlet. Viene riportato di seguito un frammento del file.

```
<fragment name="MyPortlet" type="page">
    <navigation>
          <title>My First Portlet</title>
          <description>First Portlet</description>
    </navigation>
    <fragment name="row3" type="row">
          <fragment name="col3" type="column">
             <fragment name="p4" type="portlet">
                <property name="portlet" value="6.1"/>
             </fragment>
          </fragment>
    </fragment>
</fragment>
```

Da notare che il valore 6.1 è dovuto al fatto che 6 è l'identificativo dell'applicazione e 1 quello della Portlet così come è stato definito nel file `portletentityregistry.xml`.

- [Portal-HomeDirectory]/WEB-INF/data/`portletcontexts.txt` questo file contiene l'elenco delle Portlet application che sono state deployate in Pluto. Ogni applicazione avrà la linea corrispondente in questo file, ad esempio:

```
/testsuite
/pluto
/wsrp4j-testportlet
/PublicationPortlet
/AdminPortlet
```

Prendiamo ora in considerazione l'utilizzo dell'Admin Portlet Application.
L'Admin Portlet Application permette il deploy delle Portlet utilizzando un'interfaccia grafica direttamente inclusa in Pluto. Questa applicazione inserisce automaticamente le Portlet che vogliono essere deployate aggiornando in modo opportuno i file di configurazione.
Per effettuare il deploy con l'Admin Portlet Application basterà seguire i seguenti passi:

- Assemblare la Portlet application in un file .war valido.
- Lanciare Pluto e una volta visualizzata la home page selezionare il link Admin così che appaia l'Admin Portlet Application come indicato in figura:

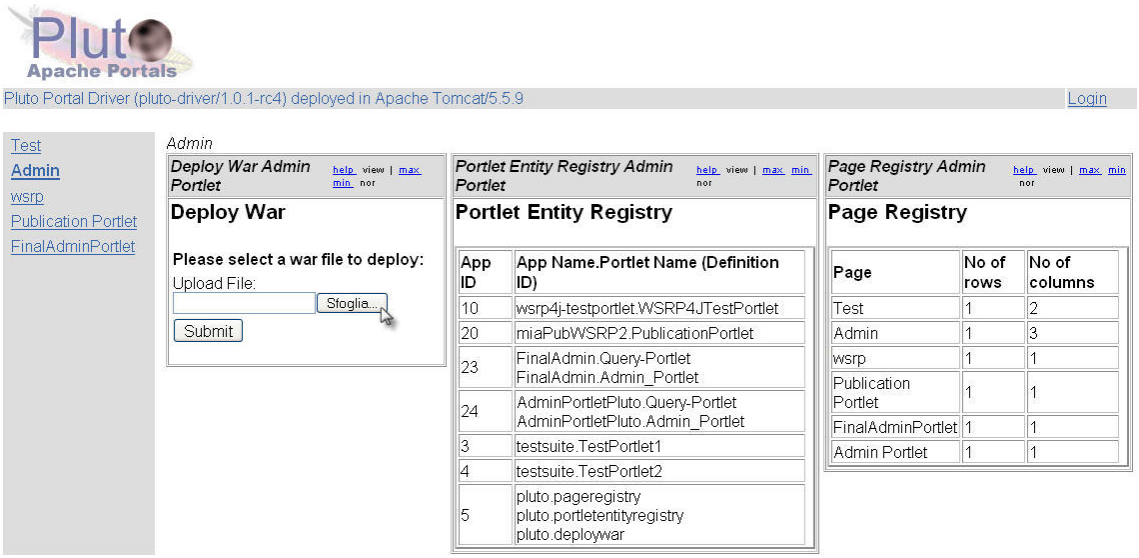

Figura 18: Deploy con l'Admin Portlet Application

- Una volta cliccato sul pulsante sfoglia e selezionato il file (.war) da deployare andrà confermata la scelta agendo sul bottone Submit.

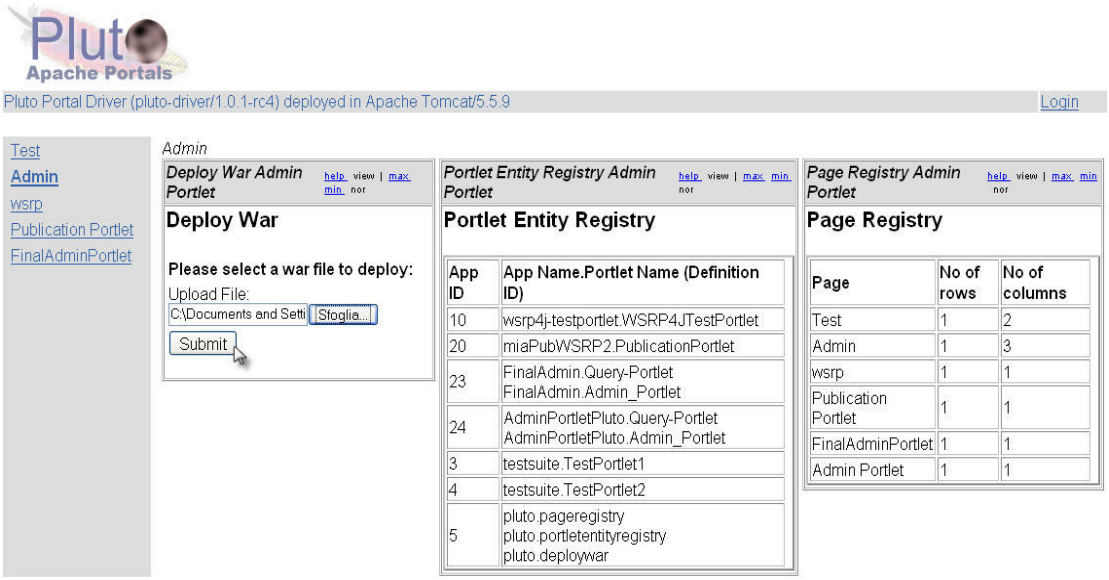

Figura 19: Selezione file war per il deploy

- Nella pagina che verrà visualizzata verrà richiesto di inserire il titolo e opzionalmente la descrizione della Portlet. Inoltre andranno specificate il numero delle righe e delle colonne che si desiderano essere presenti nel layout. Questa possibilità è stata inserita poiché in un'applicazione possono coesistere più Portlet.

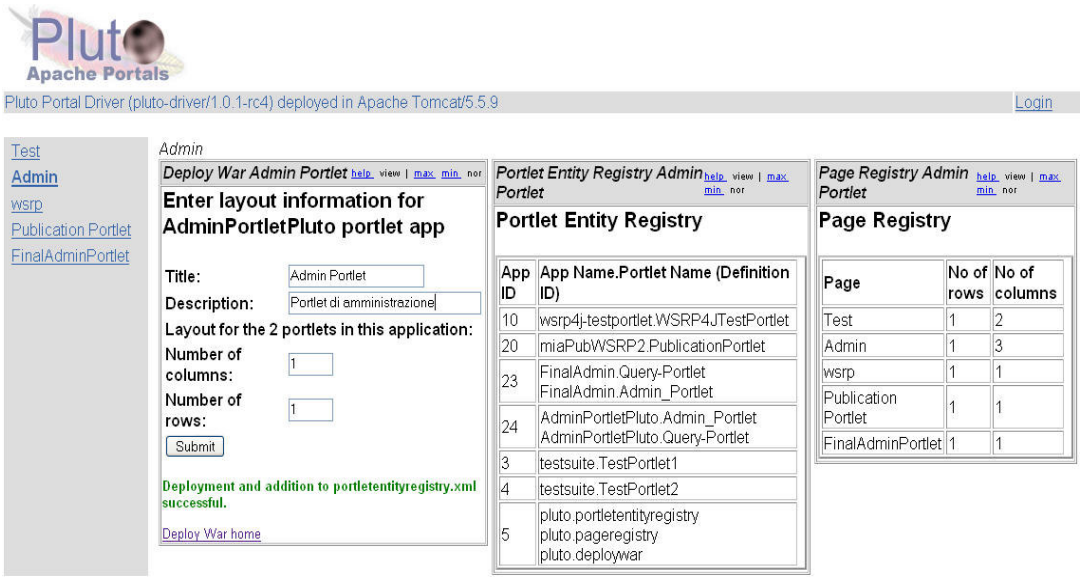

Figura 20: Inserimento informazioni generali legate alla Portlet

- Nella pagina successiva verrà chiesto di selezionare la/le Portlet da deployare.

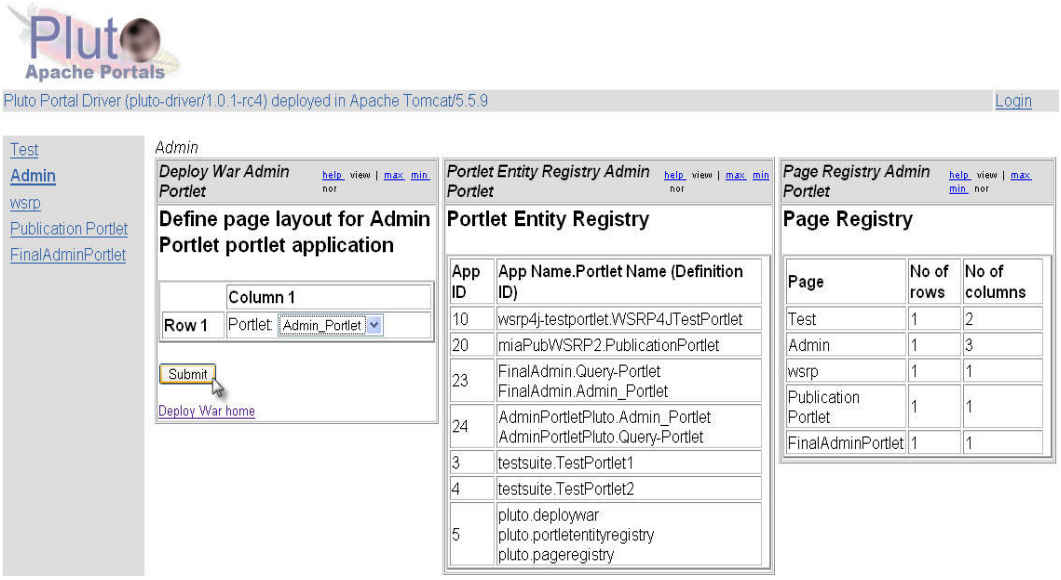

Figura 21: Selezione Portlet da deployare

- A questo punto se tutto è andato a buon fine verrà mostrato un messaggio che ci avverte dell'avvenuto deploy della Portlet e per poter effettivamente utilizzare la nuova Portlet basterà riavviare Pluto.

2.4.4 Limitazioni

Una limitazione incontrata nell'utilizzo di Pluto riguarda la gestione delle `PortletPreferences` nel caso in cui si abbiano nella stessa pagina più instanze della medesima Portlet.

Le preferenze vengono utilizzate (come già detto in precedenza) per rendere la configurazione delle Portlet persistente.

Per sapere a quale instanza di Portlet è associata una data configurazione viene utilizzato il `PortletHandle` che funge da identificatore univoco. Il problema è dovuto dal fatto che Pluto utilizza lo stesso handle per ogni instanza della Portlet presente nella pagina e quindi la configurazione verrà di volta in volta riscritta. Questo

comporterà che al successivo utilizzo, verrà ritrovata solamente l'ultima configurazione effettuata.
Supponiamo di avere una Portlet con handle 24.0, e di volerne quattro instanze, il file `pageregistry.xml` sarà simile a quello riportato di seguito.

```
<fragment name="MyApp" type="page" >
   <navigation>
           <title>MyApp</title>
           <description>Test Appl</description>
  </navigation>
  <fragment name="row1" type="row">
           <fragment name="col1" type="column">
                  <fragment name="p1" type="portlet">
                      <property name="portlet" value="24.0"/>
                  </fragment>
                 <fragment name="p2" type="portlet">
                      <property name="portlet" value="24.0"/>
                 </fragment>
           </fragment>
  </fragment>
  <fragment name="row2" type="row">
           <fragment name="col3" type="column">
                  <fragment name="p3" type="portlet">
                      <property name="portlet" value="24.0"/>
                  </fragment>
                  <fragment name="p4" type="portlet">
                      <property name="portlet" value="24.0"/>
                 </fragment>
           </fragment>
  </fragment>
</fragment>
```

Come si può notare dalla porzione di file riportata, ogni instanza della Portlet viene riferita con lo stesso `Portlet-Handle` (24.0) non rendendo possibile l'utilizzo delle `PortletPreferences` nel caso appena esposto.
Da notare che le `PortletPreferences` potranno essere utilizzate senza problemi nel caso in cui nella pagina del portale si abbia un'unica instanza per ogni Portlet poiché, Portlet diverse saranno identificate con Handle diversi.

2.5 uPortal

2.5.1 Descrizione generale

Il framework uPortal 2.5.1 è un portale utilizzato e sviluppato da numerose Università nel mondo con lo scopo di realizzare una rappresentazione sul Web del Campus Universitario (vedi bibliografia **[4]**). Per questo motivo è stato definito anche come Campus Web. Lo scopo di uPortal è quello di consentire agli utenti di diverso livello (Professori, Tecnici, Amministratori del Sistema e Studenti) di poter gestire in maniera del tutto personale il proprio Campus Web. uPortal è basato su standard aperti come: Java, J2EE, XML e JSP. La sua versatilità è proprio uno degli aspetti più apprezzabili. Il framework è open source e gratuito, sviluppato e mantenuto dall'organizzazione "Java Architectures Special Interest Group" (JA-SIG) che ha lo scopo di promuovere l'uso delle tecnologie e architetture JAVA. Il supporto della JA-SIG all'implementazione ed estensione del framework uPortal garantisce una continua crescita del sistema basato su standard aperti ed ha l'enorme vantaggio di non essere legato ad alcun tipo azienda, con la possibilità di avere accesso al codice sorgente rendendo l'estensione estremamente flessibile.

Ecco i motivi per cui molte università (soprattutto americane) hanno utilizzato uPortal:

- indipendente dalla piattaforma (il framework è stato completamente sviluppanto utilizzando standard aperti come: Java, J2EE, XML e JSP, questo ne permette l'utilizzo in qualsiasi piattaforma)
- database indipendente (si possono usare Oracle, MSSQL, Postgresql, MySQL,...)
- gratuito e open-source (non si è legati ad alcun venditore);
- flessibile: è possibile integrare servizi ed applicazioni esistenti (sistema di autenticazine, CMS, motore di ricerca, ecc.)
- scalabile
- supportato da una comunità internazionale
- funziona: è utilizzato da più di 80 Università

uPortal affronta il problema dell'accesso a un portale tenendo presente i diversi interessi e necessità dei vari utenti che si collegano.

Gli utenti possono essere aggiunti o eliminati tramite la strumentazione fornita con uPortal (target per ANT), oppure, con procedure da realizzare ad hoc, recuperati da

database esistenti (vedi bibliografia**[5]**). La configurazione standard prevede gli utenti admin, developer, demo, student, staff, faculty con username e password coincidenti. In particolare con l'utente admin è possibile gestire gruppi di utenti, assegnare le relative appartenenze e stabilire i vari permessi di accesso.
Per gestire gli utenti di uPortal è disponibile un sistema a gruppi. Per ciascun gruppo, impostando opportunamente i diritti, è possibile creare delle sottocategorie per una massima diversificazione dei contenuti e delle funzionalità proposte con i vari canali.
Quando si utilizza uPortal è necessario porre attenzione al problema di gestione dei tasti Indietro (Back) e Avanti (Forward) dei Browser. Nelle procedure guidate per le varie impostazioni del portale è necessario utilizzare esclusivamente i pulsanti di spostamento della pagina, e non quelli del browser, altrimenti le procedure vengono invalidate.
La gestione di uPortal per un utente di tipo Student è analoga a quella di un utente di tipo Faculty o Staff. Le varie pagine personali si differenzieranno per i contenuti disponibili ed inseriti dall'utente Admin.
Le icone in alto a destra nella pagina indicano le operazioni che l'utente può compiere sul portale:

- *Home* consente all'utente in qualsiasi momento della navigazione di tornare alla pagina iniziale.
- *Turn on preference* consente di selezionare i canali che desidera visualizzare nella propria pagina.
- *LogOut* il quale come dice il nome permette di effettuare il logout dell'utente.
- *View Sitemap* che permette di visualizzare la mappa del portale.

L'utente di tipo Admin dispone di un'interfaccia simile a quella degli altri utenti, ma oltre alle opzioni Home, Preferences e Logout è presente anche l'opzione Channel Admin. Questa pagina permette di pubblicare, modificare o rimuovere canali ed impostare i permessi di accesso e l'appartenenza per i vari gruppi e sottogruppi.
Cliccando sull'icona Turn on Preference e seguendo il link skin è possibile inoltre modificare l'aspetto delle proprie pagine (nell'installazione standard sono presenti quattro tipologie differenti di skin).

2.5.2 I canali

La struttura di uPortal prevede la presenza all'interno della stessa pagina Web di diverse finestre indipendenti, queste prendono il nome di canali. I canali vengono suddivisi dall'amministratore in diversi gruppi in base all'utilità e alle funzioni svolte. I gruppi predefiniti sono Applications, Development, Entertainment e News.
Un canale può essere un programma scritto in Java, un applet Java, un'immagine,ecc... Viene creato ed inserito per diversi scopi, ad esempio: la creazione di un forum, di una bacheca di Annunci, l'inserimento della pagina per l'iscrizione agli esami, oppure un'immagine con le previsioni metereologiche.
Ciascun utente può configurare uPortal a proprio uso e consumo, inserendo i canali che preferisce e che pensa gli possano essere più utili. La creazione e la pubblicazione di tutti gli utenti di nuovi canali è riservata all'amministratore del sistema. Queste caratteristiche consentono una notevole flessibilità e indipendenza degli accessi, ma non è tutto, perché uPortal, tramite una generazione dei documenti da XML con fogli di stile, permette anche di avere indipendenza dei contenuti dalla modalità e dallo strumento di visualizzazione (ad esempio device con browser WAP).
La versione di uPortal che è stata utilizzata è la 2.5.1 l'ultima versione rilasciata al momento, per il funzionamento ha bisogno del jdk1.5.0, inoltre andrà settata la variabile d'ambiente `JAVA_HOME` nel percorso in cui è stato appunto installato il jdk.
Lanciare uPortal

- Nella directory in cui è installato uPortal lanciare il comando *ant hsql* avendo preventivamente settato la variabile d'ambiente `ANT_HOME` nella directory in cui è stato installato ant.
- Lanciare Tomcat eseguendo il file *startup.bat*.
- A questo punto è possibile testare il funzionamento di uPortal digitando nel browser il seguente URL:

```
http://localhost:8080/uPortal
```

Per poter testare le Portlet secondo lo standard WSRP se si lavora con un'unica macchina oltre ad eseguire uPortal (che nel nostro caso è il Consumer) è necessario lanciare anche il Producer (ovvero Pluto).

Proprio per questo è stato necessario modificare le porte che uPortal utilizza per il proprio funzionamento. Ovviamente in alternativa potevano essere modificate le porte utilizzate da Pluto.
Per la modifica delle porte di uPortal basta andare nella directory in cui è installato Tomcat (ad esempio:[Portal-Home]\Tomcat_5-5-9), successivamente nella directory conf bisogna editare il file `server.xml` e modificare le tre porte che Tomcat utilizza per il proprio funzionamento:

- La porta in cui Tomcat riceve il segnale di shutdown (di default impostata a 8005, nel nostro caso settata a 8006)

```
    <Server port="8006" shutdown="SHUTDOWN">
```

- La porta in cui Tomcat riceve le richieste http (di default impostata a 8080, nel nostro caso settata a 8081)

```
<Connector port="8081" maxHttpHeaderSize="8192"
   maxThreads="150" minSpareThreads="25"
   maxSpareThreads="75"
   enableLookups="false" redirectPort="8443"
   acceptCount="100"
   connectionTimeout="20000" disableUploadTimeout="true"
/>
```

- La porta in cui Tomcat riceve le richieste inoltrate da Apache (di default impostata a 8009, nel nostro caso settata a 8010)

```
<Connector port="8010" enableLookups="false"
 redirectPort="8443" protocol="AJP/1.3" />
```

Per testare il funzionamento di uPortal con la nuova configurazione basterà digitare il seguente URL:

```
  http://localhost:8081/uPortal
```

Se tutto è stato configurato a dovere verrà visualizzata la seguente pagina iniziale:

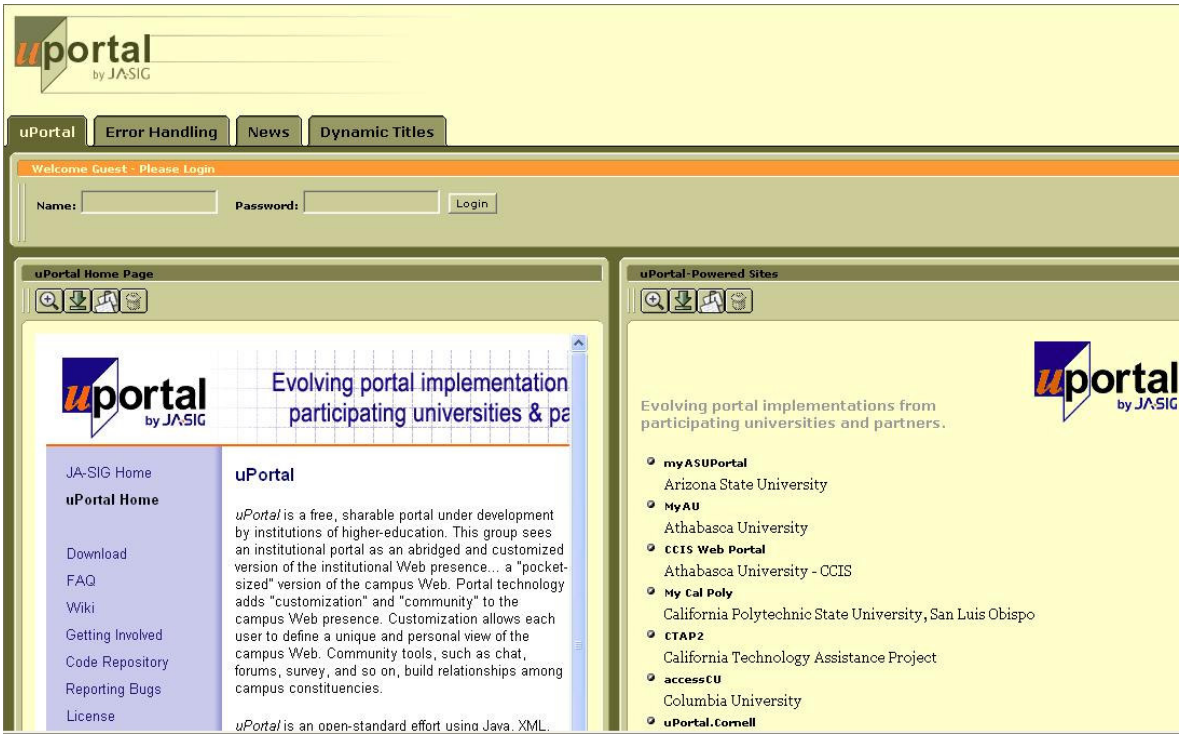

Figura 22: Home page uPortal

A questo punto per accedere all'area di amministrazione è necessario effettuare il login inserendo come username e password *admin*.

2.5.3 Configurazione del canale WSRP Consumer

Una volta essere acceduti nella zona di amministrazione, è necessario configurare i canali WSRP Consumer per poter appunto consumare le Portlet messe a dispostone in modalità remota secondo lo standard WSRP.

Per fare ciò basta cliccare sull'icona del *Channel Manager* posta in alto a destra, successivamente viene visualizzata una pagina in cui si chiede di selezionare quale tipo di canale si intende configurare, nel nostro caso sarà WSRP Consumer.

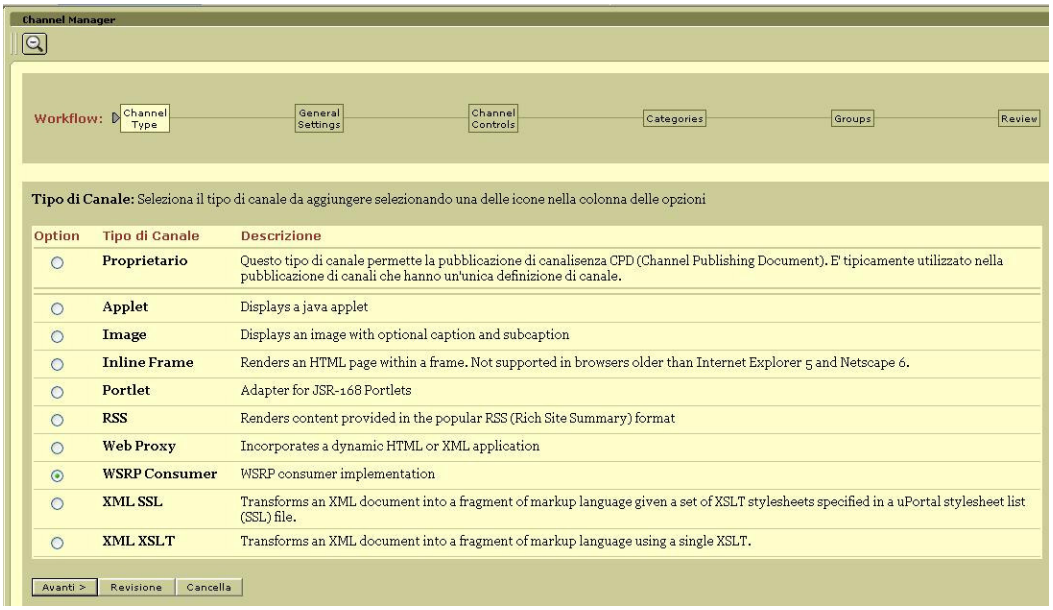

Figura 23: uPortal Canali

Andando avanti, viene richiesto di inserire le informazioni generali del canale, quali: titolo, nome, nome funzionale, descrizione del canale. Un'altra informazione molto importante che viene richiesta è il timeout del canale, lo standard WSRP non è veloce quindi per evitare errori dovuti alla scadenza del timeout, viene consigliato di settarlo almeno a 60000 millisecondi.

Viene riportato di seguito un esempio di impostazioni generali:

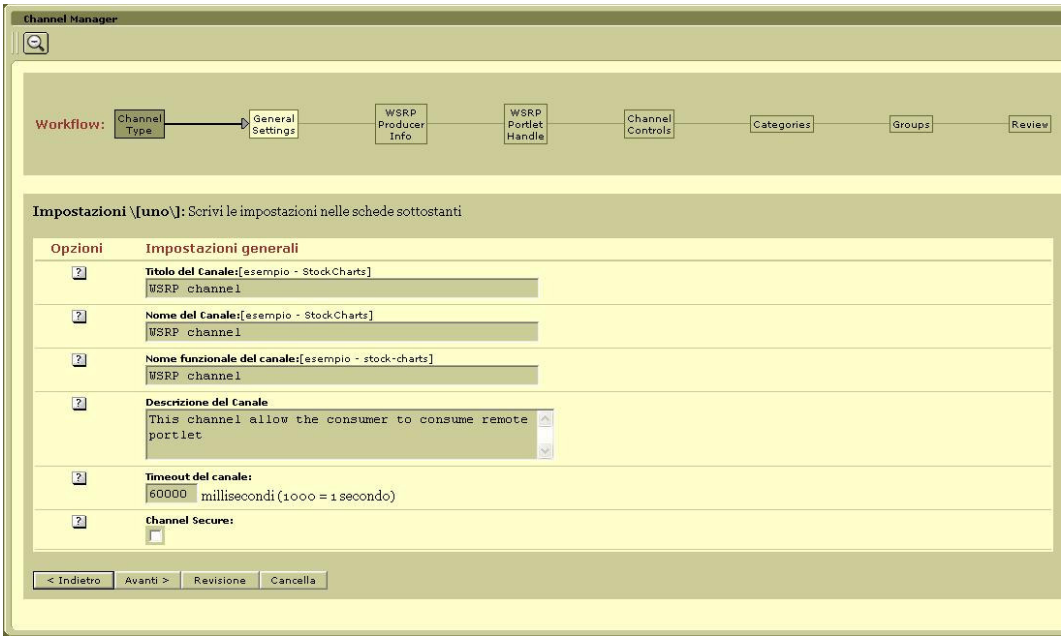

Figura 24: Impostazioni generali uPortal

Proseguendo, bisogna inserire gli Url mediante i quali il Consumer (nel nostro caso è uPortal) possa reperire le quattro interfacce del Producer (nel nostro caso Pluto).
Le interfacce sono le seguenti:

- `Markup Interface`
- `Service Description Interface`
- `Registration Interface`
- `Portlet Management Interface`

Avendo utilizzato il WSRP4J, gli Url associati alle quattro interfacce (definite dallo standard WSRP) possono essere reperiti nel file: `org.apache.wsrp4j.commons.consumer.driver.ProducerImpl@WSRP4J.xml` presente nella directory: [WSRP4J-HomeDirectory]\consumer-swingconsumer\target\persistence\producers.
A questo punto basterà copiare gli Url nelle giuste sedi, come mostrato di seguito:

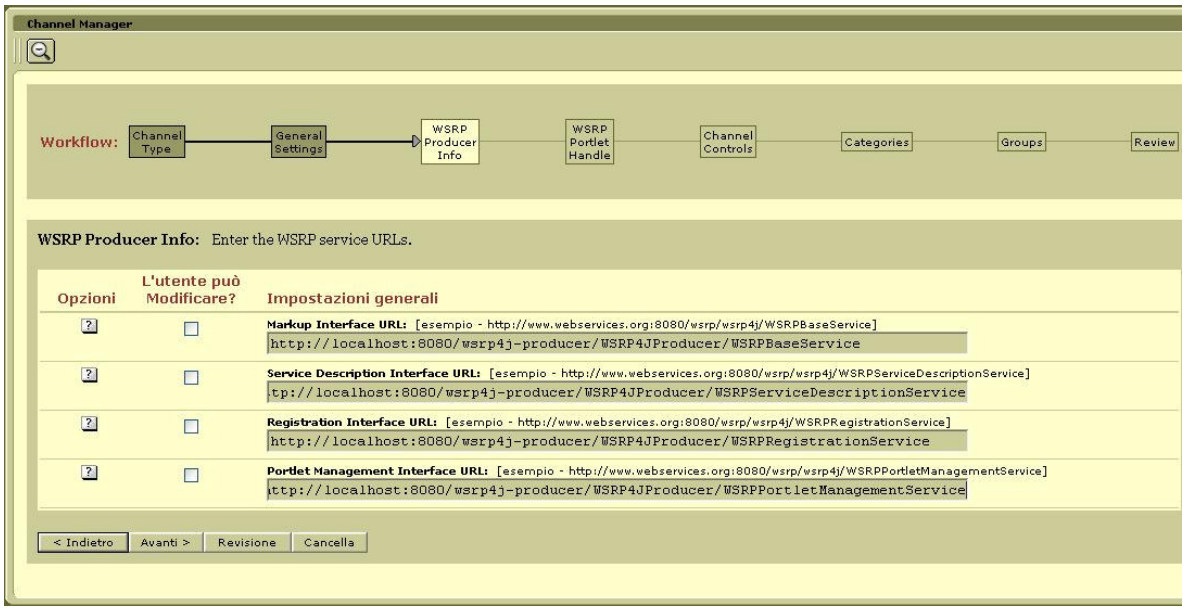

Figura 25: Interfacce uPortal

Successivamente viene richiesto il `PortletHandle` della Portlet che si intende consumare, questo può essere reperito nel file `portletentityregistry.xml` presente nella directory:
[Pluto-Home-Dir]\webapps\wsrp4j-producer\WEB-INF\data

Una volta inserito, proseguendo vengono richieste informazioni accessorie come l'appartenenza a determinati gruppi, categorie e utenti.
A questo punto il canale è stato configurato, adesso va aggiunto come contenuto nel nostro portale. Per fare ciò, bisogna selezionare *Turn on Preference* cliccando sull'icona in alto a destra.
Per comodità conviene creare un nuovo tab, così da poter effettuare tutti i test delle Portlet, come esempio creiamo un tab con nome My Test.
Successivamente aggiungiamo una nuova colonna e di seguito un nuovo contenuto (cliccando sul link *add content*). A questo punto va selezionata la categoria a cui è stato associato il canale e conseguentemente va scelto il canale che abbiamo appena creato. Fatto questo, basta cliccare in quale posizione collocare la Portlet e la Portlet viene visualizzata.

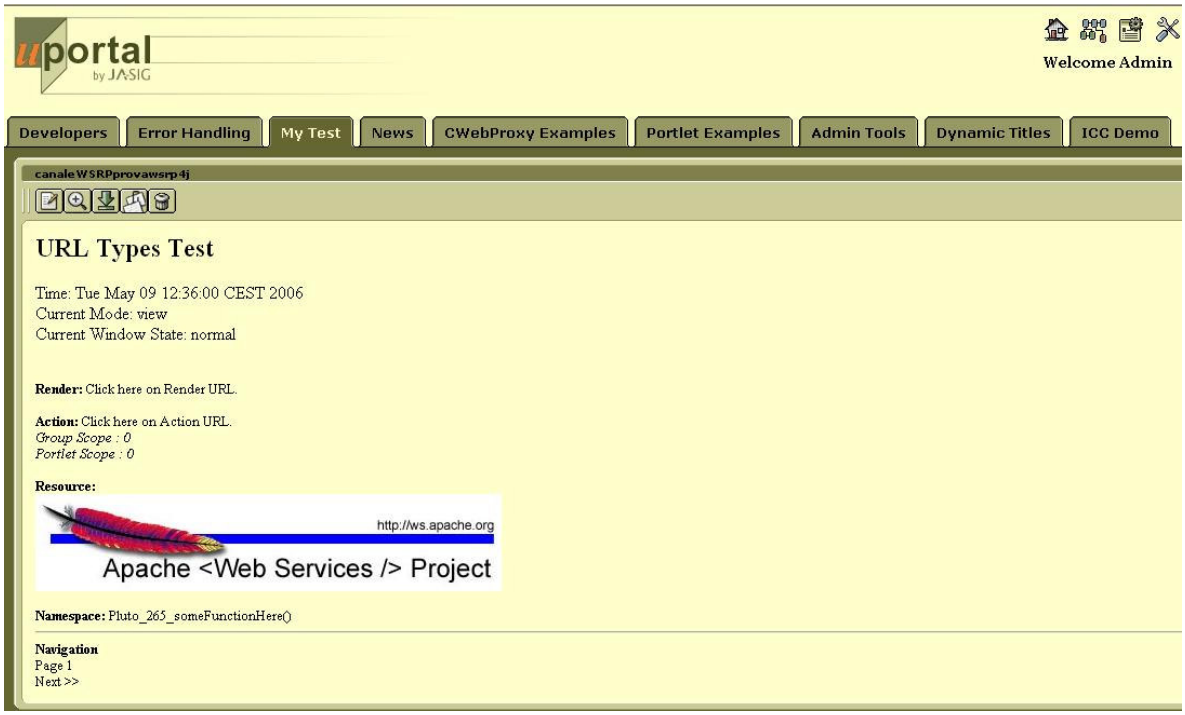

Figura 26: Esempio Portlet remota uPortal

Da ricordare che per poter consumare le Portlet remote, deve essere in esecuzione il Producer quindi è necessario avviare Pluto (nel nostro caso) prima di aggiungere la Portlet alla pagina.

2.5.4 Limitazioni

Una grande limitazione di uPortal è quella che non è possibile includere nella stessa pagina (o meglio nello stesso tab) due o più Portlet remote perché il portale invece di visualizzare le due Portlet distinte potrebbe visualizzare due Portlet uguali.
Viene riportato di seguito un esempio in cui nello stesso tab sono state inserite due Portlet.

Figura 27: Esempio uPortal due Portlet corretto

Cliccando ad esempio nel link Inserisci query della Portlet di sinistra ecco il risultato (errato) che si ottiene:

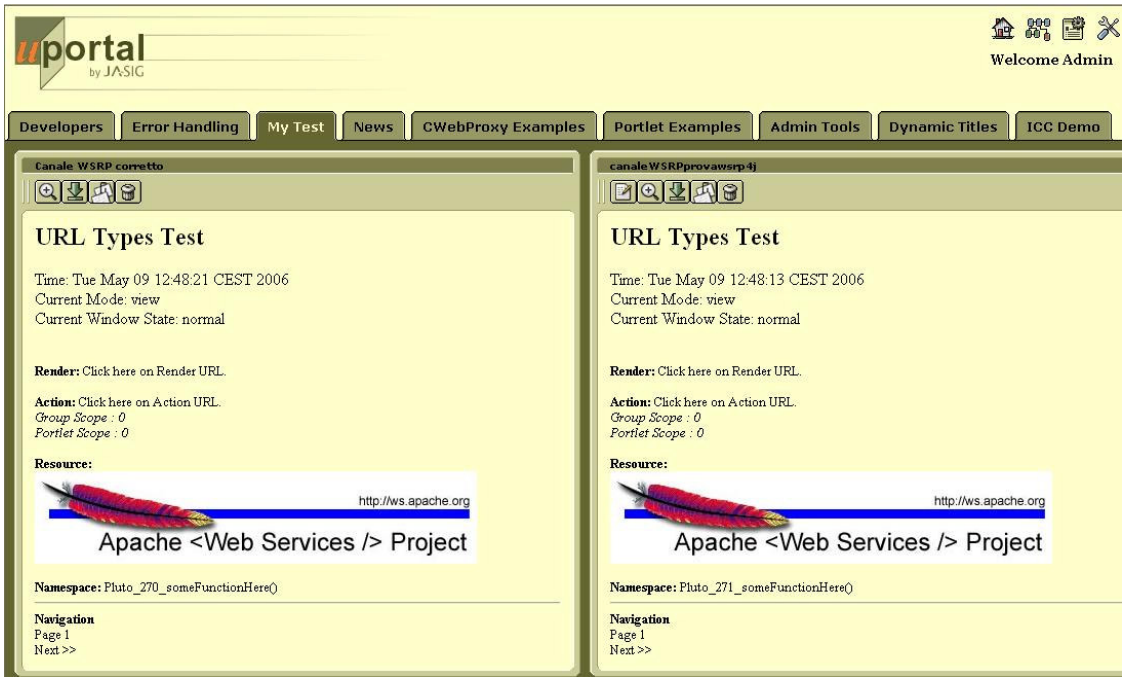

Figura 28: Esempio uPortal due Portlet errato

Si sarebbe ottenuto il risultato opposto quindi replicata due volte la Portlet di sinistra se il link ad essere selezionato fosse stato quello della Portlet di destra.

Questa limitazione viene risolta in fase di inserimento delle nostre applicazioni nel portale, inserendo solamente una Portlet per colonna come mostrato di seguito.

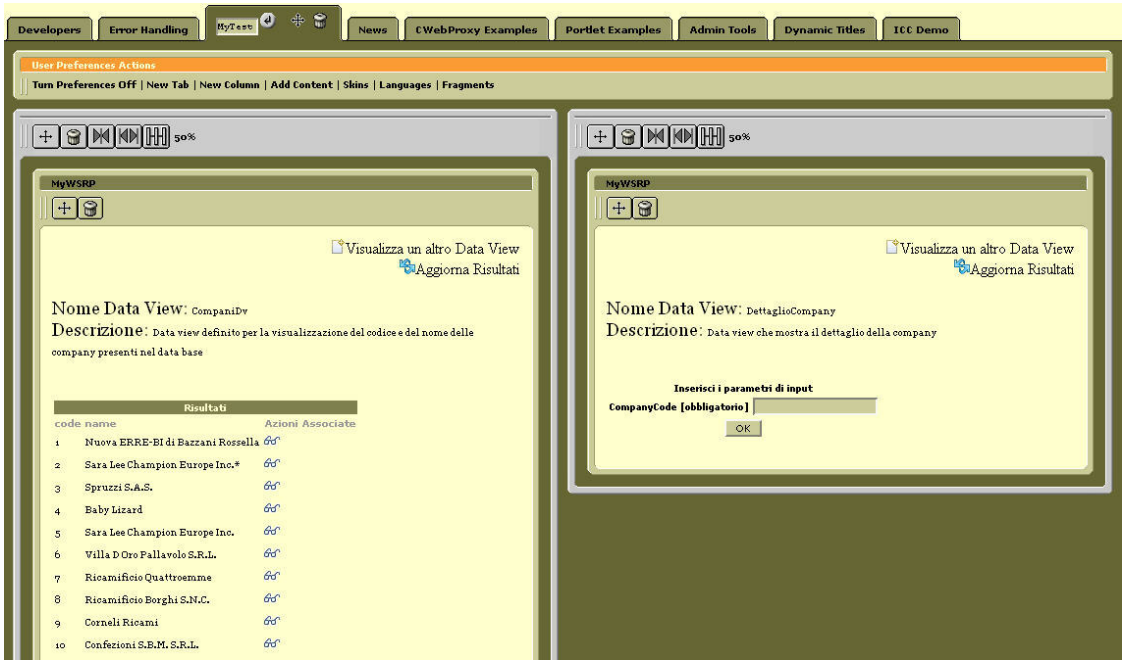

Figura 29: Esempio uPortal due Portlet corretto

Nell'utilizzo di uPortal sono state riscontrate altre limitazioni affrontate con maggiore dettaglio e proponendo la soluzione adottata nel capitolo dedicato alla descrizione delle applicazioni sviluppate.
Per completezza vengono comunque presentate due limitazioni (affrontate in seguito):

- l'impossibilità di utilizzare l'`edit` mode per le Portlet remote. Poiché il portale non permette di ritornare dopo la configurazione in `view` mode
- mancata renderizzazione ad ogni richiesta di tutte le Portlet nella pagina

2.6 Liferay

2.6.1 Descrizione generale

Liferay Portal 4.0 (vedi bibliografia**[3]**) è un portale web open source che si propone di aiutare le organizzazioni a collaborare in modo più efficiente fornendo una serie consolidata di applicazioni pronte all'uso.
È utilizzato da piccole e grandi aziende in tutto il mondo e comprende una lunga lista di funzionalità che lo mettono a confronto direttamente con molti portali commerciali, con il vantaggio di non avere oneri di licenza.
Ogni giorno nascono progetti open source nuovi, quindi spesso ci troviamo di fronte alla difficoltà di scegliere il portale che meglio si adatta ai nostri scopi.
Vengono riportate di seguito alcune caratteristiche supportate dal portale:

- JSR 168 (Portlet API) Compliant.
  È possibile fare il deploy di qualsiasi Portlet che rispetti le specifiche JSR 168. Questo vuol dire che si potranno aggiungere funzionalità al Portale scrivendo delle Portlets standard, o comprandole da Portlet vendors.
- WSRP Compliant.
  Questa caratteristica di Liferay sarà descritta con maggiore dettaglio nei paragrafi seguenti.
- CMS.
  Liferay offre una serie di Portlets che fanno da CMS per la costruzione di un sito statico esterno al portale. Questo sistema CMS è basato su un meccanismo a template che permette tramite XML/XSL/XSLT di separare completamente i dati redazionali dalle informazioni di impaginazione. È previsto un workflow minimale per l'approvazione dei contenuti. Le pagine sono poi servite da servlet apposite.
- SSO.
  Liferay comprende un connector per l'engine single sign on CAS, ma è comunque possibile integrarsi con altri motori di SSO quali Netegrity. Liferay può sincronizzare la sua lista utenti con DataSource esterni o sistemi LDAP in più è anche incluso un connector per Microsoft Exchange.

- ASP Model.
  Liferay è stato disegnato fin dall'inizio per essere utilizzato da Application Service Providers, quindi è possibile farci girare sopra più istanze di portali completamente indipendenti.
- Application Server Agnostic.
  Liferay può essere installato praticamente su qualsiasi servlet container o application server standard J2EE.
  Liferay funziona sia su Tomcat, Jetty, etc., che su application server commerciali come Borland ES, Oracle9iAS, Weblogic, etc. Ovviamente essendo scritto in Java la propria esecuzione non è legata al tipo di sistema operativo.
- Spring, EJB, and AOP.
  Lo strato business di Liferay è scritto utilizzando Spring. Ciò permette di utilizzare le caratteristiche AOP, IOC e Proxy di Spring per personalizzare più agevolmente il codice. Utilizzando Spring si può scegliere se utilizzare gli strati di servizio POJO o gli strati EJB.
- Database Agnostic.
  Liferay utilizza Hibernate come tool di persistenza, quindi può girare su qualsiasi database da esso supportato. Attualmente Hibernate supporta una vasta gamma di database quali : DB2, Firebird, Hypersonic, InterBase, JDataStore, MySQL, Oracle, PostgreSQL, SAP, SQL Server.
- Scalable N-Tier Cluster.
  Liferay utilizza OSCache per offrire un livello di cache clusterizzata. Quindi è possibile scalare i deployment senza sacrificare il livello di caching.
- Struts and Tiles.
  Lo stato di presentation utilizza Struts come framework MVC. Il suo utilizzo permette agli sviluppatori di trovare un ambiente "familiare" per lo sviluppo di nuove Portlet. Il Look and Feel del portale può essere facilmente modificato grazie al fatto che la presentazione si basa su templates Tiles.
- Internationalization.
  Liferay è sviluppato utilizzando l'internazionalizzazione e include traduzioni per una dozzina delle lingue più diffuse al mondo.

- Personalization.
  Il portale permette agli utenti di creare le proprie pagine e di disporre le Portlet a piacimento all'interno di schemi pre impostati (una, due, tre colonne).
- Administration.
  Liferay comprende delle Portlet di amministrazione per la gestione di Utenti, Ruoli, Gruppi e Permessi. I Gruppi sono insiemi di utenti, i Ruoli sono dei permessi che possono essere assegnati a gruppi o a singoli utenti. L'accesso alle Portlet è determinato in base ai Ruoli.
  Gli amministratori possono gestire le pagine visibili dai vari gruppi definiti.
- Portlet già pronte.
  Uno dei punti di forza di Liferay è la grossa dotazione di Portlet già pronte. Tra le varie Portlet troviamo Blogs, Calendar, Document Library, Journal (CMS), Image Gallery, Mail, Message Boards, Polls, RSS e Wiki.

Liferay è distribuito in due versioni : Enterprise e Professional. Sostanzialmente le due distribuzioni sono identiche ma differiscono architetturalmente. L'Enterprise infatti è fatta per essere installata su un'application server che supporti gli EJB come JBoss o Borland ES. La versione Professional (quella che effettivamente è stata utilizzata nel periodo di tirocinio) può essere installata su un qualsiasi servlet container come Tomcat o Jetty.
Questa flessibilità è resa possibile da Spring che espone i servizi interni e gli strati di accesso ai dati sotto forma di Plugin. In pratica il sofware è sempre lo stesso ma nella Enterprise vengono pluggati gli EJB piuttosto che i servizi locali.

2.6.2 Installazione

Poiché nel periodo di tirocinio è stata utilizzata la versione Professional, vengono riportati i passi principali dell'installazione.
Per prima cosa andrà scompattato il file .zip della distribuzione in una directory a piacere.
Nella cartella selezionata, troveremo tutti i file necessari all'esecuzione, in particolare per la distribuzione professional si avrà anche un'installazione standard di Tomcat con una serie di webapp preconfigurate.

Da notare che Liferay utilizza per il proprio funzionamento un Data Base di appoggio quindi, prima dell'esecuzione vera e propria dovrà essere creato.
Nel sito di Liferay è presente una sezione contenente degli SQL Script che permettono la creazione o l'aggiornamento (nel caso in cui stiamo aggiornando il sistema ad una nuova versione) del Data Base. A questo punto andrà scelto quale DBMS utilizzare, infatti, sono presenti gli script dei maggiori DBMS tra cui: Db2, Firebird, Mysql, Oracle, Postgresql e Sql Server. Una volta lanciato lo script opportuno, per eseguire Liferay basterà fare doppio click sullo script ./bin/startup.bat (o ./bin/startup.sh se utilizziamo Linux come Sistema Operativo) e puntare il browser su seguente url:

```
http://localhost:8080/c/portal/layout
```

Supponendo che la porta su cui viene posto in ascolto Tomcat per le richieste http sia quella di default ossia la 8080.
Se tutto è andato a buon fine verrà visualizzata la home page come mostrato di seguito.

Figura 30: Home page Liferay

2.6.3 Configurazione WSRP Proxy Portlet

Come già detto Liferay permette la definizione di utenti e relativi permessi. Per prima cosa andrà effettuato il login cliccando sul link “Sign In” (nella pagina in alto a destra) tramite il quale verrà visualizzata la seguente schermata:

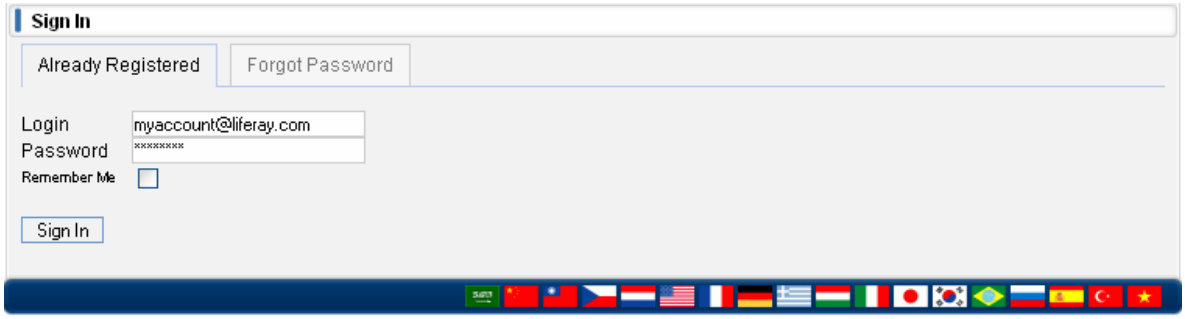

Figura 31: Login Liferay

In questa finestra andrà inserito il proprio Username e Password per accedere alla propria pagina del portale.

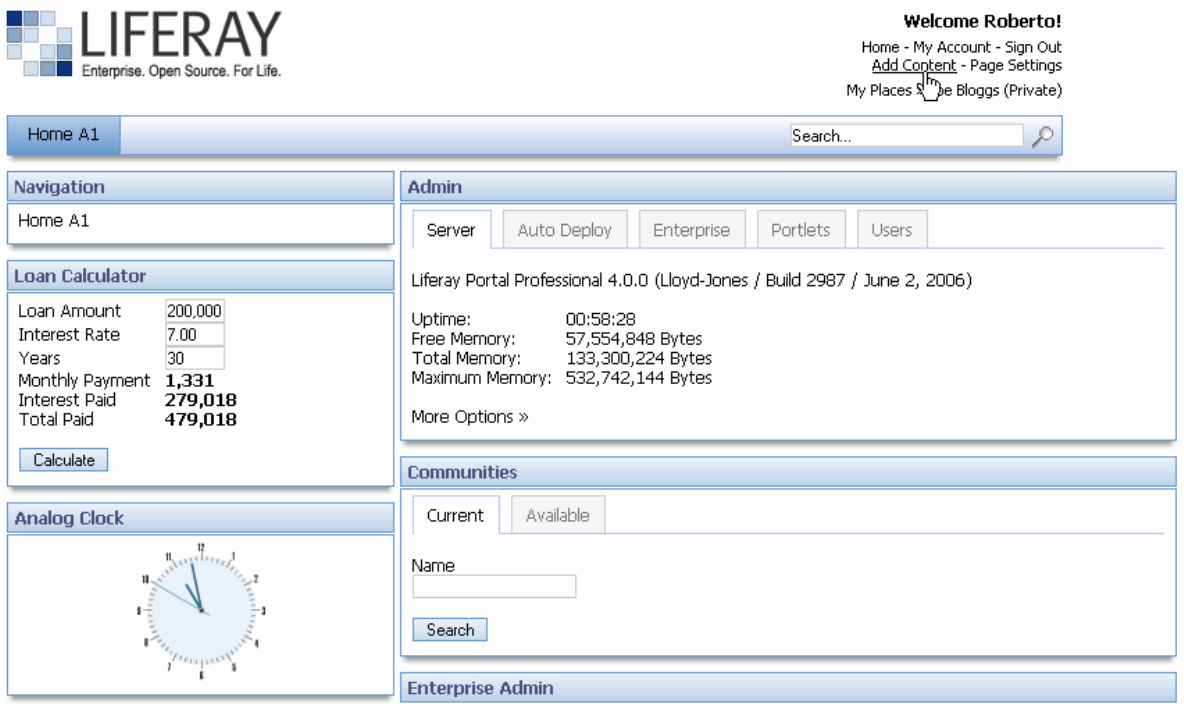

Figura 32: Pagina Portale

Come già detto in precedenza, Liferay mette a disposizione varie Portlet tra cui vi è la ProxyPortlet che permette tramite la propria configurazione di utilizzare Liferay come portale Consumer nell'ambito dello standard WSRP.
Per aggiungere una Portlet alla pagina del portale andrà selezionato il link "Add Content" posto in alto a destra della pagina come si nota dalla figura precedente.

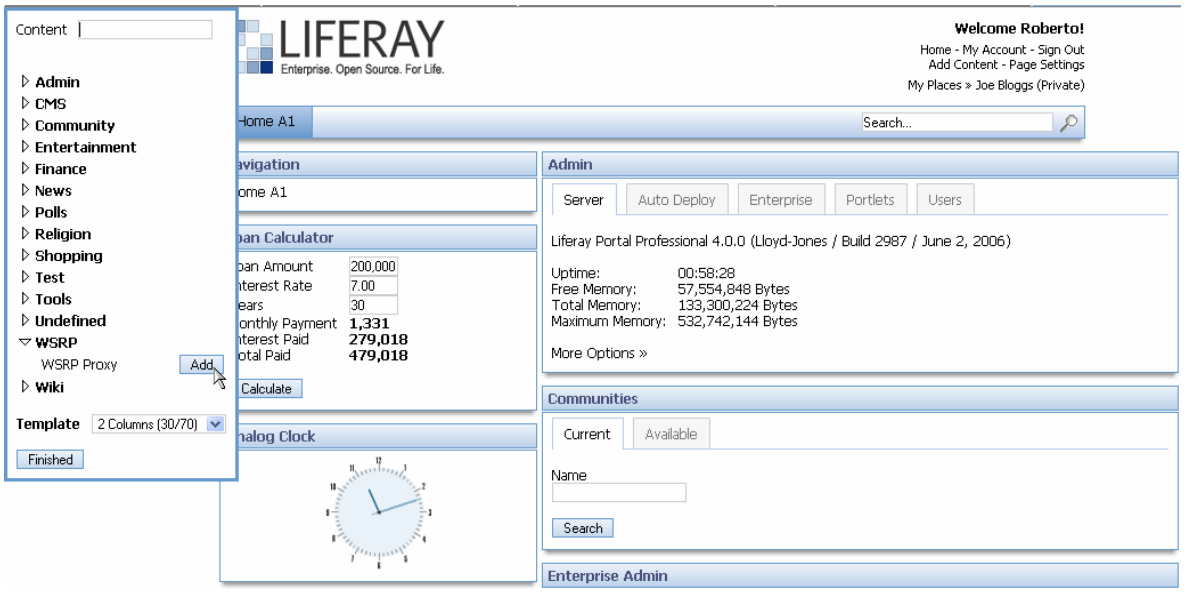

Figura 33: Aggiunta contenuto alla pagina del portale

Come si può notare, viene visualizzata una finestra di dialogo con l'elenco delle categorie presenti, all'interno di ogni categoria, ci sarà la lista delle Portlet associate. Per aggiungere alla pagina la WSRP Proxy Portlet basterà selezionare la categoria WSRP ed aggiungerla.
A questo punto sarà necessario configurarla per specificare:

- L'url del portale Producer
- Gli end point associati alle quattro interfacce richieste dallo standard WSRP
- La Portlet remota da visualizzare

Di seguito viene riportato un esempio di portale Producer che mette a disposizione due Portlet remote, selezionabili tramite un menù a tendina.

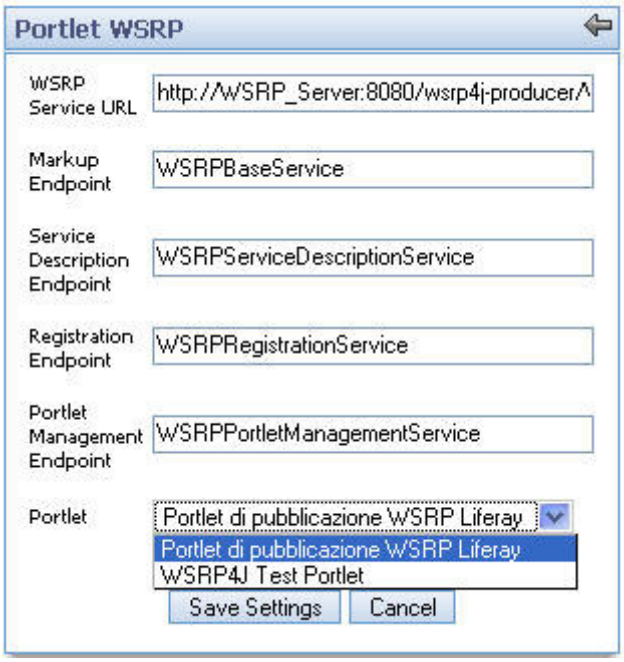

Figura 34: Configurazione Proxy Portlet

Di default è possibile avere una sola instanza di questa Portlet nella pagina poiché nel file `liferay-portlet.xml` (in cui vengono definite le Portlet presenti) l'attributo `instanceable` viene impostato a "false".

```
<portlet>
          <portlet-name>68</portlet-name>
          <struts-path>wsrp</struts-path>
          <restore-current-view>false</restore-current-view>
          <instanceable>false</instanceable>
          <private-request-attributes>
                false
          </private-request-attributes>
</portlet>
```

Per gli scopi dell'applicazione sviluppata è necessario poter avere più instanze della stessa Portlet (remota) nella pagina, quindi si è modificato il valore di questo attributo impostandolo a "true".
Precedentemente abbiamo visto la configurazione della ProxyPortlet, è possibile associare alla Portlet un portale Producer di default agendo sul file `portlet.xml` (presente nella directory portalDir\liferay\WEB-INF) come riportato di seguito.

```
<portlet>
    <portlet-name>68</portlet-name>
    <display-name>WSRP Proxy</display-name>
    <portlet-class>
        com.liferay.portlet.wsrp.WSRPProxyPortlet
    </portlet-class>
    <init-param>
                <name>edit-action</name>
                <value>/wsrp/edit</value>
    </init-param>
    <expiration-cache>0</expiration-cache>
    <supports>
          <mime-type>text/html</mime-type>
          <portlet-mode>edit</portlet-mode>
          <portlet-mode>help</portlet-mode>
    </supports>
    <resource-bundle>
          com.liferay.portlet.StrutsResourceBundle
    </resource-bundle>
    <portlet-preferences>
          <preference>
                <name>wsrp-service-url</name>
                <value>
                      http://WsrpServer:8080/wsrp4j-
                      producer/WSRP4Jproducer
                </value>
           </preference>
           <preference>
                <name>markup-endpoint</name>
                <value>WSRPBaseService</value>
           </preference>
           <preference>
                <name>
                      service-description-endpoint
                </name>
                <value>
                      WSRPServiceDescriptionService
                </value>
           </preference>
           <preference>
                <name>registration-endpoint</name>
                <value>
                      WSRPRegistrationService
                </value>
           </preference>
           <preference>
                <name>
                      portlet-management-endpoint
                </name>
```

```
                    <value>
                          WSRPPortletManagementService
                    </value>
               </preference>
               <preference>
                    <name>portlet-handle</name>
                    <value>20.0</value>
              </preference>
     </portlet-preferences>
</portlet>
```

Come si può notare le informazioni che vanno inserite sono le stesse a parte per la definizione della Portlet da visualizzare, infatti prima veniva scelta tramite un menù a tendina mentre, in questo caso viene specificata tramite il `PortletHandle` che identifica in modo univoco la Portlet lato "Producer".

2.6.4 Limitazioni

L'unica limitazione riscontrata nell'utilizzo di Liferay come portale Consumer sta nella gestione della sessione. Come detto in precedenza le Portlet hanno a disposizione due tipi di sessione (`APPLICATION_SCOPE` e `PORTLET_SCOPE`) con diversa visibilità.
Gli attributi in sessione a livello di Portlet, vengono comunque condivisi a livello di Applicazione, questo può portare a malfunzionamenti dell'applicazione. Nel capitolo dedicato alla descrizione delle applicazioni sviluppate verrà proposta la soluzione adottata.

3. Framework Utilizzati

3.1 Struts

3.1.1 Introduzione

Struts è un progetto open-source di Apache Jakarta Project , ed è ad oggi il framework largamente più adottato nella comunità degli sviluppatori Java. In questo capitolo ne verrà analizzata l'architettura e le componenti fondamentali cercando di evidenziarne pregi e difetti .

Un framework (come Struts) è una architettura generica che costituisce l'infrastruttura per lo sviluppo di applicazioni in una determinata area tecnologica. Detto in maniera molto semplice è un insieme di classi ed interfacce di base, che costituiscono l'infrastruttura di una applicazione.

In base a questa definizione è facile pensare erroneamente che utilizzare un framework equivalga ad usare una libreria di classi, mentre in realtà vi è una sostanziale differenza tra le due cose.

Una libreria di classi, quali ad esempio le classi di base del linguaggio Java, viene utilizzata dallo sviluppatore per svolgere determinate funzionalità; in questo caso il codice che viene scritto invoca il codice esistente per svolgere una certa funzione, ma il controllo del flusso applicativo rimane a nostro carico.

Adottare un framework significa invece attenersi ad un specifica architettura ovvero nella pratica estendere le classi del framework e/o implementarne le interfacce. In tal caso sono i componenti del framework che hanno la responsabilità di controllare il flusso dell'elaborazione.

Nel mondo dell'architettura del software un framework è considerato come una parte di software esistente nel quale inserire il proprio, in base al noto principio Hollywood "don't call us we call you". Il nostro codice applicativo non è direttamente invocato dall'intervento dell'utente sul sistema ma il flusso dell'elaborazione passa attraverso il codice del framework: sono le classi del framework che invocano il nostro codice applicativo e non viceversa come nel caso delle librerie di classi.

Come evidenziato precedentemente, utilizzare un framework significa implicitamente adottare una specifica architettura per la propria applicazione. Anche se questo può

sembrare vincolante è invece, nel caso di un framework valido (e vedremo quali criteri lo rendono tale), uno dei maggiori vantaggi.
All'inizio di un progetto infatti la scelta dell'architettura è uno dei momenti fondamentali che può determinare il successo o l'insuccesso del progetto stesso. A volte è una scelta che viene trascurata o sottovalutata, principalmente per un errato approccio allo sviluppo applicativo considerato esclusivamente come una attività di scrittura di codice, ma che produce effetti disastrosi se non ponderata attentamente. Utilizzare un framework maturo e già ampiamente testato significa attenersi ad una architettura che funziona e quindi significa iniziare un progetto da una base solida. Ciò porta inoltre ad un significativo risparmio di tempo e risorse in quanto lo sviluppatore non deve più preoccuparsi di realizzare componenti infrastrutturali ma può concentrarsi esclusivamente sullo sviluppo della logica di business che poi è il valore aggiunto della applicazione che si scrive.
Non è raro nello sviluppo di un progetto assistere alla riscrittura di componenti di base che già esistono e che sono stati già ampiamente testati; possiamo dire che uno dei vantaggi nell'utilizzo di un framework è che si viene aiutati a non "reinventare la ruota" come spesso purtroppo accade.
È chiaro che tutto ciò è vero quando si fa riferimento ad un framework giunto ad uno stadio di sviluppo maturo, già adottato da molti sviluppatori e quindi già ampiamente provato "sul campo".
Da un punto di vista pratico adottare un framework significa senz'altro ridurre i tempi di un progetto ed evitare errori nella fase di disegno in quanto si utilizza una infrastruttura realizzata secondo le best-practises dell'ambito tecnologico di riferimento.
È bene precisare che un framework non va confuso con un design-pattern. Un design-pattern è una strategia di soluzione di un problema comune, è qualcosa di concettuale che prescinde dall'implementazione tecnologica. Un framework è invece qualcosa di concreto, è un insieme di componenti che può essere usato per realizzare una applicazione; componenti che, quando il framework è ben strutturato, sono sviluppati secondo i design-pattern più diffusi nell'ambito specifico.
Struts implementa molti dei design-pattern J2EE di uso comune , ciò a garanzia di una architettura valida e ben sperimentata.

3.1.2 Vantaggi e svantaggi nell'utilizzo di un framework

In genere i vantaggi dell'utilizzo di un framework vanno ben oltre gli svantaggi, anzi si può affermare che quanto più il progetto sia di grosse dimensioni tanto più l'utilizzo di un framework è altamente consigliabile. È anche possibile sviluppare un proprio framework , anche se, a meno di casi del tutto particolari, è difficile pensare di scrivere in casa un framework che risolva problematiche diverse da quelle risolte da quelli già esistenti. Se questa fosse però la propria scelta conviene comunque studiare almeno l'architettura ed il codice, ove disponibile, dei framework più diffusi per conoscere le soluzioni adottate per i vari problemi e confrontarle con le proprie.
Di seguito vengono schematicamente riassunti alcuni dei principali vantaggi che si ottengono nell'adozione di un framework nello sviluppo di applicazioni J2EE.

- Disegno architetturale
  Un buon framework è fondato su un disegno architetturale valido, in quanto il suo codice è scritto in base alle best-practises della tecnologia in uso. Ciò conferisce al proprio progetto fondamenta solide dalle quali partire.
- Riduzione dei tempi di progetto
  Lo sviluppatore deve implementare esclusivamente la logica applicativa potendo risparmiare le energie e il tempo necessari alla scrittura di componenti infrastrutturali.
- Semplificazione dello sviluppo
  Un buon framework semplifica lo sviluppo applicativo perché fornisce tutta una serie di componenti che risolvono la gran parte dei compiti comuni a tutte le applicazioni web J2EE (controllo del flusso, logging, gestione messaggi di errore, custom tags per la presentation logic, internazionalizzazione, validazione dei dati, etc..)

Va precisato che ovviamente un framework non è una panacea o la soluzione di tutti i problemi. Adottarne uno che non si adatta al proprio problema può portare molti svantaggi, per questo la scelta di quello giusto per le proprie esigenze è di fondamentale importanza.
In genere è comunque sempre preferibile evitare framework poco generici, che impongono l'utilizzo di strumenti proprietari e che legano indissolubilmente la propria applicazione ad una specifica struttura.
Il framework deve fornire una base per lo sviluppo ma la logica applicativa sviluppata deve essere utilizzabile anche al di fuori della struttura del framework stesso.

Esistono molti framework per lo sviluppo di applicazioni web J2EE, sia open-source che prodotti commerciali. La scelta di un framework è importante per tutte le ragioni che abbiamo visto precedentemente e investe aspetti non solo tecnici ma anche economici. I criteri per la scelta sono molteplici ed è bene chiarire che non esiste il framework "ideale". Di seguito sono elencate alcune caratteristiche che nella maggior parte dei casi devono essere considerate valutazione.

- Maturità del progetto
  È sconsigliabile adottare un framework che sia in una fase iniziale di sviluppo e che sia poco adottato nella comunità degli sviluppatori e quindi poco testato sul campo in progetti reali. Meglio indirizzarsi verso progetti già stabili e sperimentati.
- Documentazione
  Va sempre verificato che la documentazione sia ricca e ben fatta. Questo facilita la risoluzione dei problemi che si incontrano nella realizzazione dell'applicazione e la comprensione del suo funzionamento.
- Validità del disegno architetturale
  Proprio perché la scelta di un framework influisce sull'architettura applicativa è bene verificare che sia disegnato correttamente e quindi che siano adottati i design-pattern e le best-practises della tecnologia di riferimento.
- Adozione degli standard
  Un framework deve essere fondato sui componenti standard della tecnologia di riferimento. Nel nostro caso sulle API che costituiscono la J2EE. Quanto più un framework impone soluzioni proprietarie, l'uso di specifici tool di sviluppo o un modello troppo indirizzato ad uno specifico caso applicativo tanto più va evitato.
- Estensibilità
  Deve essere possibile estenderne le funzionalità per adattarlo alle proprie esigenze.

Come si vedrà Struts rispetta i criteri sopra elencati e se utilizzato seguendo alcune principali linee guida consente di realizzare applicazioni ben strutturate, assolutamente conformi agli standard J2EE e la cui logica applicativa è riutilizzabile anche in altri contesti.

Non a caso Jakarta Struts è il framework in assoluto più diffuso a livello mondiale nello sviluppo di applicazioni J2EE. Esistono molteplici esempi di casi reali di progetti

di successo sviluppati con Struts il che sicuramente è una garanzia per coloro che volessero adottarlo in un nuovo progetto senza averne esperienza diretta. Inoltre essendo un progetto open-source lo si può adottare senza gravare sui costi di progetto e si ha a disposizione tutto il codice sorgente.

3.1.3 Il pattern MVC (Model-View-Controller)

Jakarta Struts è un MVC web application framework, ovvero è un framework per lo sviluppo di applicazioni web J2EE basato sul pattern Model-View-Controller. Per chiarire meglio questa definizione introduciamo brevemente il pattern MVC. Uno dei principali requisiti di qualsiasi applicazione web è quello di definire un modello applicativo che consenta di disaccoppiare i diversi componenti dell'applicazione in base al loro ruolo nell'architettura per ottenere vantaggi in termini di riusabilità e manutenibilità.
Esempio tipico di questo problema è l'utilizzo nello sviluppo di una applicazione web J2EE del modello applicativo che nella letteratura è spesso indicato come "JSP Model 1". In base a questo modello l'applicazione è costruita secondo una logica "JSP centric" in base alla quale presentation , control e business logic dell'applicazione sono tutti a carico delle pagine JSP. Il web browser accede direttamente alle pagine JSP dell'applicazione che al loro interno contengono logica applicativa e logica di controllo del flusso; all'interno delle pagine JSP sono cablati i riferimenti alle viste successive in base alla logica di flusso dell'applicazione che è codificata all'interno della pagina stessa. In questo modello non esiste un controllo centralizzato del flusso ma ogni vista si fa carico della selezione delle viste ad essa collegate. Un modello di questo tipo , come suggerito dalla stessa Sun, va evitato se non per lo sviluppo di piccoli prototipi o applicazioni molto semplici e dal flusso elaborativo praticamente statico, in quanto porta a scrivere applicazioni difficilmente gestibili al crescere della complessità e non riusabili nei suoi componenti.
Quando l'applicazione cresce in complessità non è pensabile svilupparla seguendo un simile approccio. Il pattern MVC è una implementazione di quello che va sotto il nome di "Model 2"; il Model 2 introduce il concetto di controllo centralizzato dell'applicazione, implementato da una servlet di controllo che gestisce tutte le richieste e le soddisfa delegando l'elaborazione a opportune classi Java.

In questo modello i ruoli di presentation, control e business logic vengono affidati a componenti diversi e sono tra di loro disaccoppiati, con evidenti vantaggi in termini di riusabilità, manutenibilità, estensibilità e modularità.
In un'applicazione costruita secondo il pattern MVC si possono quindi individuare tre livelli logici ben distinti che molto schematicamente svolgono i seguenti compiti:

- *Controller*: determina il modo in cui l'applicazione risponde agli input dell'utente. Esamina le richieste dei client, estrae i parametri della richiesta e li convalida, si interfaccia con lo strato di business logic dell'applicazione. Sceglie la successiva vista da fornire all'utente al termine dell'elaborazione.
- *Model*: contiene i dati visualizzati dalle viste; è ciò che viene elaborato e successivamente presentato all'utente.
- *View*: visualizza all'utente i dati contenuti nel model. È la rappresentazione dello stato corrente del Model.

3.1.4 L'implementazione di Struts del design pattern MVC

Come già detto Struts è un MVC web application framework, ovvero è un insieme di classi ed interfacce che costituiscono l'infrastruttura per costruire web application J2EE conformi al design pattern MVC.
I componenti fondamentali di Struts sono:

- *ActionServlet*: è la servlet di controllo centralizzata che gestisce tutte le richieste dell'applicazione.
- *struts-config.xml* è il file XML di configurazione di tutta l'applicazione. In questo file vengono definiti gli elementi dell'applicazione e le loro associazioni.
- *Action*: le Action sono le classi alle quali la *ActionServlet* delega l'elaborazione della richiesta.
- *ActionMapping*: contiene gli oggetti associati ad una Action nello *struts-config.xml* come ad esempio gli *ActionForward*.
- *ActionForm*: gli ActionForm sono classi contenitori di dati. Vengono popolati automaticamente dal framework con i dati contenuti nelle request http.
- *ActionForward*: contengono i path ai quali la servlet di Struts inoltra il flusso elaborativo in base alla logica dell'applicazione.

- *Custom-tags*: Struts fornisce una serie di librerie di tag per assolvere a molti dei più comuni compiti delle pagine JSP.

La *ActionServlet* è la servlet di controllo di Struts. Gestisce tutte le richieste client e smista il flusso applicativo in base alla logica configurata. Si potrebbe definire come la "spina dorsale" di una applicazione costruita su Struts. Tutta la configurazione dell'applicazione è contenuta nello *struts-config.xml*. Questo file XML viene letto in fase di start-up dell'applicazione dalla *ActionServlet* e definisce le associazioni tra i vari elementi di Struts. Nello *struts-config.xml* sono ad esempio definite le associazioni tra i path delle richieste http e le classi *Action* associate alla richieste stesse.

Le associazioni tra le *Action* e gli *ActionForm*, che vengono automaticamente popolati dal framework con i dati della richiesta ad essi associata e passati in input alla *Action*. Contiene inoltre l'associazione tra la *Action* e le *ActionForward* , ovvero i path configurati nello *struts-config.xml* ai quali la *ActionServlet* redirigerà il flusso applicativo al termine della elaborazione della Action.

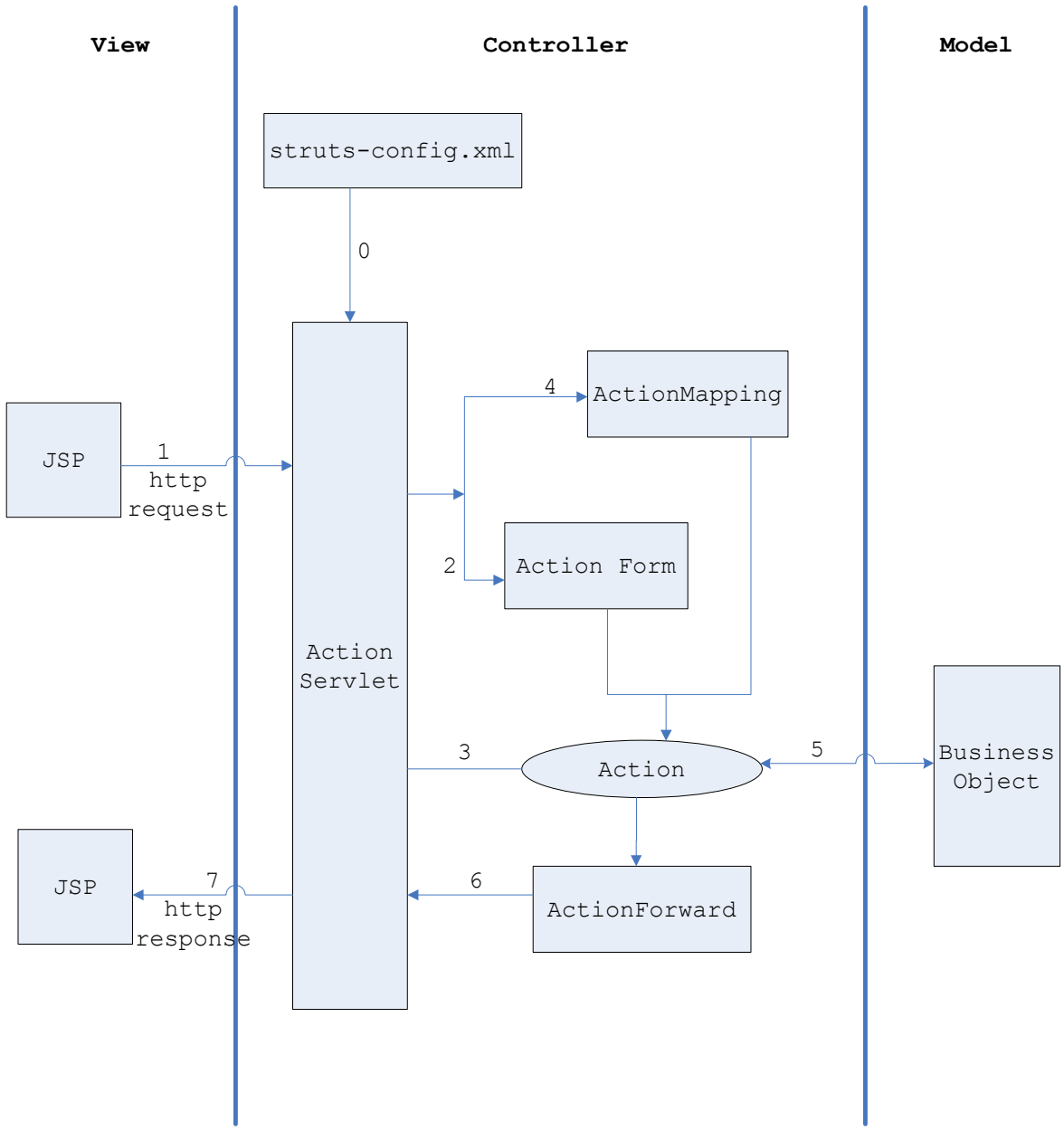

Figura 35: Implementazione di Struts del pattern MVC

Nella figura è rappresentato schematicamente il flusso dell'elaborazione secondo la logica di Struts:

1. Il client invia una richiesta http (1)
2. La richiesta viene ricevuta dalla servlet di Struts che provvede a popolare l'*ActionForm* associato alla richiesta con i dati della request (2) e l'*ActionMapping* con gli oggetti relativi alla *Action* associata alla richiesta (4) .

Tutti i dati di configurazione sono stati letti in fase di start-up dell'applicazione (0) dal file XML *struts-config.xml*.

3. La *ActionServlet* delega l'elaborazione della richiesta alla *Action* associata al path della richiesta (3) passandole in input request e response http e l'*ActionForm* e l'*ActionMapping* precedentemente valorizzati.
4. La *Action* si interfaccia con lo strato di business che implementa la logica applicativa. Al termine dell'elaborazione restituisce alla *ActionServlet* un *ActionForward* (6) contenente l'informazione del path della vista da fornire all'utente.
5. La *ActionServlet* esegue il forward alla vista specificata nell'*ActionForward* (7).

Ovviamente il flusso di operazioni elencato non è completo ma fornisce una indicazione di base su come viene gestito il flusso elaborativo in una applicazione sviluppata con Struts. Anche se il flusso descritto può apparire complesso, è in realtà di comprensione piuttosto semplice.

3.1.5 Caratteristiche di base di Jakarta Struts

Dalla discussione dei precedenti paragrafi possiamo quindi individuare alcune caratteristiche peculiari di Struts, che sono poi comuni anche ad altri MVC framework.

- Esiste una sola servlet di controllo centralizzata. Tutte le richieste sono mappate sulla *ActionServlet* nel `web.xml` dell'applicazione. Ciò consente di avere un unico punto di gestione del flusso applicativo e quindi permette di implementare in modo univoco e centralizzato funzioni quali sicurezza, logging, filtri etc.
- Le viste dell'applicazione non contengono al loro interno il riferimento al flusso dell'applicazione e non contengono logica applicativa. I livelli logici dell'applicazione sono disaccoppiati.
- Le viste sono identificate con nomi logici definiti nel file di configurazione *struts-config.xml*. Nel codice Java non è presente alcun riferimento a nomi di pagine JSP il che rende molto più semplice variare il flusso applicativo.

- Tutta la configurazione dell'applicazione è scritta esternamente in un file XML il che consente di modificare le associazioni tra le richieste http e le classi ad essa associate in modo molto semplice.

C'è da osservare che tutti i componenti di Struts descritti nel paragrafo precedente fanno parte del livello di controllo tranne i custom-tags che fanno parte della view. Le stesse *Action* che sono le classi alle quali la *ActionServlet* delega l'elaborazione delle richieste sono componenti del controller e non del model.
Ciò per sottolineare che Struts è un framework model-neutral, ovvero che implementa esclusivamente i livelli di controller e view.
Utilizzando Struts è possibile realizzare il livello di business logic in base alle proprie scelte; con semplici classi Java quindi implementando la logica applicativa nel web-container, o ricorrendo agli EJB quindi sfruttando i servizi del EJB-container.
Da notare che Struts indirizza lo sviluppo per lo strato di controller dell'applicazione e fornisce un aiuto mediante i suoi custom-tags nello sviluppo della view. Non interviene invece nella scelta implementativa dello strato di Model. È quindi buona norma non inserire logica applicativa nelle *Action* né tantomeno nelle pagine JSP. La logica applicativa è parte del Model indipendentemente da come questo venga realizzato, semplici classi Java, DAO, EJB, o altro.
Il Model non deve avere conoscenza di Struts, ovvero non bisogna mai passare al model riferimenti ad oggetti facenti parte del framework. Così facendo si renderebbe il proprio strato di business-logic dipendente da Struts e quindi inutilizzabile in altri contesti.
Ogni livello dell'applicazione deve svolgere il compito a cui è destinato.

3.1.6 Principali vantaggi nell'uso di Jakarta Struts

Da quanto esposto fin ora si possono evidenziare alcune delle caratteristiche di una applicazione sviluppata con Jakarta Struts e alcuni vantaggi conseguenti al suo utilizzo:

- Modularità e Ricusabilità
  I diversi ruoli dell'applicazione sono affidati a diversi componenti. Ciò consente di sviluppare codice modulare e più facilmente riutilizzabile.

- Manutenibilità
  L'applicazione è costiuita da livelli logici ben distinti. Una modifica in uno dei livelli non comporta modifiche negli altri. Ad esempio una modifica ad una pagina JSP non ha impatto sulla logica di controllo o sulla logica di business, cosa che avveniva nel JSP Model 1.
- Rapidità di sviluppo
  A differenza di quanto avveniva utilizzando il JSP Model 1, è possibile sviluppare in parallelo le varie parti dell'applicazione, view (JSP/HTML) e logica di business (Java) sfruttando al meglio le conoscenze dei componenti del team di sviluppo. Si possono utilizzare sviluppatori meno esperti e anche con poche conoscenze di Java per la realizzazione delle view, permettendo agli sviluppatori Java più esperti di concentrarsi sulla realizzazione della business logic.

3.1.7 La ActionServlet

La `org.apache.struts.action.ActionServlet` è la servlet di controllo di Struts. Come già accennato nel precedente paragrafo è la servlet che gestisce tutte le richieste http che provengono dai client e indirizza il flusso applicativo in base alla configurazione presente nel file XML `struts-config.xml`. Come è ovvio la `ActionServlet` estende la `javax.servlet.http.HttpServlet`; i suoi metodi `doGet()` e `doPost()` chiamano entrambi un metodo `process()` che esegue quindi l'elaborazione sia in caso di richieste di tipo GET che di tipo POST. Di seguito è riportato il metodo `doGet()`, il `doPost()` è identico:

```
public void doGet(HttpServletRequest request,
HttpServletresponse response) throws
IOException,ServletException {
     process(request,response);
}
```

La ActionServlet esegue l'elaborazione che schematicamente comprende i seguenti step:

1. I metodi doGet() e doPost() invocano il metodo process() della ActionServlet
2. Nel metodo process() la ActionServlet ottiene l'istanza del RequestProcessor, configurato per l'applicazione nel tag <controller> dello struts.config.xml, e ne esegue il metodo process().
3. Nel metodo process() del RequestProcessor viene eseguita l'elaborazione vera e propria, ed in output al metodo viene fornito un oggetto ActionForward che consente alla ActionServlet di inoltrare l'elaborazione in base alla configurazione presente nello struts-config.xml.

La ActionServlet viene configurata, come ogni servlet, nel web.xml. I parametri di inizializzazione sono molti e per un elenco completo si rimanda al sito ufficiale di Struts (http://jakarta.apache.org/struts/). Di seguito è riportato un blocco <servlet> di configurazione standard della ActionServlet nel quale è settato ad 1 il parametro <load-on-startup> in base al quale il container instanzia la ActionServlet allo start-up della web-application e ne invoca il metodo init(). Inoltre mediante il parametro config è specificata la posizione del file XML di configurazione dell'applicazione. Il parametro debug abilita il debugging dell'applicazione. Mediante il blocco <servlet-mapping> si specifica che tutte le richieste con path terminante in .do vengono mappate sulla servlet di controllo di Struts.
Configurazione standard della ActionServlet:

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet
    </servlet-class>
    <init-param>
       <param-name>config</param-name>
       <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
       <param-name>debug</param-name>
```

```
        <param-value>2</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Con questa configurazione di base è già possibile utilizzare la `ActionServlet` come servlet di controllo della propria applicazione.

3.1.8 La classe Action

La classe `Action` è l'elemento fondamentale del controller di Struts in quanto per ogni funzione realizzata con Struts bisogna creare una propria classe che la estende e ne implementa il metodo `execute()` che è fatto come segue:

```
public ActionForward execute(ActionMapping mapping, ActionForm
form,HttpServletRequest request, HttpServletResponse response)
    throws Exception{

  //acquisizione form
  MyForm myForm = (MyForm)form;
  //acquisizione parametri dal form
  String param1 = myForm.getParam1();

  //business logic
  ...
  //fine business logic
  //inoltro dell'elaborazione
  return mapping.findForward("ok");
}
```

Il metodo `execute()` riceve in input request e response http, un'istanza dell'oggetto `ActionForm` prima descritto, e un oggetto `ActionMapping` che contiene le

informazioni configurate nell'elemento `<action>` tra le quali i forward, ovvero i percorsi a cui inoltrare in uscita l'elaborazione. Restituisce un oggetto `ActionForward` che contiene il path di inoltro dell'elaborazione. È nel metodo `execute()` della propria `Action` che lo sviluppatore inserisce il proprio codice di elaborazione della richiesta per la funzione specifica. Bisogna subito sottolineare due aspetti fondamentali riguardo alle `Action`:

- Le `Action` vengono gestite esattamente come delle `Servlet`. Ovvero il loro funzionamento è basato sulla stessa logica multithread delle `servlet` quindi sono soggette a tutti i problemi comunemente noti nelle `servlet`. Il codice scritto nelle `Action` deve essere thread-safe per un corretto funzionamento delle stesse.
- Le `Action` fanno parte del Controller e non del Model. La logica applicativa non deve essere scritta nella `Action`, ma questa deve delegare allo strato di Model l'elaborazione della business-logic.

In base a quanto detto una `Action` dovrebbe:

- Acquisire i dati della request dal form
- Delegare l'elaborazione della business-logic alle classi del Model
- Acquisire i risultati dell'elaborazione e prepararli per la vista da inviare all'utente mettendoli nello scope opportuno (se necessario).
- Inoltrare il flusso elaborativo in base alla logica applicativa.

Le `Action` costituiscono quindi il "ponte" applicativo tra lo strato di Controller e di Model di un applicazione scritta con Struts ed hanno un ruolo fondamentale perché sono le classi che lo sviluppatore scrive continuamente nello sviluppo di una applicazione Struts.

3.1.9 La classe ActionForm

Abbiamo già visto che i form (come sono chiamate nella terminologia Struts le classi che estendono la `org.apache.struts.action.ActionForm`) sono

sostanzialmente dei bean contenenti dati, che il framework popola automaticamente con i dati della request svincolando lo sviluppatore dal doverlo fare con proprio codice applicativo. Associando ad un path un oggetto di una classe che estende la `org.apache.struts.action.ActionForm`, definito mediante il tag `<form-bean>`, è possibile fornire al metodo `execute()` della `Action` un oggetto contenente tutti i dati inseriti in un form HTML, con la possibilità di validarli prima che gli stessi giungano alla `Action` stessa come visto in precedenza. Ritorneremo comunque a parlare dei form quando analizzeremo lo strato di View di Struts.

In ogni applicazione web la view ha due compiti fondamentali: presentare all'utente i dati frutto dell'elaborazione eseguita e consentire all'utente l'immissione di dati elaborare.

Normalmente in una applicazione J2EE tradizionale i dati da visualizzare sono contenuti negli attributi di un JavaBean memorizzato nell'appropriato scope al quale la pagina fa riferimento.

L'inserimento di dati è realizzata mediante un form HTML contenuto nella pagina JSP e i vari input type che consentono l'invio nella request di dati che saranno reperiti dai componenti del controller e passati allo strato di business logic.

Il reperimento dei dati dalla request, la loro validazione e il popolamento con essi degli oggetti del model è a carico dello sviluppatore che dovrà scrivere codice per reperire i dati dalla request , instanziare un oggetto di una classe opportuna per memorizzarli , validarli e fornirli allo strato di business-logic.

Gli `ActionForm` di Struts consentono di automatizzare in parte questo compito che è uno dei più frequenti e ripetitivi in una applicazione web J2EE.

Un `ActionForm` è una classe che estende la `org.apache.struts.actions.ActionForm` e che viene utilizzata per acquisire i dati da un form HTML e fornirli ad una classe `Action`.

In pratica il controller di Struts provvede a popolare in automatico gli attributi di un `ActionForm` associato ad una determinata `Action` con i dati inviati nella request, associandoli in base alla corrispondenza nome-parametro nome-attributo, e a passare l'istanza dell'`ActionForm` così valorizzata al metodo `execute()` della `Action` stessa.

Gli `ActionForm` costituiscono quindi una sorta di buffer nel quale vengono posti i dati digitati in un form HTML, che possono così essere ripresentati facilmente all'utente in caso di errori nella validazione. Allo stesso tempo costituisce per così dire

un “firewall” per l’applicazione in quanto facilita il controllo dei dati prima che questi vengano passati allo strato di logica.
Gli `ActionForm` sono del tutto equivalenti a dei JavaBean e possono essere quindi utilizzati per contenere i dati restituiti dallo strato di business logic e da presentare all’utente oltre che a essere usati come classi corrispondenti ad un form HTML come il loro nome suggerisce.
Di seguito è riportato il codice di esempio di un `ActionForm` corrispondente ad un classico form di immissione di username e password:

```
public class LoginForm extends ActionForm {
  private String password = null;
  private String username = null;
  public String getPassword() {
    return this.password;
  }
  public void setPassword(String password) {
    this.password = password;
  }
  public String getUsername() {
    return this.username;
  }
  public void setUsername(String username) {
    this.username = username;
  }
}
```

Come si vede la struttura è esattamente quella di un JavaBean a parte l’estensione della classe `ActionForm`. Gli `ActionForm` estendono la `org.apache.struts.actions.ActionForm` , ed hanno attributi privati e corrispondenti metodi get e set pubblici. In più hanno due metodi particolari: `reset()` e `validate()` che ne caratterizzano il comportamento.
Il metodo `reset()` viene chiamato dal controller dopo che questo ha creato o reperito dallo scope opportuno l’istanza dell’`ActionForm`. Può quindi essere usato per inizializzare gli attributi del form ad un valore stabilito.
Il metodo `validate()` viene chiamato dal controller dopo la valorizzazione degli attributi dell’`ActionForm` qualora nello `struts-config.xml` sia stato valorizzato a

true l'attributo validate del tag <action> nel quale si fa riferimento all'ActionForm in questione.
Nel metodo validate() va inserito il codice per la validazione formale dei dati del form. Ciò garantisce di avere un punto standard nel codice nel quale questa validazione viene effettuata e che i dati che arrivano alla Action siano già stati formalmente validati.

3.1.10 Gli ActionErrors

Il metodo validate() di un ActionForm è il punto nel quale viene inserito il codice di validazione formale dei dati immessi dall'utente in un form HTML. La signature del metodo è la seguente:

```
public ActionErrors validate(ActionMappings mapping,
      HttpServletRequest request)
```

Il tipo di ritorno del metodo è un oggetto della classe ActionErrors che è un contenitore di oggetti della classe org.apache.struts.action.ActionError . Ogni oggetto della classe ActionError rappresenta un errore verificatosi nella validazione dei dati. Qualora durante la validazione si verifichino degli errori, per ciascuno di essi viene creata una istanza di un oggetto ActionError e aggiunta all'oggetto ActionErrors restituito dal metodo. Se il controller verifica che l'oggetto ActionErrors in uscita al metodo validate() non è nullo, non trasferisce il controllo al metodo execute() della classe Action associata alla richiesta in elaborazione ma bensì alla pagina JSP il cui path è configurato nell'attributo input del tag <action> corrispondente.
Con un opportuno custom tag <html:errors> posto nella pagina stessa sarà possibile visualizzare i messaggi di errore associati agli errori verificatisi senza scrittura di codice aggiuntivo.
Il messaggio di errore viene reperito automaticamente dal framework dal resouce bundle dell'applicazione; la chiave del messaggio è fornita nel costruttore dell'oggetto ActionError associato all'errore in questione. Di seguito è riportato l'esempio di un

metodo che esegue la validazione della username e della password immessi nel form html associato all'`ActionForm` visto in precedenza:

```
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
  ActionErrors errors = new ActionErrors();
  if ((username == null) || (username.length() < 1))
     errors.add ("username", new
                 ActionError("errore.username.obbligatorio"));
  if ((password == null) || (password.length() < 1))
     errors.add("password",new
                 ActionError("errore.password.obbligatoria"));
  return errors;
}
```

Le label `errore.username.obbligatorio` e `errore.password.obbligatorio` sono le chiavi alle quali sono associati i messaggi di errore nel resource bundle dell'applicazione. Con il metodo `validate()`, le classi `ActionErrors` ed `ActionError` ed il tag `<html:errors>` il framework fornisce quindi un automatismo standard per la gestione della validazione dei dati immessi nella view dell'applicazione e per la visualizzazione dei messaggi di errore.

3.1.11 Le librerie di custom-tag di Struts

Per la costruzione delle viste dell'applicazione , Struts mette a disposizione alcune librerie di tag che svolgono alcuni dei compiti più frequenti in una pagina JSP. Le librerie di tag sono raggruppate logicamente in base al tipo di funzione svolta e sono:

- html Tag per la generazione di form HTML che interagiscono con gli `ActionForm` e degli altri elementi HTML di una pagina JSP
- bean Tag usati per accedere a proprietà di JavaBeans e per creare instanze di bean
- logic Tag usati per logica condizionale, iterazioni e controllo di flusso
- nested Tag che estendono le funzionalità dei tag base

La libreria html comprende una serie di tag che consentono di generare in automatico ed in maniera standard tag html. In particolare per ciò che riguarda i form HTML questi tag sono strettamente collegati alla discussione fatta sugli `ActionForm`. Il tag `<html:form>` è senza dubbio uno dei più comuni in quanto consente di definire un form HTML. Vi sono poi tag per ciascuno degli input type html. Nell'esempio seguente viene definito un form la cui action è il path "/login". Questo corrisponderà al path configurato in un blocco `<action></action>` dello `struts-config.xml`. Il form contiene due input uno di tipo text ed uno di tipo password i cui valori corrispondono agli attributi username e password dell'`ActionForm` associato al form in questione. Il valore dell'attributo `property` dei tag `<html:text>` e `<html:password>` corrisponde al nome dell'attributo dell'`ActionForm` associato al form. Ciò significa che il framework valorizzerà in automatico gli attributi username e password dell'`ActionForm` con i valori corrispondenti immessi nel form. Il tag `<html:submit>` genera il codice html di un button di tipo submit.

```
<html:form action="/login" >
   <TABLE>
     <TR>
       <TH>Username:</TH>
       <TD><html:text property="username"/></TD>
     </TR>
     <TR>
       <TH>Password:</TH>
       <TD><html:password property="password"/></TD>
     </TR>
     <TR>
       <TD><html:submit/></TD>
     </TR>
   </TABLE>
</html:form>
```

Nella libreria logic esistono numerosi tag per eseguire logica condizionale quali `<logic:empty>` `<logic:notEmpty>` `<logic:equal>` `<logic:notEqual>` `<logic:greaterThan>` ed altri., tag per eseguire iterazioni su array e collection quali `<logic:iterate>`, e tag per eseguire controllo di flusso come il `<logic:redirect>`.

Nella libreria bean sono presenti tag per la definizione di variabili di scripting utilizzabili all'interno della pagina quali `<bean:define>` , tag per la scrittura in output del valore di attributi di un `<bean: write>` e così via.

3.1.12 Internazionalizzazione e Java

Internazionalizzare un'applicazione significa predisporla a supportare lingue diverse e regioni geografiche diverse senza dover intervenire con modifiche all'architettura. Le specifiche standard riguardanti l'internazionalizzazione enunciano che un'applicazione internazionalizzata deve rispondere ai seguenti requisiti:

- Lingue diverse devono essere supportate senza modifiche al codice
- Testi e immagini devono essere memorizzati esternamente al codice
- Date, numeri , valute devono essere correttamente formattate in base alle regole della regione geografica nella quale l'applicazione è eseguita
- Sono supportati caratteri non standard
- L'applicazione si adatta rapidamente a supportare una nuova lingua o una nuova regione

Il problema non è di poco conto poiché gli elementi che sono affetti da cambiamenti al variare della lingua e/o della regione sono molteplici all'interno di una applicazione. Spesso l'errore è di focalizzarsi solo sul primo punto che è il più evidente; si pensa infatti che internazionalizzare un'applicazione significhi rendere l'applicazione multilingua in quanto è immediato pensare alle label presenti nelle interfacce grafiche come unico elemento critico. In realtà anche date, numeri , valute sono elementi che variano al variare della lingua e del paese e quindi sono da tenere in conto. È sicuramente molto importante sapere in fase di progettazione dell'applicazione che l'applicazione deve essere internazionalizzata poiché un intervento a posteriori sarebbe sicuramente estremamente oneroso.
Internazionalizzare una applicazione quindi dal punto di vista tecnico si traduce nello scrivere del codice che si comporti correttamente qualora la lingua o il paese cambino ma senza che sia necessario apportare modifiche allo stesso. È ovvio che tutta una serie di risorse dell'applicazione saranno diverse a seconda della lingua. La label

“benvenuto” nella pagina iniziale dell’applicazione visitata da un utente di lingua italiana dovrà apparire come “welcome” per un utente di lingua inglese. Per far si che un’applicazione internazionalizzata abbia un comportamento simile è necessario localizzarla.
Il processo di adattare un’applicazione internazionalizzata ad una specifica lingua e regione si definisce quindi localizzazione.
Per la localizzazione di una applicazione predisposta all’internazionalizzazione Java fornisce una serie di classi che consentono una gestione del problema estremamente flessibile ed elegante.
La localizzazione si basa sull’utilizzo di due classi fondamentali: `java.util.Locale` e `java.util.ResourceBundle`, e su una serie di altre classi che fanno riferimento a queste.
La classe `java.util.Locale` è quella che consente all’applicazione di conoscere in quale lingua e in quale paese viene eseguita. È un identificatore per ogni combinazione di lingua e regione geografica, una istanza di questa classe identifica una combinazione language/country. Per instanziare un oggetto della classe `Locale` bisogna passare al costruttore un identificativo della lingua e della regione. Ad esempio il locale che identifica la lingua italiano e la regione Italia va costruito nel seguente modo:

```
Locale locale = new Locale("it", "IT").
```

Le classi del linguaggio che cambiano il proprio comportamento in funzione del locale corrente si dicono locale-sensitive. Rientrano ad esempio. in questa categoria le classi per la formattazione delle date e degli importi che vengono utilizzate nella scrittura di una applicazione internazionalizzata. Mediante queste classi è possibile scrivere codice che ha comportamento diverso in base al locale corrente e che quindi è predisposto ad essere eseguito per lingue diverse e in paesi diversi.
La classe `java.util.ResourceBundle` è invece un contenitore di oggetti cosiddetti locale-specific ovvero oggetti che assumono valori differenti in base al locale. Ad esempio una risorsa locale-specific è sicuramente la stringa che rappresenta una label di una interfaccia grafica; l’applicazione internazionalizzata reperirà la stringa dal `ResourceBundle` specifico per il locale corrente e quindi sarà predisposta al cambiamento della lingua o della regione. In realtà la classe

`java.util.ResourceBundle` è una classe astratta. Una implementazione concreta è la `java.util.PropertyResourceBundle` che gestisce le risorse specifiche per un dato locale memorizzandole in file di properties. I file di properties sono file di testo contenenti un elenco di coppie chiave/valore, dove la chiave è un identificativo associato ad una determinata stringa e valore è la stringa stessa per quel determinato locale.

3.1.13 I componenti di Struts per la gestione dell'internazionalizzazione

Struts fornisce supporto all'internazionalizzazione essenzialmente per ciò che riguarda il reperimento di testo e immagini localizzate. Per gli altri aspetti, quali formattazione di date, importi etc. bisogna fare ricorso alle classi Java.
Struts gestisce l'internazionalizzazione fornendo gli strumenti per reperire risorse localizzate da opportuni resource bundle in base al locale corrente.
In una web-application, e quindi anche in quelle costruite con Struts, è possibile reperire l'informazione relativa al locale dell'utente mediante il metodo `getLocale()` dell'oggetto `HttpServletRequest`.
Infatti l'informazione del locale utilizzato dall'utente è inviata al container in ogni request; Struts come default memorizza questa informazione nella sessione, ma è possibile variare questo comportamento impostando il valore dell'attributo locale del tag `<controller…/>` nello `struts-config.xml`. Il valore di default è false.
Con l'informazione del locale presente in sessione l'applicazione è quindi in grado di reperire dal resource bundle appropriato la risorsa localizzata.
Per la gestione dei resource bundle in Struts viene usata la classe `org.apache.struts.util.MessageResources` che segue la stessa logica della `java.util.ResourceBundle` arricchendola con alcune funzioni di utilità.
Anche la classe `org.apache.struts.util.MessageResources` è una classe astratta, e la sua concreta implementazione è fornita dalla classe `org.apache.struts.util.PropertyMessageResources` che consente di leggere stringhe localizzate alle quali è associata una chiave reperendole da file di properties, esattamente come fa la `java.util.PropertyResourceBundle`.

Il primo elemento da definire quindi per localizzare una applicazione Struts sono proprio i resource bundle ovvero i file di properties contenenti la lista in formato nome/valore di tutte le label dell'applicazione. Esisterà un file di properties per la lingua di default della propria applicazione chiamato ad esempio `ApplicationResources.properties` che avrà una serie di elementi del tipo :

```
button.aggiorna=Aggiorna
button.conferma=Conferma
button.elimina=Elimina
button.inserisci=Inserisci
button.salva=Salva
```

Dovranno poi essere definiti tanti altri altri file di properties per tutte le combinazioni lingua/regione per le quali si vuole che l'applicazione sia predisposta. Ad esempio
`ApplicationResource_en_En.properties` per inglese/regnoUnito
`ApplicationResource_en_US.properties` per inglese/Stati Uniti
`ApplicationResources_fr_FR.properties` per francese/Francia
e così via. Il file `ApplicationResource_en_En.properties` sarà del tipo :

```
button.aggiorna=Update
button.conferma=Confirm
button.elimina=Delete
button.inserisci=Insert
button.modifica=Modifica
button.salva=Save
```

Nello `struts-config.xml` andrà indicato qual è il resource bundle utilizzato dall'applicazione con il tag

```
<message-resources parameter="it.prova.ApplicationResources"/>
```

indicando il nome radice della famiglia di resource bundle che si riferiscono alle stesse risorse localizzate.
I file di properties così definiti vanno quindi installati nella cartella /WEB-INF/classes dell'applicazione rispettando la struttura di package dichiarata nella definizione precedente.

3.1.14 La lettura dei resource bundle in Struts

I file di properties vengono letti allo start-up dell'applicazione, e usati per valorizzare istanze della classe `org.apache.struts.util.PropertyMessageResources` memorizzate poi nel `ServletContext`. È quindi possibile accedere ai resource bundle dalle `Action`, dagli `ActionForm` o dalle pagine JSP. Nelle `Action` il reperimento può essere fatto utilizzando i metodi della classe `org.apache.struts.util.MessageResources` come nell'esempio seguente relativo ad una `Action`:

```
//acquisizione del locale corrente
Locale locale = this.getLocale(request);
// acquisizione del MessageResources
MessageResources messages = servlet.getResources();
// acquisizione della stringa localizzata corrispondente al
//locale corrente e alla chiave memorizzata nel parametro key
String value = messages.getMessage(locale,key);
```

oppure più frequentemente mediante i costruttori delle classi `ActionMessages` e `ActionErrors` utilizzate nella gestione dei messaggi di errore, che ricevono come parametro la chiave della label da reperire e acquisiscono in automatico dal resource bundle del locale corrente il valore della label stessa localizzata. Un esempio è il metodo validate di una `ActionForm` nel quale viene utilizzata la classe `ActionError` per rappresentare un messaggio di errore il cui valore localizzato viene reperito passando al costruttore la chiave corrispondente.

```
public ActionErrors validate(ActionMapping mapping,
HttpServletRequest request) {
   ActionErrors errors = new ActionErrors();
   if ((username == null) || (username.length() < 1))
      errors.add ("username",new
          ActionError("errore.username.obbligatorio"));
   if ((password == null) || (password.length() < 1))
      errors.add("password",new
          ActionError("errore.password.obbligatoria"));
   return errors;
}
```

Per reperire le stringhe localizzate nelle pagine JSP si utilizza invece un custom-tag della libreria struts-bean, precisamente il tag `<bean:message>`. L'utilizzo è banale , basta fornire come attributo la chiave corrispondente alla label che si vuole acquisire come nell'esempio seguente:

```
<bean:message key="label.username" />
```

In questo modo è molto semplice scrivere pagine JSP nelle quali non sono presenti label direttamente scritte nel codice , e quindi utilizzare lo stesso sorgente della pagina per visualizzare informazioni in lingue differenti.

3.1.15 Il Validator

Il codice di validazione è spesso duplicato nell'applicazione e un cambiamento delle regole di validazione implica una modifica ai sorgenti. Per ovviare a questi problemi è stato sviluppato il Validator, un componente del Jakarta Common Project che può essere usato come un add-in di Struts.
Il Validator è un framework che fornisce gli strumenti per effettuare in modo automatico e dichiarativo i controlli formali sui campi digitati in un form HTML. Usando il Validator non è necessario scrivere alcun codice di validazione nel metodo `validate()` degli `ActionForm`, ma è il Validator stesso che fornisce questa funzione purchè i form bean dell'applicazione estendano uno degli `ActionForm` del Validator stesso.
Il Validator è costiuito da un insieme di classi predisposte per eseguire tutti i più comuni controlli di validazione in genere usati nelle applicazioni, ma.esiste anche la possibilità di creare routine di validazione non fornite dal Validator. Il Validator inoltre supporta sia la validazione server-side che quella client-side mediante opportune funzioni JavaScript, cosa non fornita dal meccanismo standard degli `ActionForm` di Struts.
La configurazione delle routine di validazione da applicare ad un campo di un form è fatta mediante un file di configurazione XML, quindi esternamente all'applicazione ed è facilmente modificabile al mutare delle esigenze applicative. Nel file `validator-`

`rules.xml` vengono dichiarate tutte le routine di validazione disponibili, i loro nomi logici e il codice JavaScript corrispondente a ciascuna routine di validazione per l'esecuzione dei controlli client-side.
Nel file `validation.xml` si specifica come queste routine vengano applicate ai vari campi di input dei form dell'applicazione, ai quali si fa riferimento mediante i nomi dei form beans dichiarati nello `struts-config.xml`.
Per utilizzare il Validator con Struts basterà eseguire i seguenti step:

- Abilitare il Validator plug-in
- Configurare i due file XML appena citati, `validator-rules.xml` e `validation.xml`
- Creare form bean che estendano gli `ActionForm` del Validator.

3.1.16 Configurare il Validator per l'uso con Struts

Il Validator viene configurato come un plug-in di Struts. Per abilitare il Validator basterà aggiungere nello `struts-config.xml` il seguente frammento:

```
<plug-in
  className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-
   INF/validator- rules.xml, /WEB-INF/validation.xml"/>
</plug-in>
```

In questo modo Struts all'avvio dell'applicazione carica ed inizializza il Validator; si può notare come vengano definiti i percorsi dei due file XML di configurazione del Validator.
Il `validator-rules.xml` dichiara le routine di validazione messe a disposizione dal Validator. Esistono una serie di routine fornite dalla versione standard del Validator che coprono la gran parte delle comuni esigenze per ciò che riguarda i controlli formali da eseguire sui campi di input di un form. Il file in genere non va quindi modificato a meno che non si vogliano definire delle proprie routine. Di seguito è riportato un esempio di una sezione del file `validator-rules.xml` riguardante la definizione della routine di validazione corrispondente al criterio

“required”, ovvero di obbligatorietà di un campo del form. Come si può vedere ad ogni regola è associato un nome logico, required in questo caso, è definita la classe Java che effettua la validazione e il metodo che viene mandato in esecuzione con i suoi parametri.
Il nome logico della routine sarà usato come riferimento nel file `validation.xml` che vedremo in seguito.
Inoltre nel tag <javascript> potrà essere inserita la routine JavaScript per la validazione client-side.

```
<form-validation>
  <global>
    <validator name="required"
     classname="org.apache.struts.validator.FieldChecks"
     method="validateRequired"
     methodParams="java.lang.Object,
     org.apache.commons.validator.ValidatorAction,
     org.apache.commons.validator.Field,
     org.apache.struts.action.ActionErrors,
     javax.servlet.http.HttpServletRequest"
     msg="errors.required">
       <javascript>
        …
       </javascript>
    </validator>
  </global>
</form-validation>
```

Come già accennato per utilizzare il Validator i form bean dell’applicazione non devono estendere la classe `ActionForm` standard di Struts ma la classe `org.apache.struts.validator.ValidatorForm` che fornisce l’implementazione del metodo `validate()`. In questo caso non è più necessario scrivere il codice di validazione perché è il Validator che lo fa per noi. La configurazione dei form bean all’interno dello `struts-config.xml` è identica a quella fatta in precedenza utilizzando la classe `ActionForm` standard di Struts. La classe è simile ad un `ActionForm` di Struts tranne l’estensione di una classe diversa. Inoltre in questo caso non bisogna implementare i metodi `reset()` e

`validate()` che sono invece implementati dalla classe `ValidatorForm`. Il nome dato al form bean nello `struts-config.xml` è usato nel `validation.xml` per far riferimento al form in questione. Il `validation.xml` viene usato per dichiarare i controlli che verranno effettuati sui campi di input di un form dell'applicazione. In relazione al form definito in precedenza il `validation.xml` potrebbe essere fatto come segue:

```
<form-validation>
  <formset>
    <form name="logonForm">
      <field property="username" depends="required">
        <arg0 key="label.username"/>
      </field>
      <field property="password" depends="required">
        <arg0 key="label.password"/>
      </field>
    </form>
  </formset>
</form-validation>
```

Per ogni form dell'applicazione va definito un elemento `<form></form>` in cui l'attributo name corrisponde al nome del form bean dichiarato nello `struts-config.xml`. Gli elementi `<field></field>` dichiarano i controlli da eseguire per i campi del form. Nell'esempio si dichiara che i campi username e password sono obbligatori mediante l'attributo depends.
Per i messaggi di errore associati a ciascuna routine di validazione il Validator utilizza il file `ApplicationResource.properties` di Struts. In questo file vengono definiti i messaggi di errore associati a ciascun criterio di validazione definito nel `validator-rules.xml`:

```
errors.required={0} is required.
errors.minlength={0} cannot be less than {1} characters.
errors.maxlength={0} cannot be greater than {2} characters.
errors.invalid={0} is invalid.
errors.byte={0} must be a byte.
errors.short={0} must be a short.
```

```
errors.integer={0} must be an integer.
errors.long={0} must be a long.0.
errors.float={0} must be a float.
errors.double={0} must be a double.
errors.date={0} is not a date.
errors.range={0} is not in the range {1} through {2}.
errors.creditcard={0} is not a valid credit card number.
errors.email={0} is an invalid e-mail address.
```

I messaggi hanno dei segnaposto per renderne variabile il contenuto a seconda del campo controllato. Il valore che il Validator sostituisce al segnaposto è fornito nel `validation.xml` nella sezione dedicata al form specifico. Nel caso del required dell'esempio precedente verrà inserita la descrizione corrispondente alla `key="label.password"` fornita nella sezione `<arg0></arg0>` del field corrispondente.

3.1.17 La DispatchAction

Struts fornisce alcune `Action` base che arricchiscono il framework di alcune funzionalità rispetto alla `Action` base standard e che sono utili a diversi scopi . Abbiamo già visto precedentemente che nel funzionamento standard di Struts , il framework esegue il metodo `execute()` della Action che corrisponde all'URL della richiesta effettuata dal client. Questo metodo costituisce quindi, il punto di ingresso nel framework a fronte di una richiesta. Tale funzionamento si adatta poco alle situazioni nelle quali è necessario eseguire una serie di elaborazioni tra loro logicamente collegate. Tipico è l'esempio di operazioni di inserimento, cancellazione , lettura e aggiornamento su una stessa tabella di un database. Sarebbe poco efficiente dover definire una `Action` per ciascuna singola operazione, in quanto questa tecnica porterebbe ad un proliferare di `Action` nell'applicazione e quindi ad una difficile gestione della stessa. Struts ci viene incontro fornendo una particolare azione che prende il nome di `DispatchAction` appartenente al package `org.apache.struts.actions.DispatchAction`.

La `DispatchAction` è assolutamente analoga ad una `Action` base ma fornisce la possibilità di invocare diversi metodi della stessa purché il client specifichi il metodo

da chiamare. In pratica è come una `Action` che non ha un solo metodo `execute()` ma ne ha vari con nomi diversi. Ognuno di questi metodi deve avere la stessa signature del metodo `execute()`.
Ad esempio una `DispatchAction` potrebbe avere i seguenti metodi:

```
public ActionForward inserisci(ActionMapping mapping,
  ActionForm form,HttpServletRequest req, HttpServletResponse
  res)  throws IOException,ServletException;

public ActionForward aggiorna(ActionMapping mapping,
  ActionForm form,HttpServletRequest req, HttpServletResponse
  res)  throws IOException,ServletException;

public ActionForward cancella(ActionMapping mapping,
  ActionForm form,HttpServletRequest req, HttpServletResponse
  res)  throws IOException,ServletException;

public ActionForward leggi(ActionMapping mapping, ActionForm
  form,HttpServletRequest req, HttpServletResponse res)
  throws IOException,ServletException;
```

Affinché il framework sappia a quale metodo delegare l'elaborazione della richiesta, il client deve fornire nella request un parametro contenente il nome del metodo corrispondente. Questo parametro va ovviamente specificato nella definizione della `Action` nello `struts-config.xml` nel seguente modo:

```
<action path="/gestioneTabella"
        type="it.quix.action.GestioneTabellaAction"
        name="gestioneTabellaForm"
        scope="request"
        input="/tabella.jsp"
        parameter="task"/>
```

Se quindi da una pagina JSP si vuole invocare il metodo `inserisci()` della `Action GestioneTabellaAction` sarà sufficiente specificare:

```
http://servername/context-root/gestioneTabella?task=inserisci
```

Chiaramente il nome del metodo può essere specificato in diversi modi, come un hidden contenuto in un form HTML oppure può essere impostato da una funzione JavaScript prima di eseguire il `submit()` del form.
L'aspetto più importante è la possibilità di raggruppare logicamente azioni tra di loro correlate in un'unica `Action` il che porta ad una migliore strutturazione dell'applicazione ed evita duplicazioni inutili di codice.

3.1.18 La ForwardAction

La `ForwardAction` consente di inoltrare il flusso dell'applicazione ad un'altra risorsa individuata mediante un URI valido, risorsa che può essere una pagina JSP una servlet o altro.
Questa particolare azione non fa altro che creare un `RequestDispatcher` ed effettuare il forward alla risorsa individuata dall'attributo parameter specificato nello `struts-config.xml`.

```
<action path="/vaiAllaPagina1"
        type="org.apache.struts.actions.ForwardAction"
        name="pagina1Form"
        scope="request"
        input="/pagina0.jsp"
        parameter="/pagina1.jsp "/>
```

La `ForwardAction` è molto utile quando nell'applicazione si hanno pagine JSP che non richiedono alcuna elaborazione a monte prima di essere visualizzate. Affinché si eviti di effettuare un inoltro alla pagina in questione direttamente da un'altra pagina JSP dell'applicazione si può usare la `ForwardAction`. In questo modo si resta aderenti al modello di Struts che prevede un controllo centralizzato di tutte le richieste e si predispone l'applicazione a modifiche future. Se un domani infatti la pagina in questione richiedesse invece qualche elaborazione basterà sostituire una `Action` opportuna al mapping precedentemente corrispondente alla `ForwardAction`.
Un altro utilizzo della `ForwardAction` è come elemento di integrazione con altre applicazioni data la sua caratteristica di effettuare un inoltro ad un URI generico.

3.2 Portal Bridge

3.2.1 Introduzione

I Portals Bridges permettono ad applicazioni sviluppate con i più comuni web framework come Struts, JSF, PHP, Perl e Velocity di essere utilizzate come Portlet JSR-168 compliant (vedi Bibliografia **[6]**).
Per quanto riguarda le applicazioni sviluppate in JSF (Java Server Faces) potranno essere eseguite come Portlet senza o con piccole modifiche utilizzando il JSF bridge che crea una JSFPortlet pronta per essere deployata nel portale. Discorso analogo per Velocity e Perl.
Per le applicazioni sviluppate con Struts e PHP il discorso è leggermente più complicato poiché questi due framework sono servlet based il che sta a significare che si ha il supporto di un singolo evento per la request, mentre per le Portlet ne sono previsti due quali action e render.
In più, la specifica delle Portlet non permette l'accesso alla servlet durante l'`ActionRequest`.
Una caratteristica distintiva dei Portals Bridges è che rimuovono queste barriere e quindi permettono il mapping di una `ActionRequest` sottostando all'implementazione della servlet (es: `StrutsAction`).
Tutto quello che serve è un'implementazione specifica a seconda del portale dell'interfaccia generica `ServletContextProvider`, messa a disposizione con le common library dei Portals Bridges.

3.2.2 Struts Bridge

Lo Struts Bridge è uno dei tanti software bridge appartente al progetto Apache Portal Bridge (vedi bibliografia **[21]**).
Compito dello Struts Bridge è quello di permette l'esecuzione di un'applicazione Struts come una Portlet JSR-168 compliant (Vedi Bibliografia **[20]**).
Applicazioni nuove o già esistenti potranno essere deployate in modo trasparente sia come Portlet che come normali Web Application.

Il bridge ha come obiettivo il superamento delle limitazioni e delle incompatibilità di Struts con la specifica delle Portlet.
Un'applicazione Struts esistente può essere utilizzata come una Portlet senza apportare modifiche al codice Java o alle pagine JSP se:

- vengono rispettate le regole legate al pattern MVC
- sono stati utilizzati i tag specifici di Struts per il rendering di tutte le risorse (come le immagini) e degli action links.

Se una Struts Portlet non utilizza caratteristiche specifiche delle Portlet, sarà possibile allo stesso tempo accedervi ed utilizzarla come Web-Application così da poterla testare in modo completamente indipendente dal Portale.
Come accennato nel paragrafo precedente lo Struts Bridge è stato sviluppato per essere indipendente dai vari Portali e utilizza solamente un'interfaccia (del portale) per avere l'accesso al Servlet enviroment a tempo di esecuzione. L'implementazione di questa interfaccia cambierà a seconda del portale e permetterà di avere il mapping tra i due eventi della request e la Struts Action.

3.2.3 L'interfaccia ServletContextProvider

Come detto in precedenza il problema della corrente specifica delle Portlet sorge quando si deve effettuare il processing dell'`ActionRequest` e della `RenderRequest` da una Servlet application.
Anche se pagine JSP e servlet possono essere invocate dalla Portlet, la specifica delle Portlet offre il supporto solamente durante la `RenderRequest`.
Quindi se i parametri della request sono solamente disponibili durante l'`ActionRequest` non potranno essere utilizzati da Struts per il processing il che si traduce in una seria limitazione.
Poiché la specifica delle Portlet è costruita sopra la specifica 2.3 delle Servlet, l'implementazione del `Portlet Container` e/o del portale stesso, solitamente ha accesso diretto al `ServletContext` il quale viene utilizzato durante l'`ActionRequest`.
Lo Struts Bridge utilizza l'interfaccia `ServletContextProvider` dalle portals-bridges-common library per ottenere l'accesso al `ServletContext`. Per essere in condizione di utilizzare lo Struts Bridge, la StrutsPortlet deve quindi prevedere

un'implementazione concreta dell'interfaccia `ServletContextProvider` (con i relativi parametri di inizializzazione).
Questa interfaccia è stata definita in modo da essere di facile implementazione per ogni specifico portale, non è possibile avere un'implementazione che sia abbastanza generica per tutti.

3.2.4 ActionRequest

Durante l'`ActionRequest` non è permesso scrivere contenuti nella risposta. Questo è permesso solamente durante la `RenderRequest` ma in questo caso la Portlet non ha più accesso ai parametri di input della request il quale fa diventare la `RenderRequest` generalmente inutile per il processing della Struts Action (vedi bibliografia **[22]**).
Struts d'altra parte conosce solamente un evento request (poiché Struts è servlet-based) e dopo aver processato l'`Action` si avrà il forwarding ad una pagina JSP o altro per effettuare il rendering.
Lo Struts Bridge risolve tale problema premettendo alla Struts Action di processare all'occorrenza l'`ActionRequest`.
Lo Struts Bridge salverà il contesto corrente (come i percorsi da includere o di cui fare il forward) in un oggetto separato chiamato `StrutsRenderContext`.
Quando verrà invocata la `RenderRequest`, lo Struts Bridge utilizzerà lo `StrutsRenderContext` per ripristinare lo stato così da poter continuare e terminare in modo corretto il processing. In questo oggetto viene anche salvato l'`ActionForm` corrente (se non già presente in sessione), gli `ActionMessages` e/o gli `ActionErrors`.
Un aspetto molto importante da tenere in considerazione è che l'oggetto `StrutsRenderContext` viene recuperato soltanto una volta.
La specifica delle Portlet prevede che le stesse utilizzino l'architettura MVC cambiando il proprio stato solamente durante l'`ActionRequest`, mentre nella `RenderRequest` (che può essere richiamata più volte) si aspetta solamente una nuova renderizzazione del corrente stato della Portlet.

Per avere massima conformità tra lo Struts Bridge e la specifica delle Portlet, le applicazioni Struts dovrebbero essere configurate in modo da utilizzare Action separate per il processing e per il rendering.
La dipendenza dallo `StrutsRenderContext` che permette il recupero dello stato durante la `RenderRequest` dovrebbe essere limitata ad una sola situazione pena l'insorgere di errori dovuti ad elaborazioni degli input.
Quando lo Struts Bridge incontra errori di validazione sui dati di input, viene effettuato il forward ("all'indietro") alla pagina di input dell'azione così come accade per Struts "nativo".

3.2.5 RenderContextAttributes

Anche se lo `StrutsRenderContext` può essere utilizzato per comunicare i parametri nella request presenti durante l'`ActionRequest` alla successiva `RenderRequest` (salvandoli nell'`ActionForm`), lo Struts Bridge prevede un'altra soluzione che può essere utilizzata senza dover apportare modifiche alla Struts Application.
Il bridge può essere configurato utilizzando una semplice definizione xml (chiamato di default `struts-portlet-config.xml`), per salvare durante la `ActionRequest` il valore di attributi dei quali viene specificato il nome e recuperarli durante la seguente `RenderRequest` o opzionalmente in ogni `RenderRequest` fino alla seguente `ActionRequest`. I valori degli attributi nella request verranno salvati nella sessione quindi richiede che venga implementata l'interfaccia `Serializable`.
Viene riportato di seguito un esempio di definizione del file `struts-portlet-config.xml`:

```
<config>
     <render-context>
          <attribute name="errors"/>
          <attribute name="message" keep="true"/>
     </render-context>
</config>
```

Con la configurazione riportata di sopra, tutti gli attributi nella request con nome errors e message vengono salvati dopo l'`ActionRequest`. L'oggetto errors può essere recuperato una sola volta ("single use only"), mentre grazie alla definizione `keep="true"` l'oggetto message potrà essere recuperato fino alla successiva `ActionRequest`.

3.2.6 Rendering Portlet Urls

Un altro problema che sorge utilizzando Struts per lo sviluppo di Portlet sta nel rendering di url validi.
Attualmente ci sono due problemi differenti:

- url per le risorse come le immagini, JavaScript o CSS stylesheets che richiedono di essere caricati dal client (browser)
- url per l'interazione con la Portlet stessa.

Il problema con gli url delle risorse è che il contenuto della Portlet è incluso solamente dal portale, insieme a questo è possibile avere contenuti di altre Portlet visualizzate nella stessa pagina.
Le Portlet eseguono in un proprio contesto (come avviene per le Web-Application) e il client (browser) vede solamente il contesto della pagina del portale a cui sta puntando.
Una Portlet quindi non può utilizzare percorsi relativi per indirizzare le risorse mentre è possibile anzi preferibile utilizzarli nelle Web-Application.
Per l'interazione con la Portlet, possono essere utilizzati i *PortletURLs* che vengono generati utilizzando le Portlet API o attraverso i Portlet JSP tag.
Vi sono a disposizione due differenti tipi di *PortletUrl*:

- `ActionURLs`
- `RenderURLs`

È veramente importante generare il tipo corretto di url a seconda dell'interazione che si vuole stabilire con la Portlet.
In più è possibile specificare nello `struts-portlet-config.xml` gli url definendo a quale tipologia appartengono scegliendo la categoria di appartenenza tra:

- `Resource`
- `ActionURL`
- `RenderURL`

Da notare che per poter usare gli Struts HTML JSP Tags (`html:form`, `html:link`, `html:rewrite`, `html:image` e `html:img`) bisogna importare nelle pagine in cui se ne fa uso la rispettiva TLD come ad esempio:

```
<%@ taglib uri="http://portals.apache.org/bridges/struts/tags-
    portlet-html" prefix="html" %>
```

Per i tag `html:link` e `html:rewrite`, sono presenti tre attributi booleani quali: `actionURL`, `renderURL`, `resourceURL`, usando questi attributi è possibile specificare quale tipo di url deve essere generato (è supportato solamente il valore "true"). Se nessun attributo risulta impostato e true di default viene generato un `RenderURL` è possibile comunque cambiare questa impostazione (solamente tra `renderURL` e `actionURL`) di default andando a modificare la definizione nello `struts-portlet-config.xml`.
Viene riportato di seguito un esempio di `struts-portlet-config.xml` con la definizione di alcuni *PortletUrl*:

```
<config>
      <render-context>
      <attribute name="errors"/>
          <attribute name="message" keep="true"/>
     </render-context>
     <portlet-url-type>
      <action path="/shop/add"/>
          <action path="/shop/switch"/>
          <action path="/shop/remove"/>
          <action path="/shop/signoff"/>
          <action path="/shop/viewCategory"/>
          <action path="/shop/viewItem"/>
          <action path="/shop/viewProduct"/>
          <action path="/shop/viewCart"/>
          <action path="/shop/newOrder"/>
          <render path="/shop/newOrderForm"/>
          <action path="/shop/listOrders"/>
          <resource path="/images/"/>
      </portlet-url-type>
</config>
```

Nell'esempio sopra riportato l'elemento `portlet-url-type` contiene tre differenti sub elementi: `action`, `render` e `resource`. L'attributo `path` specifica il prefisso

con il quale fa match ogni singolo url. Per modificare l'impostazione di default in base alle proprie esigenze basterà scrivere:

```
<config>
     <portlet-url-type default="render"|"action"/>
</config>
```

Avendo cura nel scegliere come default `render` o `action`.

3.2.7 Esecuzione dell'applicazione come Portlet e Web-Application allo stesso tempo

Lo Struts Bridge è stato progettato per permettere ad una Struts Application di essere eseguita come se fosse a tutti gli effetti una Portlet.
Se la `PortletServlet` (che estende l'`ActionServlet`) dello StrutsBridge rileva che non è acceduta "in modalità" Portlet, delega le richieste al sottostante Struts framework.
Se l'applicazione Struts non usa particolari specifiche delle Portlet e il mapping delle azioni è ancora valido per la Web-Application, allo stesso tempo l'applicazione potrà essere utilizzata anche come Portlet, basterà creare il file war e deployare tale file nel portale.

3.2.8 Modifiche da apportare alla Web-Application per eseguirla come Portlet

Vengono riportate di seguito le principali modifiche da apportare ad una Web-Application esistente per renderla deployabile come Portlet (vedi bibliografia **[21]**).
La chiave per la conversione è nel file `web.xml`, non viene più utilizzata la servlet `org.apache.struts.action.ActionServlet` ma la `org.apache.portals.bridges.struts.PortletServlet`.

```
<servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>
             org.apache.portals.bridges.struts.PortletServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
</servlet>
```

Nelle pagine JSP va modificata la versione della Struts HTML taglib come riportato:

```
<%@ taglib uri="http://portals.apache.org/bridges/struts/tags-
    portlet-html"   prefix="html" %>
```

Nello `struts-config.xml` bisogna aggiungere l'elemento controller come di seguito:

```
<controller pagePattern="$M$P" inputForward="false"
 processorClass=
 "org.apache.portals.bridges.struts.PortletRequestProcessor"/>
```

Andrà poi aggiunto un file di configurazione il cui nome di default è `struts-portlet-config.xml`, di cui ne è già stata descritta ampiamente la struttura nei paragrafi precedenti.
Come detto in precedenza è necessario fornire un'implementazione dell'interfaccia `ServletContextProvider` come esempio viene riportato il frammento di codice che viene utilizzato in Pluto (vedi bibliografia **[1]**).

```
Package it.quix;
import javax.portlet.GenericPortlet;
import javax.portlet.PortletRequest;
import javax.portlet.PortletResponse;
import javax.servlet.ServletContext;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;
import org.apache.pluto.core.impl.PortletContextImpl;
import
org.apache.portals.bridges.common.ServletContextProvider;
```

```
public class PortalServletContextProvider implements
ServletContextProvider {
  public ServletContext getServletContext(GenericPortlet
  portlet)
      {
      return ((PortletContextImpl)
             portlet.getPortletContext()).getServletContext();
      }

  public HttpServletRequest getHttpServletRequest(
  GenericPortlet portlet, PortletRequest request)
      {
        return (HttpServletRequest)
          ((HttpServletRequestWrapper)request).getRequest();
      }
  public HttpServletResponse
  getHttpServletResponse(GenericPortlet portlet,
  PortletResponse response)
      {
        return (HttpServletResponse)
        ((HttpServletResponseWrapper) response).getResponse();
      }
}
```

Poiché si vuole portare un Web-Application ad essere eseguita come Portlet bisogna aggiungere il file `portlet.xml`.

```
<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
      http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
      <portlet>
            <description>Admin</description>
            <portlet-name>Admin_Portlet</portlet-name>
            <display-name>Admin Portlet</display-name>
            <portlet-class>
               org.apache.portals.bridges.struts.StrutsPortlet
            </portlet-class>
            <init-param>
                  <name>ServletContextProvider</name>
                  <value>
                        it.quix.PortalServletContextProvider
                  </value>
            </init-param>
```

```
            <init-param>
                  <name>ViewPage</name>
                  <value>/dataView.do</value>
            </init-param>
            <expiration-cache>0</expiration-cache>
            <supports>
                  <mime-type>text/html</mime-type>
                  <portlet-mode>VIEW</portlet-mode>
            </supports>
            <portlet-info>
                  <title>Admin portlet</title>
                  <keywords>Quix Admin Portlet</keywords>
            </portlet-info>
      </portlet>
</portlet-app>
```

Nel file riportato si nota che l'azione associata alla pagina iniziale in view mode (che in questo caso è l'unica modalità a cui si offre supporto) è `dataView.do`, il caching è disabilitato poiché il valore dell'`expiration-cache` è impostato a 0 infine si nota anche il riferimento al package `it.quix.PortalServletContextProvider` che contiene l'implementazione dell'interfaccia `ServletContextProvider`.

3.2.9 Struts Portlet Liferay

L'applicazione sviluppata con l'ausilio dello Struts Bridge è stata testata su Pluto (vedi bibliografia**[1]**) e su Liferay (vedi bibliografia**[3]**). Per maggiore completezza in questo paragrafo vengono riportati i file di configurazione associati a Liferay poiché utilizza per il proprio funzionamento altri file oltre a quelli "canonici" di configurazione.
Come primo file di configurazione delle Portlet prendiamo in considerazione il file `portlet.xml`

```
<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/por
     tlet-app_1_0.xsd
     http://java.sun.com/xml/ns/portlet/portlet-app_1_0.xsd">
```

```
  <portlet>
      <description>Portlet di amministrazione</description>
      <portlet-name>Admin_Portlet</portlet-name>
      <display-name>Admin Portlet</display-name>
      <portlet-class>
            org.apache.portals.bridges.struts.StrutsPortlet
      </portlet-class>
      <init-param>
            <name>ServletContextProvider</name>
            <value>
                  com.liferay.util.apache.bridges.struts.
                  LiferayServletContextProviderWrapper
            </value>
      </init-param>
      <init-param>
            <name>ViewPage</name>
            <value>/dataView.do</value>
      </init-param>
      <expiration-cache>0</expiration-cache>
      <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
      </supports>
      <portlet-info>
             <title>Admin portlet</title>
             <keywords>Quix Admin Portlet</keywords>
      </portlet-info>
  </portlet>
</portlet-app>
```

Questo è il file più importante per la definizione della Portlet. Come si può notare, per prima cosa viene definita una breve descrizione seguita dal nome della Portlet.
La definizione del nome della Portlet rappresenta, forse, l'aspetto più delicato, poiché questa informazione (come si vedrà di seguito) verrà utilizzata anche in altri file di configurazione per fare riferimento alla medesima Portlet. Utilizzando Liferay, un aspetto da tenere in considerazione è che il portale non accetta spazi nella definizione del nome della Portlet pena il non corretto funzionamento dell'applicazione stessa, proprio per questo nell'esempio sopra riportato al posto dello spazio è stato inserito un underscore.
Successivamente viene definito il nome che verrà visualizzato nella Portlet e dalla classe dello Struts Bridge che permetterà il deploy della web application come se fosse una Portlet. Di seguito vengono definite altre impostazioni di configurazione tra cui le azioni (di Struts) che devono essere associate alle modalità di funzionamento della

Portlet (`view`, `edit` e `help`), inoltre molto importante è anche il campo `expiration-cache` il quale se impostato a zero impone al portale di evitare di portarsi in cache le informazioni riguardo alla Portlet, evidentemente questo parametro va impostato a seconda dell'applicazione.
Per Liferay oltre ai normali file xml di definizione delle Portlet vanno aggiunti due file:

- `liferay-display.xml`
- `liferay-portlet.xml`

Per quanto riguarda il `liferay-display.xml` basta definire a quale categoria appartiene la Portlet in questione (in questo caso viene definita come appartenente alla categoria test), e l'identificativo che corrisponde al `portlet-name` definito nel file `portlet.xml`.

```
<display>
	<category name="category.test">
		<portlet id="Admin_Portlet" />
	</category>
</display>
```

Prendiamo adesso in considerazione il file `liferay-portlet.xml` nel quale bisogna specificare il nome della Portlet ricordandosi di inserire lo stesso nome definito nel file `portlet.xml`, successivamente vengono definiti i ruoli che sono richiesti per l'esecuzione della Portlet, nel caso seguente viene data la possibilità di visualizzazione ad ogni tipologia di utente (Administrator, Guest, power User ed User).

```
<liferay-portlet-app>
	<portlet>
		<portlet-name>Admin_Portlet</portlet-name>
		<portlet-url-class>
			com.liferay.portal.apache.bridges.struts.
			LiferayStrutsPortletURLImpl
		</portlet-url-class>
		<use-default-template>true</use-default-template>
		<restore-current-view>true</restore-current-view>
		<instanceable>false</instanceable>
		<private-request-attributes>
			true
		</private-request-attributes>
```

```
	</portlet>
	<role-mapper>
		<role-name>administrator</role-name>
		<role-link>Administrator</role-link>
	</role-mapper>
	<role-mapper>
		<role-name>guest</role-name>
		<role-link>Guest</role-link>
	</role-mapper>
	<role-mapper>
		<role-name>power-user</role-name>
		<role-link>Power User</role-link>
	</role-mapper>
	<role-mapper>
		<role-name>user</role-name>
		<role-link>User</role-link>
	</role-mapper>
</liferay-portlet-app>
```

3.2.10 Considerazioni

Uno dei benefici di questo "dual mode" è che l'applicazione potrà essere testata come Web-Application che è sicuramente più agevole rispetto al testing delle Portlet JSR-168 compliant.

Inoltre lo sviluppo di una Web-Application è molto più veloce e offre maggiori gradi di libertà rispetto alle Portlet, infatti, utilizzando Struts si ha la possibilità di avere un reale disaccoppiamento dei compiti (Model, View e Controller), avendo come risultato un codice maggiormente strutturato, di più semplice manutenzione e più facilmente estendibile.

L'applicazione che è stata sviluppata è stata progettata in modo da poter essere utilizzata sia come normale Web-Application che come Portlet.

Nello specifico, non sono state utilizzate le azioni normali di Struts, ma è stato scelto di avvalersi della `DispatchAction` (già descritta in precedenza). Quest'ultima ha creato qualche problema quando si andava ad utilizzare l'applicazione come Portlet. Infatti ogni qualvolta nello `struts-config.xml` si presentava un forward in cui veniva passato un parametro come ad esempio:

```
<forward name="save" path="/dataView.do?task=list" />
```

Quando si andava a testare il funzionamento della Portlet, veniva segnalato il seguente errore:

```
Request[/dataView] does not contain handler parameter named
'task'. This may be caused by whitespace in the label text.
```

Questo errore è dovuto dal fatto che l'applicazione va a cercare un parametro che non viene trovato anche se è stato correttamente inserito. Per risolvere tale problema, si è cercato di limitare il più possibile i forward di questo tipo, in più per gli scopi della nostra applicazione i forward dallo `struts-config.xml` passavano sempre come parametro "task=list" per cui si è deciso di implementare il metodo `unspecified()` della `DispatchAction` che viene invocato ogni volta che non viene trovato il parametro atteso (come nel nostro caso). Implementando tale metodo non è stato fatto altro che richiamare il metodo list già scritto in precedenza:

```
public ActionForward unspecified(ActionMapping mapping,
     ActionForm  form, HttpServletRequest request,
     HttpServletResponse response) throws Exception {
          return list(mapping, form, request, response);
}
```

Da notare che tale problema sorge solamente nei forward fatti nello `struts-config.xml`, mentre negli `html:form` (`<html:form action="/dataView?task=list">`) o negli `html:link` (`<html:link action="dataView.do?task=list">`) tutto funziona correttamente.
Avendo sviluppato l'applicazione in modo da poter essere utilizzata sia come Web-Application sia come Portlet si è provato a rendere disponibile quest'ultima attraverso lo standard WSRP (vedi bibliografia **[14]**) così da poter essere utilizzata da portali "Consumer" differenti.
L'unica limitazione che è stata riscontrata sta nei path delle immagini, infatti solitamente venivano indicati in questo modo:

```
<html:img src=
 "<%= request.getContextPath()%>/images/void.gif"/>
```

Impostando in questo modo i percorsi delle immagini, lato Consumer non venivano riscritti in modo corretto e come risultato non venivano ritrovate le immagini associate. Questo accade perché i percorsi (nello standard WSRP) vanno codificati nell'Url, infatti nelle Portlet JSR-168 che vogliono essere messe a disposizione in modalità remota i path vengono definiti nel modo seguente:

```
<img SRC="<%=renderResponse.encodeURL(
   renderRequest.getContextPath() + "/images/void.gif")%>"/>
```

Il problema è che utilizzando lo Struts Bridge non si ha a disposizione l'oggetto `renderResponse` e di conseguenza non è possibile la codifica dei percorsi nell'Url. Questa è una grande limitazione poiché la soluzione a questo problema è di definire tutti i percorsi assoluti come ad es:

```
<html:img src="http://WSRPServer:8080/
  <%= request.getContextPath() %>/images/void.gif"/>
```

Questo però ha come effetto negativo un irrigidimento dell'applicazione, per cui tale soluzione non è stata adottata.
Utilizzando lo Struts Bridge per lo sviluppo di un'applicazione utilizzabile in "dual mode", non è possibile, come già detto in precedenza, l'utilizzo di peculiarità delle Portlet come la sessione, con le due tipologie di visibilità (`APPLICATION_SCOPE` e `PORTLET_SCOPE`) poiché l'applicazione non sarebbe più eseguibile come Web-Application. Di fatto tutti gli oggetti messi in sessione avranno visibilità di tipo `APPLICATION_SCOPE` così che verranno condivisi da tutte le Portlet appartenenti alla stessa applicazione. Questa è una limitazione soprattutto se si vorrebbe lasciare degli oggetti visibili solamente a livello di Portlet.

4. Descrizione applicazioni sviluppate

4.1 DataBase di Supporto

Per le applicazioni sviluppate è stato definito un DataBase per il supporto di tutte le operazioni che verranno messe a disposizione. Come DBMS è stato utilizzato MySQL 5.0 poiché gratuito e adatto per i nostri scopi.

4.1.1 Schema E/R

In questo paragrafo viene riportato lo schema E/R (Entity Relationship) privo degli attributi presenti per favorire una più agevole comprensione dello schema.

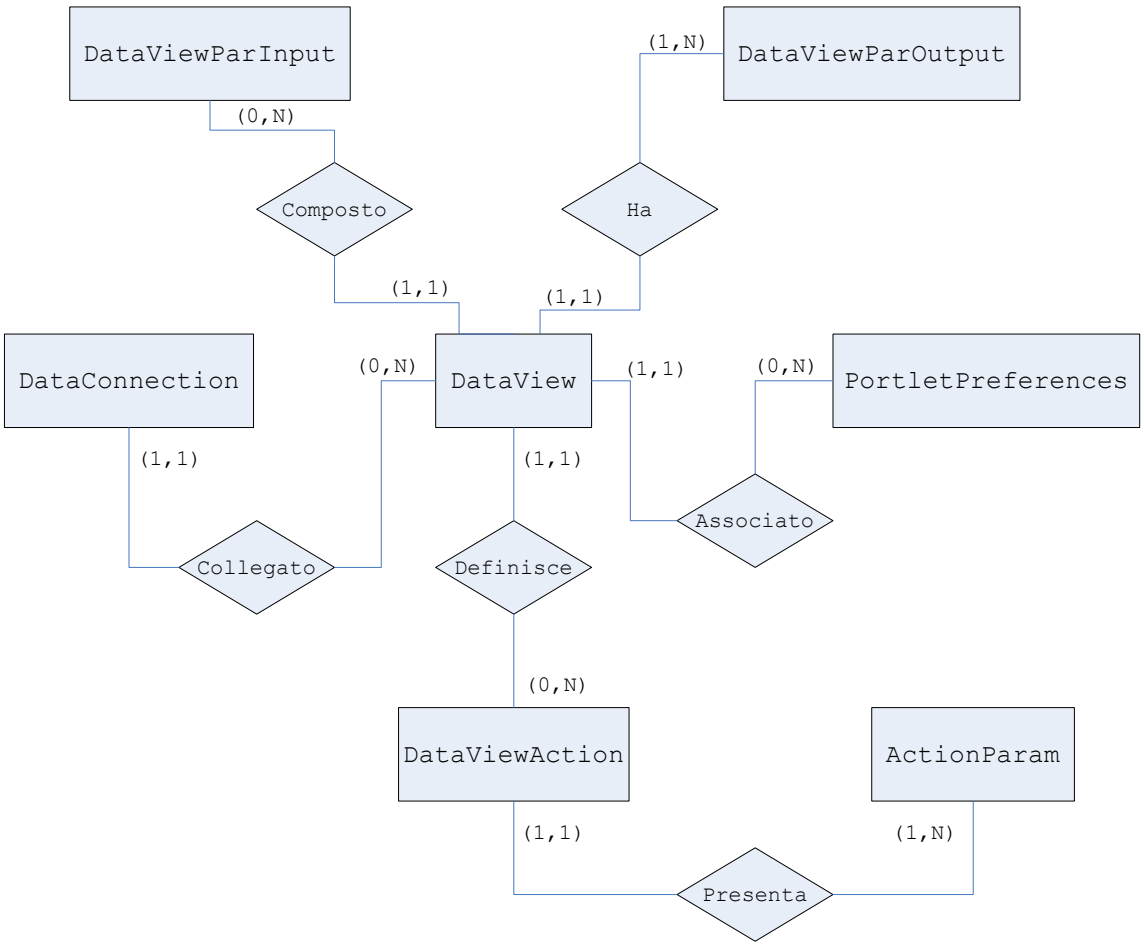

Figura 36: Schema E/R

4.1.2 Schema Logico

In questo paragrafo viene riportato lo schema logico in cui vengono riportate le chiavi primarie (sottolineate) e le chiavi esterne associate al DataBase.

DataView(idDV,nomeDV,idDSAppoggio,descrizioneDV,query)
FK:idDSAppoggio REFERENCES DataConnection(id)
DataViewParInput(idDV, nome, tipo, obbligatorio, valDefault)
FK:idDV REFERENCES DataView(idDV)
DataViewParOutput(idDV, nomeCampo, tipo, nomeVisualizza)
FK:idDV REFERENCES DataView(idDV)
DataViewAction(idAction, idDV, nomeAzione, iconFile)
FK:idDV REFERENCES DataView(idDV)
DataConnection(id, dataBaseUrl, dataBaseUser, dataBasePassword, dataBaseDriver)
Counter(value, name)
ActionParam(idParam, idAction, nomeCampo, nomeScelto)
FK:idAction REFERENCES DataViewAction(idAction)
PortletPreference(portletHandle, idDV)
FK:idDV REFERENCES DataView(idDV)

4.1.3 Utilizzo tabelle

Di seguito viene riportata una breve descrizione delle tabelle presenti nel DataBase:

- DataView: utilizzata per la gestione delle informazioni generali del DataView quali:
  - Nome associato al DataView
  - Descrizione (opzionale)
  - Query
  - idDSAppoggio che è la chiave esterna che referenzia la tabella DataConnection la quale serve per sapere su quale DataBase effettuare l'interrogazione.
- DataViewParInput: contiene le informazioni legate ai parametri di input di ogni DataView quali:

    - idDV è la chiave esterna che referenzia la tabella DataView la quale serve per sapere a quale DataView è associato ogni parametro di input presente nella tabella
    - nome del parametro
    - tipo associato (Stringa, Intero, Data, ecc…)
    - obbligatorio che indica l'obbligatorietà o meno del parametro
    - valDefault se specificato indica il valore di default associato al parametro
- DataViewParOutput: contiene le informazioni legate ai campi da riportare in output di ogni DataView quali:
    - idDV è la chiave esterna che referenzia la tabella DataView la quale serve per sapere a quale DataView è associato ogni parametro di output presente nella tabella
    - nomeCampo così come definito nel DataBase
    - tipo del campo
    - nomeVisualizza nome alternativo che è possibile associare al nome del campo. Questo parametro è stato inserito poiché campita molto spesso che nei DataBase si abbiano nomi di campi poco significativi, in questo modo sarà possibile specificare un nome alternativo così da avere una visualizzazione più leggibile.
- DataViewAction: contiene le informazioni di base delle azioni:
    - idDV è la chiave esterna che referenzia la tabella DataView la quale serve per sapere a quale DataView è associata ogni azione presente nella tabella
    - nome dell'azione
    - iconFile (opzionale) indica il file dell'icona associata all'azione
- ActionParam: contiene i nomi dei campi associati ad ogni azione che verranno utilizzati per instaurare la comunicazione tra Portlet, in particolare:
    - idAction è la chiave esterna che referenzia la tabella DataViewAction la quale serve per sapere a quale azione è associato ogni parametro presente nella tabella
    - nomeCampo che contiene il nome "nativo" del campo associato al DataView

    - nomeScelto che se specificato rappresenta un nome alternativo rispetto al precedente
- DataConnection: contiene le informazioni per connettersi ai DataConnection definiti:
    - dataBaseUrl contiene l'url del DataBase
    - dataBaseUser contiene il nome dell'utente
    - dataBasePassword contiene la password per l'accesso
    - dataBaseDriver contiene i driver per l'accesso al DataBase così da poter supportare vari DBMS
- Counter: tabella di appoggio utilizzata per la generazione delle chiavi per la tabella DataConnection
- PortletPreference: utilizzata per la gestione delle preferenze delle Portlet remote composta da:
    - PortletHandle utilizzato per identificare ogni singola istanza di Portlet
    - idDV è la chiave esterna che referenzia la tabella DataView la quale serve per sapere a quale DataView è associata ogni Portlet presente nella tabella

4.2 Applicazione di amministrazione

L'applicazione di amministrazione è stata sviluppata utilizzando Struts e lo Struts Bridge (vedi bibliografia **[20]**) così da renderne possibile l'utilizzo sia come normale Web-Application che come Portlet.
Questa applicazione si propone di affrontare principalmente i seguenti punti:

- Creazione, modifica, cancellazione e visualizzazione DataView
- Creazione, modifica e cancellazione Data Source
- Creazione, modifica e cancellazione Azioni

Nei paragrafi seguenti si entrerà in maggiore dettaglio fornendo vari screen-shot per mostrare il reale funzionamento dell'applicazione sviluppata.

4.2.1 Parte dedicata ai Data Source

I Data Source sono sorgenti dati caratterizzati dai seguenti attributi:

- url del Database
- username indica il nome utente per accedere al DBMS
- password associata al nome utente per accedere al DBMS
- driver dipendenti dal DBMS con cui è stato progettato il Database a cui collegarsi. Questo attributo permette di essere totalmente svincolati dalla tipologia di DBMS e offre la possibilità di avere maggiori gradi di libertà nella definizione dei Data Source

La finestra iniziale dell'applicazione associata ai Data Source avrà l'aspetto mostrato in figura.

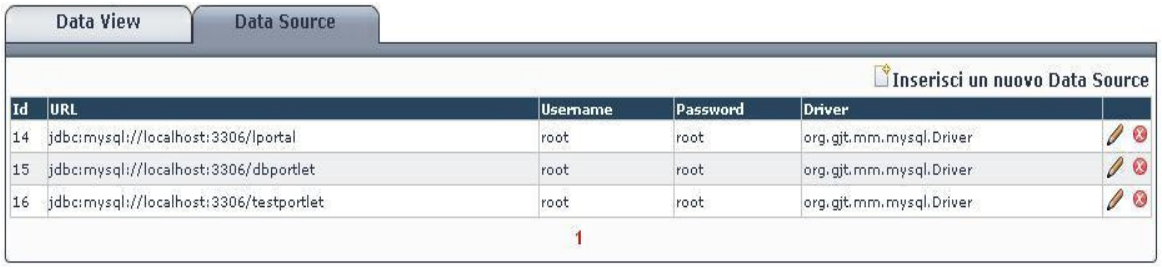

Figura 37: Home page Data Source

Si ha un elenco dei Data Source presenti in quel momento nella tabella DataConnection (presente nel data base di amministrazione).
Come si può notare oltre agli attributi sopra citatati, viene riportato anche l'id che è la chiave primaria della tabella DataConnection.
Un altro aspetto che può sembrare strano è quello che le password non vengono mostrate mascherate ma bensì in chiaro. Questa scelta è dovuta al fatto che questa applicazione verrà utilizzata da un utente "particolare" (come ad esempio un amministratore) e può essere più comodo avere una visione completa d'insieme con tutti gli attributi presenti.
Dall'immagine riportata di sopra, si nota che è possibile effettuare l'inserimento di un nuovo Data Source. Cliccando sull'icona verrà visualizzata una finestra che richiederà l'inserimento delle informazioni associate come mostrato in figura.

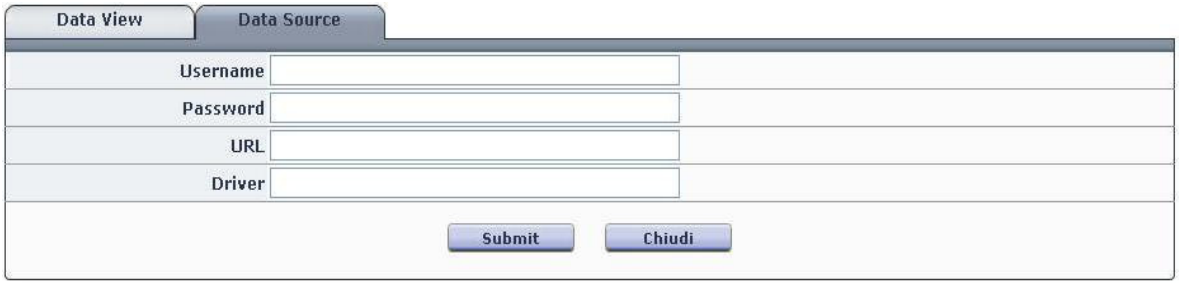

Figura 38: Inserimento Data Source

A questo punto si può decidere di tornare nella schermata precedente cliccando sul bottone "Chiudi" oppure si potranno inserire le informazioni associate ad un nuovo Data Source come da figura.

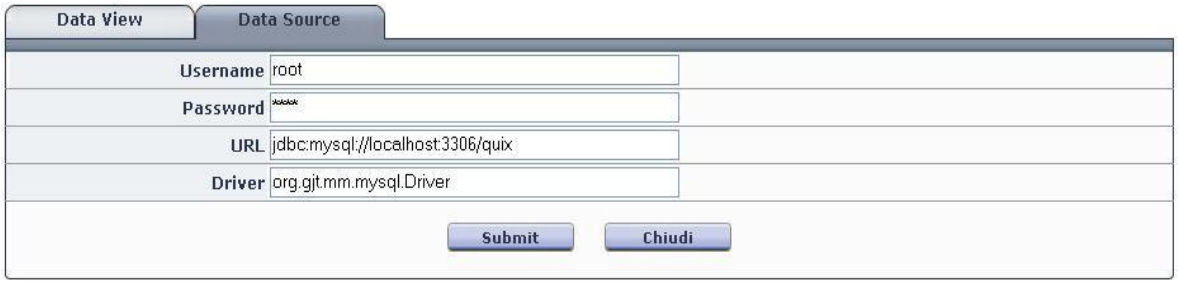

Figura 39: Inserimento parametri Data Source

Premendo sul bottone “Submit” si ritornerà nella schermata di visualizzazione iniziale con la lista aggiornata dei Data Source presenti.
Un’altra operazione che può essere effettuata è la modifica di un Data Source già esistente. Basterà selezionare l’icona in corrispondenza del Data Source che si desidera modificare. Verrà visualizzata una finestra di dialogo totalmente analoga alla figura precedente, già compilata con i valori associati al Data Source. Basterà agire sui parametri da modificare e cliccare sul pulsante “Submit” per aggiornare il Database con i nuovi valori.
L’ultima operazione che viene messa a disposizione è la cancellazione. È possibile l’eliminazione di un Data Source selezionando l’icona in corrispondenza dell’elemento che si desidera cancellare.

4.2.2 Parte dedicata ai DataView

In questo paragrafo verranno descritte le operazioni riguardanti i DataView ad eccezione delle azioni che saranno argomento del paragrafo successivo.
La pagina iniziale riguardante i DataView ne mostra la lista prendendo le informazioni dal Database di amministrazione. Nella figura sottostante ne viene riportato un esempio in cui è possibile notare la presenza di quattro DataView.

Figura 40: Lista DataView

Per ogni DataView viene riportato il nome, il Data Source di appoggio e se è stata inserita la descrizione.
Per inserire un nuovo DataView basterà selezionare l’icona posta in alto a destra.
Per prima cosa viene visualizzata una finestra di dialogo come da figura.

Figura 41: Inserimento informazioni generali DataView

Come si può notare viene richiesto di inserire il nome del DataView, di selezionare attraverso un menù a tendina il Data Source di appoggio (nel quale verrà eseguita la query che verrà inserita successivamente) e opzionalmente di inserire la descrizione associata.

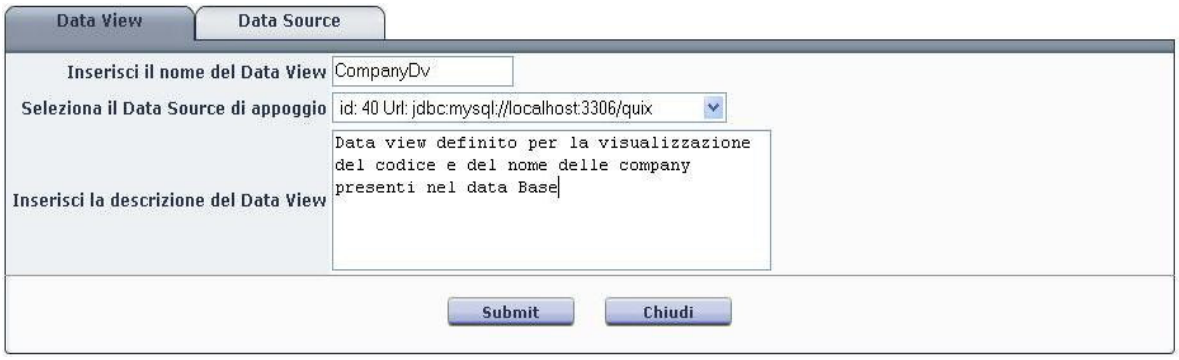

Figura 42: Informazioni generali DataView inserite

Una volta inserite le informazioni richieste e selezionato il pulsante "Submit" verrà visualizzata una schermata riassuntiva con i parametri inseriti.

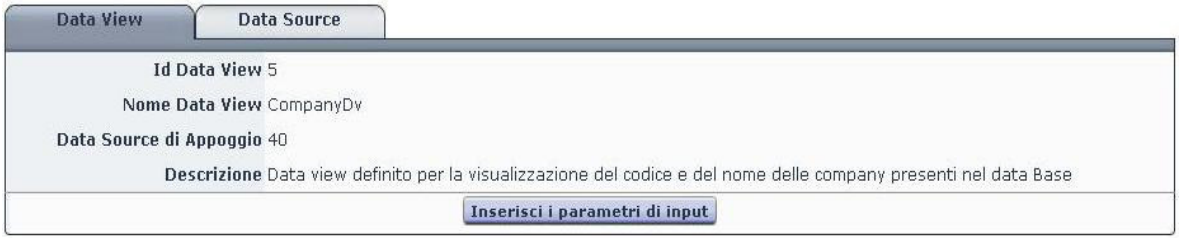

Figura 43: Schermata riassuntiva informazioni inserite

Cliccando sul pulsante “Inserisci parametri di input” verrà visualizzata una finestra di dialogo come da figura.

Figura 44: Schermata parametri di input

In questa schermata, di default, è possibile inserire al massimo dieci parametri di input, per ognuno, andrà specificato il nome, il tipo (scegliendo tra: String, Integer, Float, Double e Date), l’obbligatorietà ed opzionalmente il valore di default che si intende associare.

Non è detto che tutti i DataView debbano avere associati dei parametri di input, prendiamo in considerazione il caso in cui non ce ne siano.

Per evitare l’inserimento dei parametri basterà cliccare sul bottone “Submit”. A questo punto verrà visualizzata una finestra come da figura:

Figura 45: Inserimento query

Come si può notare in alto della finestra di dialogo, il DataView non ha associato alcun parametro di input. In questa fase è richiesto solamente l'inserimento della query.

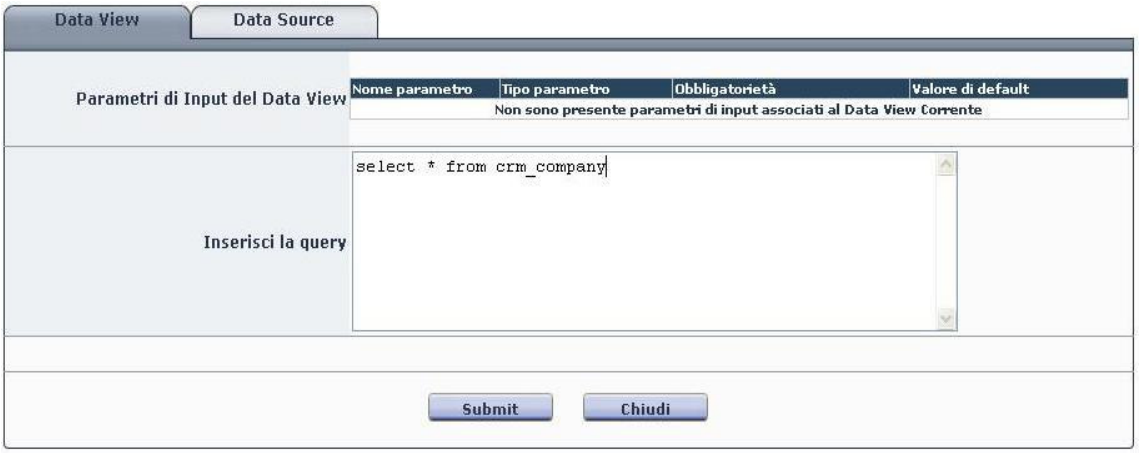

Figura 46: Query senza parametri di input

Una volta inserita la query basterà selezionare il bottone "Submit" per passare allo step successivo. Da notare, che in questa fase, vengono effettuati vari controlli sulla query inserita (per evitare malfunzionamenti) come ad esempio l'utilizzo di una tabella non presente nel Database associato al DataView.

Una volta superati i controlli di consistenza della query si avrà la selezione dei campi che si desidera vengano riportati in output.

Prima di questo, viene riportato per completezza l'esempio di un DataView che presenta un parametro di input per vedere come cambia la definizione della query.

Per prima cosa va indicato il parametro di input come da figura.

Figura 47: Definizione parametro di input

Nell'esempio riportato è stato definito un solo parametro di input con nome `CompanyCode`, di tipo `String`, obbligatorio e senza valore di default associato. Una volta selezionato il bottone "Submit" verrà richiesto l'inserimento della query.

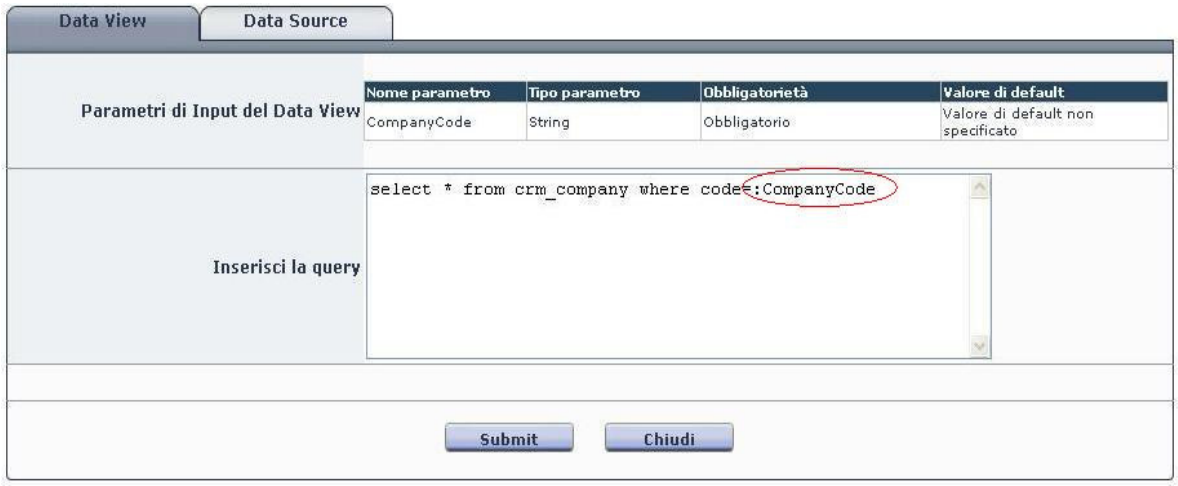

Figura 48: Query con parametro di input

Come si può notare in alto nella schermata viene riportato il parametro di input definito al passo precedente, mentre per inserire il parametro nella query viene utilizzata la seguente convenzione: `:NomeParametro`.
Anche in questo caso una volta superati i controlli di consistenza della query si avrà la schermata che riporta i campi di output (associati alla query appena inserita) come da figura.

Figura 49: Schermata che mostra i campi di output

In questa finestra viene richiesto di selezionare i campi che si desidera vengano riportati in output. Nella lista, si nota il nome del campo, il tipo (scegliendolo

attraverso un menù a tendina) ed opzionalmente il nome da visualizzare in output (poiché molto spesso i campi nei Database non hanno nomi significativi).
Può accadere che i campi da riportare in output siano parecchi, selezionando la checkbox in alto a sinistra verranno automaticamente selezionati tutti i campi presenti.

Figura 50: Selezione automatica di tutti i campi di output

Una volta selezionati i campi desiderati, basterà selezionare il pulsante "Submit" per terminare la definizione del DataView e vedersi riproporre l'elenco con i dati aggiornati.

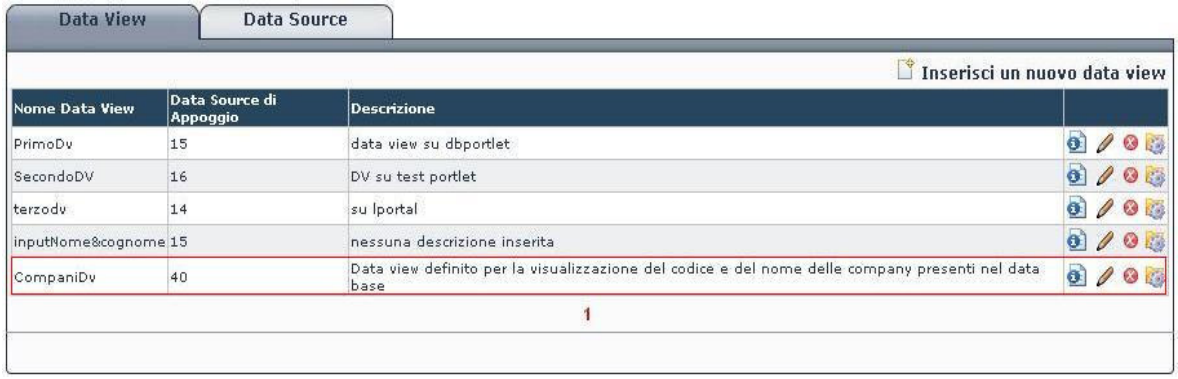

Figura 51: Elenco DataView aggiornato

Tramite l'icona sarà possibile la modifica di un DataView esistente, gli step saranno gli stessi affrontati per l'inserimento con la particolarità che saranno già presenti le varie informazioni, così che basterà agire solamente su quelle da modificare. Da notare che tramite la procedura di modifica sarà possibile completare la definizione dei DataView che per qualche ragione non era stata terminata.

Selezionando l'icona sarà possibile visualizzare i risultati associati al DataView corrispondente.

In questo caso si hanno tre possibilità:

- se il DataView non è stato definito completamente, il sistema negherà la visualizzazione mostrando un messaggio di notifica
- se il DataView (definito correttamente) non ha alcun parametro di input associato verranno visualizzati i risultati

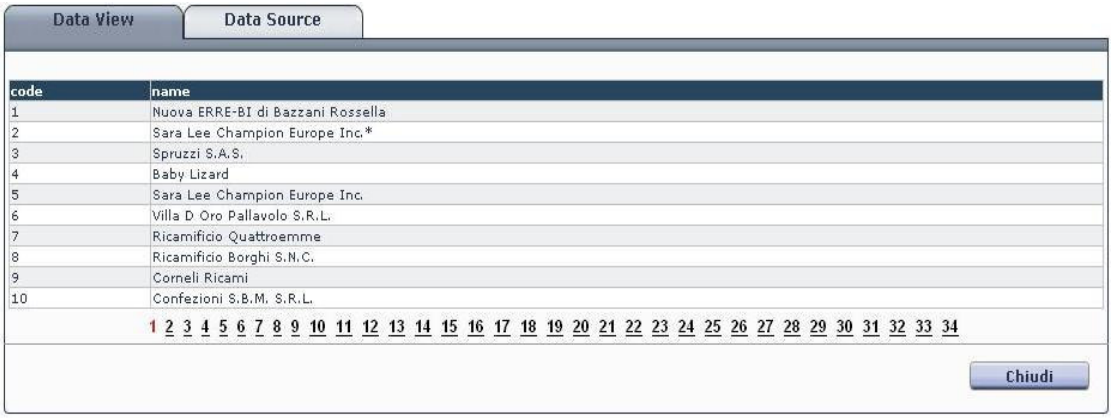

Figura 52: Risultati DataView senza parametri di input

- se il DataView (definito correttamente) ha dei parametri di input associati verrà visualizzata una finestra di dialogo che ne richiederà l'inserimento. Una volta che tali parametri saranno stati inseriti si avrà la visualizzazione dei risultati associati

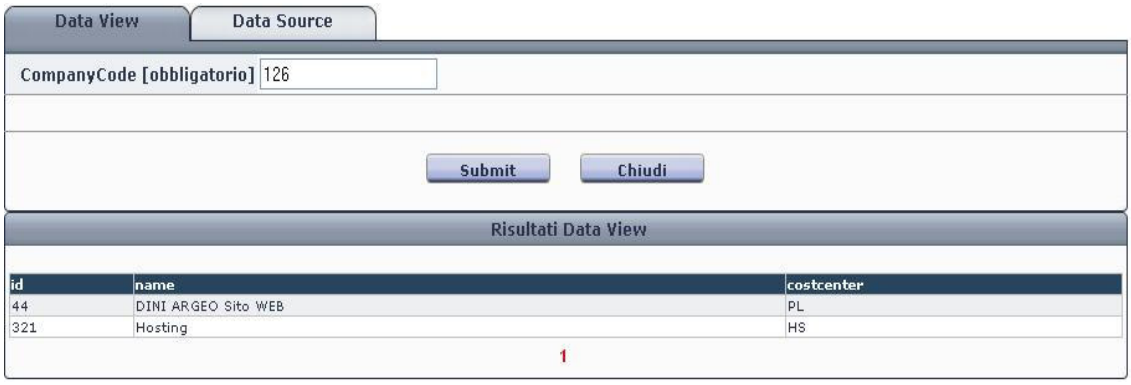

Figura 53: Risultati DataView con parametri di input

Un'altra operazione che è possibile effettuare sul DataView è la cancellazione selezionando l'icona in corrispondenza dell'elemento che si vuole eliminare.

4.2.3 Parte dedicata alle Azioni

Le azioni vengono definite per instaurare la comunicazione tra Portlet così come verrà illustrato con maggiore dettaglio nella parte dedicata alla Portlet di pubblicazione WSRP.

Per accedere all'elenco delle azioni associate ad un determinato DataView basterà selezionare l'icona in corrispondenza dell'elemento desiderato. A questo punto si possono presentare tre alternative:

- il DataView selezionato non è stato definito completamente per cui l'applicazione negherà attraverso un messaggio di notifica l'inserimento di azioni
- il DataView selezionato è stato definito correttamente ma non ha associata alcuna azione (ad esempio se il DataView è appena stato creato)

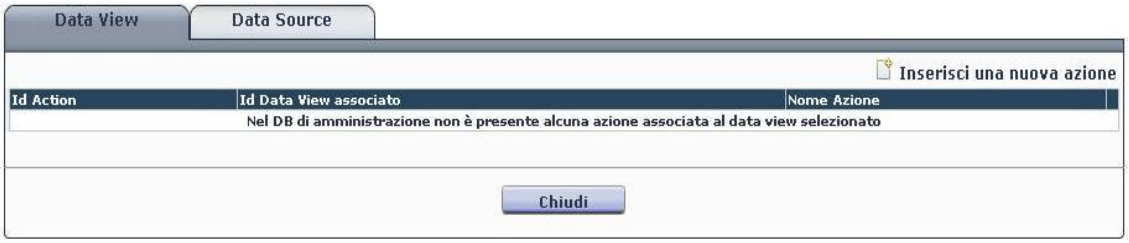

Figura 54: Lista azioni vuota

- Il DataView è stato definito correttamente e ha già associate alcune azioni

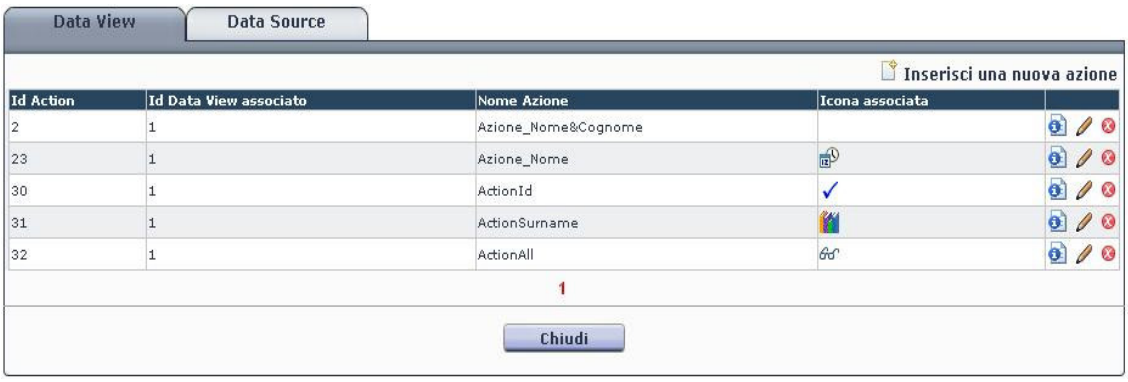

Figura 55: Lista azioni

Come si nota dalla figura, per ogni azione viene riportato l'id che funge da identificatore, l'id del DataView a cui è associata, il nome e se specificata l'icona.

Come prima operazione prendiamo in considerazione l'inserimento di una nuova azione, per fare ciò basterà selezionare l'icona posta nella pagina in alto a destra. Verrà visualizzata una finestra di dialogo analoga alla figura riportata di seguito.

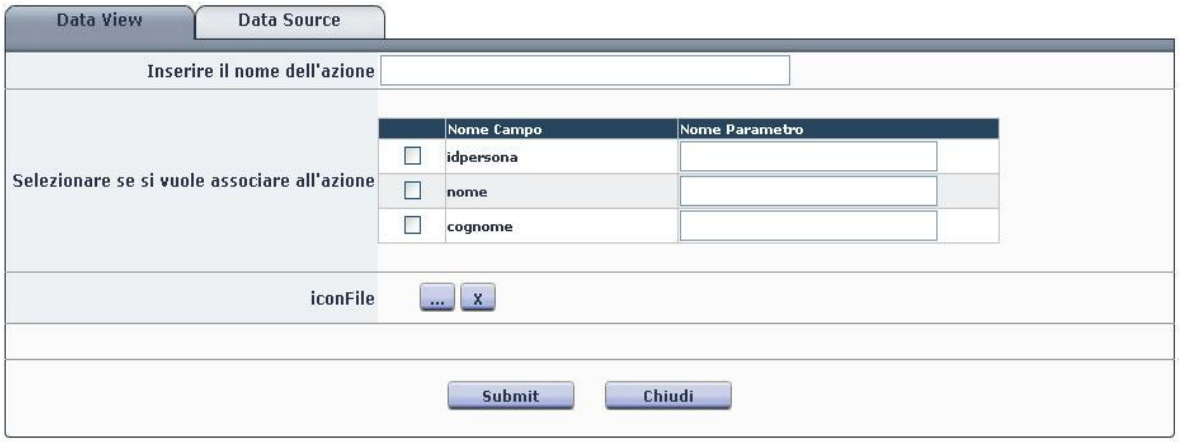

Figura 56: Inserimento nuova azione

A questo punto sarà necessario inserire il nome dell'azione, selezionare i parametri che si vogliono associare e specificare eventualmente un nome alternativo da dare ad ogni parametro selezionato.

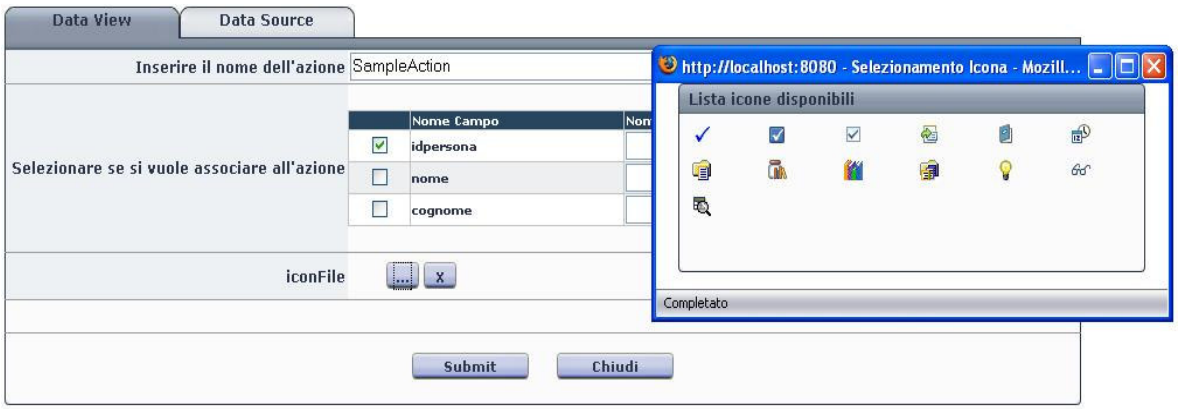

Figura 57: Inserimento parametri dell'azione

Nell'esempio sopra riportato è stata definita un'azione con nome "SampleAction" in cui è stato associato il parametro "idpersona". Come si può notare è presente la possibilità di associare all'azione un'icona, per aprire la finestra di dialogo con le icone presenti nel sistema basterà selezionare il bottone con i tre punti di sospensione mentre, se si desidera eliminare l'associazione ad una icona basterà selezionare il pulsante a fianco.

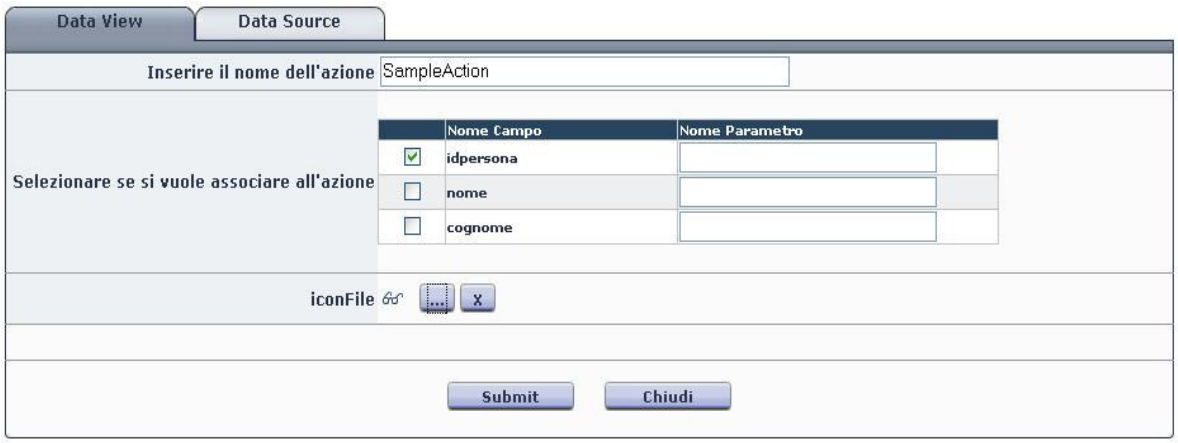

Figura 58: Parametri azione completi

Una volta terminato l'inserimento dei parametri basterà selezionare il bottone "Submit" e si tornerà alla pagina che mostrerà l'elenco aggiornato delle azioni.
Per quanto riguarda la modifica di un'azione esistente, basterà selezionare l'icona in corrispondenza dell'elemento desiderato. La finestra di dialogo che verrà visualizzata sarà del tutto analoga a quelle già viste per l'inserimento, con la particolarità che i vari campi saranno già compilati con i valori presenti nel Database. Basterà agire sui parametri da modificare ed automaticamente il sistema provvederà all'aggiornamento dei nuovi valori.
Selezionando l'icona sarà possibile visualizzare il dettaglio dell'azione selezionata. Come già visto nella definizione delle azioni, può essere specificata o meno l'icona associata per cui si avranno due tipologie di visualizzazione.

Figura 59: Dettaglio azione senza icona associata

Figura 60: Dettaglio azione con icona associata

Come ultima operazione prendiamo in considerazione l'eliminazione di un'azione. Basterà selezionare l'icona in corrispondenza dell'elemento desiderato per determinarne la cancellazione; una volta terminata questa operazione verrà visualizzata la lista delle azioni aggiornata .

4.3 Portlet di pubblicazione WSRP

4.3.1 Introduzione

Nel periodo di tirocinio è stata sviluppata una Portlet con l'obiettivo di assolvere i seguenti compiti:

- permettere la gestione delle preferenze così da non dover essere costretti ad ogni accesso al portale di riconfigurare le Portlet
- ogni Portlet sarà configurata in modo da visualizzare i risultati associati ad un particolare DataView, definiti con l'applicazione di amministrazione (precedentemente descritta)
- instaurare la comunicazione tra Portlet tramite l'utilizzo delle azioni definite sui DataView

Questa Portlet è stata sviluppata in accordo con la specifica JSR-168 e per il testing è stato utilizzato il portale Pluto (vedi bibliografia **[1]**).
Una volta terminato lo sviluppo della Portlet si è deciso di rendere disponibile l'applicazione tramite lo standard WSRP (vedi bibliografia **[14]**).
Poiché la Portlet è stata sviluppata con il supporto di Pluto (come portale), si è deciso di utilizzare quest'ultimo come Producer, ovvero, come il portale che mette a disposizione l'applicazione, in modo che possa essere utilizzata da ogni portale Consumer che supporti lo standard WSRP.
Come già detto in precedenza, Pluto rappresenta l'implementazione di riferimento della specifica JSR-168, con in aggiunta le funzionalità base di un portale. Per poterlo quindi utilizzare come Producer, è stato integrato col progetto WSRP4J sviluppato dall'Apache Software Foundation (vedi bibliografia **[16]**).
Per quanto riguarda i portali Consumer sono stati testati Liferay ed uPortal.

4.3.2 Configurazione della Portlet

Con il termine configurazione della Portlet si intende il processo che permette di associarvi un DataView da visualizzare.

Quando la Portlet viene deployata per la prima volta nel portale, viene mostrata la una finestra che consente, tramite un menù a tendina, di selezionare quale DataView si intende associare. L'elenco che verrà visualizzato dipenderà da quali DataView sono stati definiti tramite l'applicazione di amministrazione. Per estrarre questa lista sarà necessario interrogare la tabella `DataView` nel DataBase di amministrazione.
Come esempio viene riportata una figura che mostra quanto descritto.

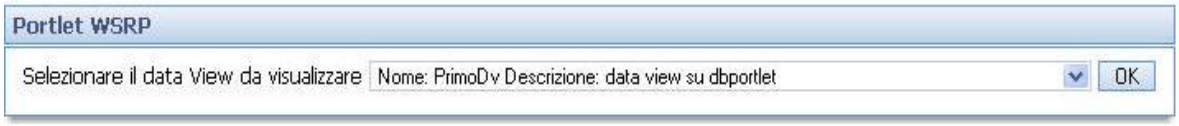

Figura 61: Configurazione Portlet

Di default verrà selezionato il primo DataView presente nel DataBase, basterà agire sul menù a tendina per scegliere il DataView da visualizzare.

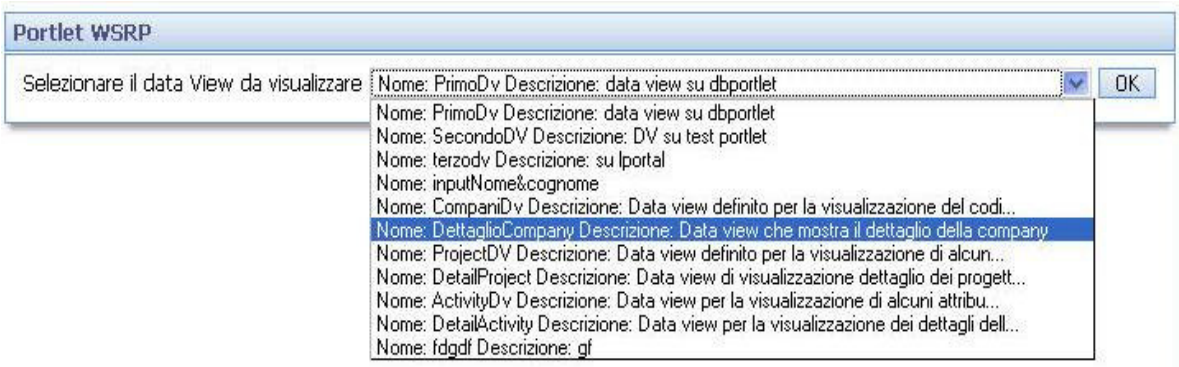

Figura 62: Menù di selezione DataView

Come si può notare nel menù vengono riportati il nome e la descrizione (se è stata inserita) del DataView, avendo cura del fatto che descrizioni troppo lunghe porterebbero ad avere un menù molto largo e si andrebbero a creare problemi nella visualizzazione. Per evitare questo inconveniente, le descrizioni più lunghe di una determinata soglia vengono troncate. Lo si può notare dove compaiono i classici tre punti di sospensione.
Una volta selezionato il DataView e premuto il pulsante "OK", verrà visualizzata una finestra con il nome e la descrizione (per intero) del DataView selezionato, in più per

quanto riguarda la presentazione dei risultati si potranno avere situazioni differenti descritte nel paragrafo dedicato alla visualizzazione dei risultati.
La configurazione verrà mantenuta anche per i successivi utilizzi. Questo è un aspetto molto utile ed importante che verrà trattato con maggiore dettaglio nel paragrafo "Remote Portlet Preferences".

4.3.3 Modifica della configurazione

Non bisogna pensare che la configurazione delle Portlet possa avvenire una sola volta infatti, come illustrato in questo paragrafo è possibile modificare la configurazione a seconda delle proprie esigenze. I DataView vengono definiti mediante l'applicazione di amministrazione, la quale ne permette la modifica e la cancellazione. Per questo motivo è possibile modificare la configurazione delle Portlet al fine di associarvi un altro DataView.
Per fare ciò con Liferay basterà andare nella sezione dedicata alle Preferences, verrà visualizzata una finestra di dialogo come da figura:

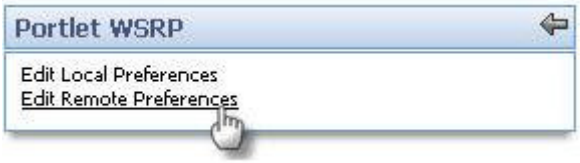

Figura 63: Edit Remote Preferences

Selezionando il link `Edit Remote Preferences` non si farà altro che accedere alla parte di edit mode della Portlet. La pagina che verrà visualizzata sarà del tutto analoga a quella che viene presentata quando la Portlet viene deployata per la prima volta nel portale. Basterà selezionare dal menù a tendina il nuovo DataView e successivamente si avrà la presentazione dei nuovi risultati.
L'utilizzo dell'edit mode non funziona utilizzando come portale uPortal, questo perché una volta terminata la configurazione il portale non consente di ritornare in view mode nemmeno forzandolo tramite il codice. Per superare tale limitazione e rendere utilizzabile l'applicazione anche da questo portale, in view mode è stato inserito un link che permette la modifica della configurazione senza dover andare in edit mode.

Figura 64: Modifica configurazione in view mode

Resta sottinteso che gli utenti che utilizzano Liferay potranno scegliere in modo totalmente equivalente quale delle due modalità utilizzare per la modifica da apportare.

4.3.4 Remote Portlet Preferences

Come detto in precedenza una volta avvenuta la configurazione della Portlet questa viene mantenuta anche nei successivi utilizzi senza il bisogno ad ogni successivo login di rieseguirla.

La specifica JSR-168 prevede l'utilizzo delle `PortletPreferences` che permette di salvare in modo persistente la configurazione delle Portlet, delegando i compiti da svolgere al portale. Utilizzando lo standard WSRP, si è notato che le `PortletPreferences` non funzionano correttamente. Il salvataggio delle informazioni necessarie deve avvenire "lato Producer", quest'ultimo salva la configurazione tramite il proprio handle della Portlet invece di utilizzare quello del Consumer. In questo modo avendo più istanze della stessa Portlet su vari "Consumer" le informazioni verranno continuamente sovrascritte ritrovando al successivo utilizzo solamente l'ultima configurazione effettuata.

Per risolvere tale problema, si è deciso di inserire una tabella aggiuntiva nel Database di amministrazione chiamata appunto `PortletPreference` composta da una campo handle che identifica ogni Portlet e dal codice del DataView associato. In questo modo

ad ogni prima richiesta di visualizzazione (dopo il login al portale) delle Portlet si andrà a verificare se l'handle:

- è presente in tabella, verrà visualizzato il DataView associato
- non è presente verrà visualizzata la finestra di dialogo per la configurazione della Portlet

Ovviamente nel caso di modifica della configurazione verrà aggiornato il record opportuno nella tabella.

4.3.5 Visualizzazione risultati

Una volta che la Portlet è stata configurata, l'utente viene mandato nella parte di visualizzazione dei risultati associati al DataView.
La visualizzazione dei risultati è differenziata in due modalità:

- se il risultato è composto da un unico record, la visualizzazione avviene per righe ovvero: nome del campo sulla sinistra e valore del campo sulla destra. Questo è stato pensato per agevolare la lettura dato che i campi possono essere numerosi
- se viceversa il risultato è composto da più record si ha la classica visualizzazione a tabella. Poiché il numero di record può essere numeroso, è stata inserita la gestione delle pagine (di default dieci record per pagina).

La finestra di dialogo contiene al suo interno le informazioni generali del DataView quali il nome e se presente la descrizione. Per quanto riguarda la visualizzazione dei risultati si potranno avere in generale quattro situazioni differenti:

- il DataView selezionato non ha parametri in ingresso e quindi verranno automaticamente visualizzati i risultati associati

Figura 65: DataView senza parametri di input

- il DataView ha solamente parametri facoltativi e quindi l'esecuzione del DataView non è subordinata all'inserimento di tali parametri e quindi verranno anche in questo caso presentati i risultati
- il DataView ha dei parametri obbligatori associati, in questo caso si possono presentare due situazioni diverse:
  - nella definizione del DataView è stato specificato il valore di default dei parametri
  - il valore dei parametri è stato trovato in sessione per effetto di un'azione associata ad un altro DataView.

In entrambi i casi si avrà la visualizzazione dei risultati associati.

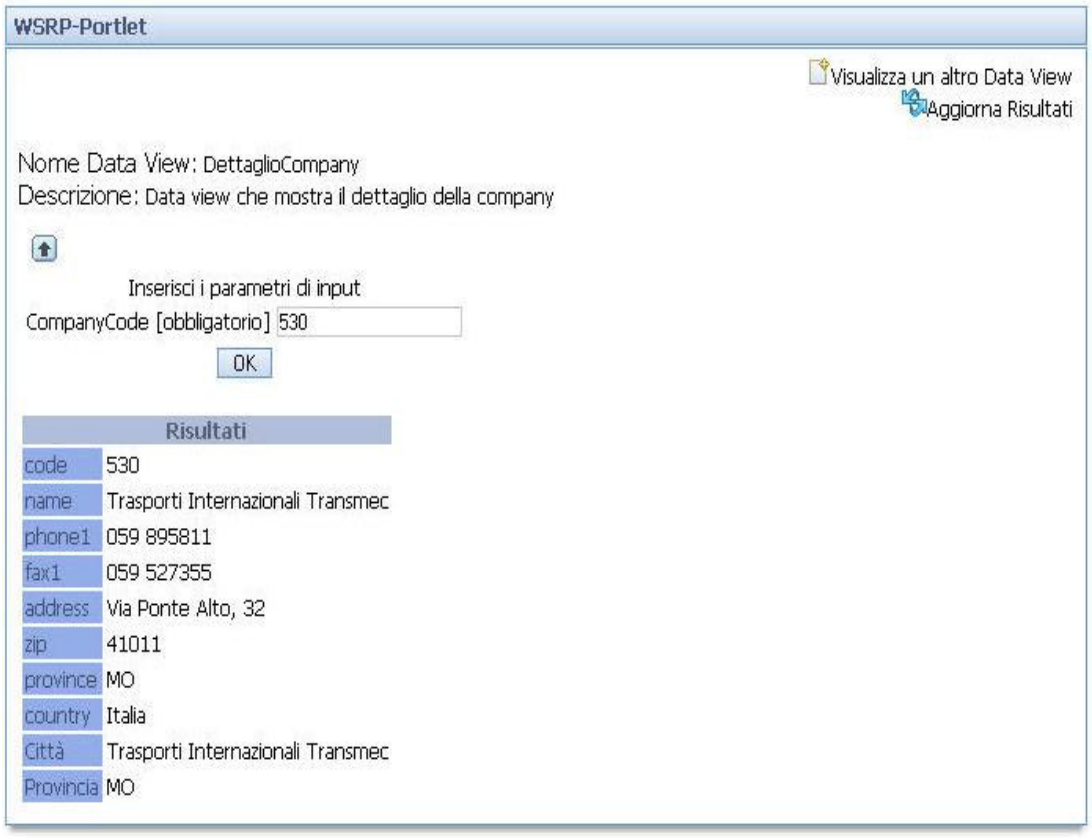

Figura 66: DataView con parametri di input obbligatori

- il DataView ha dei parametri obbligatori associati che non sono presenti in sessione e di cui non è stato specificato in fase di definizione il valore di default. Si avrà la visualizzazione di una pagina che ne richiederà l'inserimento:

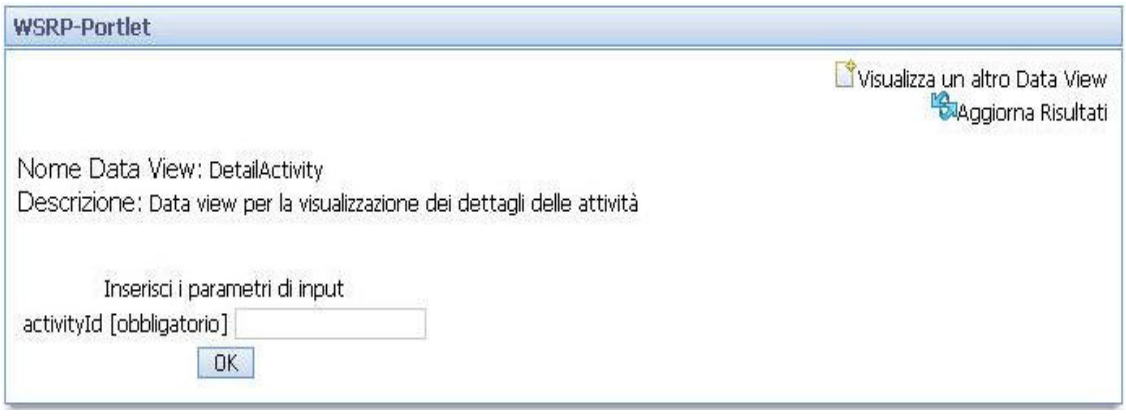

Figura 67: Inserimento parametri obbligatori

Da notare che a parte il caso in cui il DataView non abbia alcun parametro di input, l'utente potrà in ogni momento inserire tali parametri manualmente a seconda delle proprie esigenze. Se il DataView presenta dei risultati, per maggiore chiarezza viene utilizzato un "div a scomparsa" che permette di nascondere i parametri di input a volte abbastanza poco significativi, si pensi ad esempio ad un codice.
Vengono riportate di seguito due figure per illustrare ciò che è appena stato esposto.

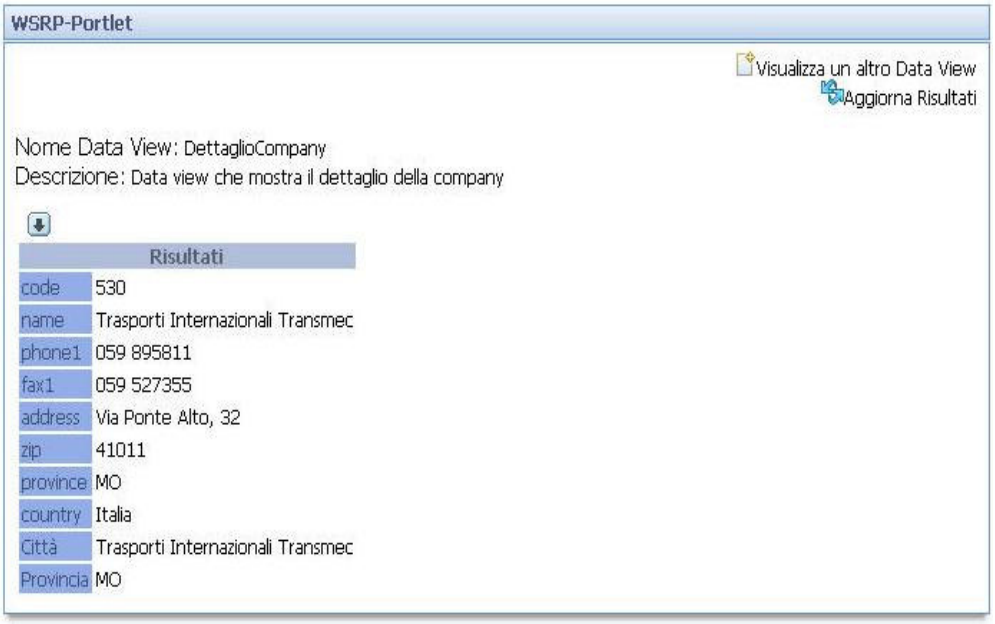

Figura 68: Parametri di input nascosti

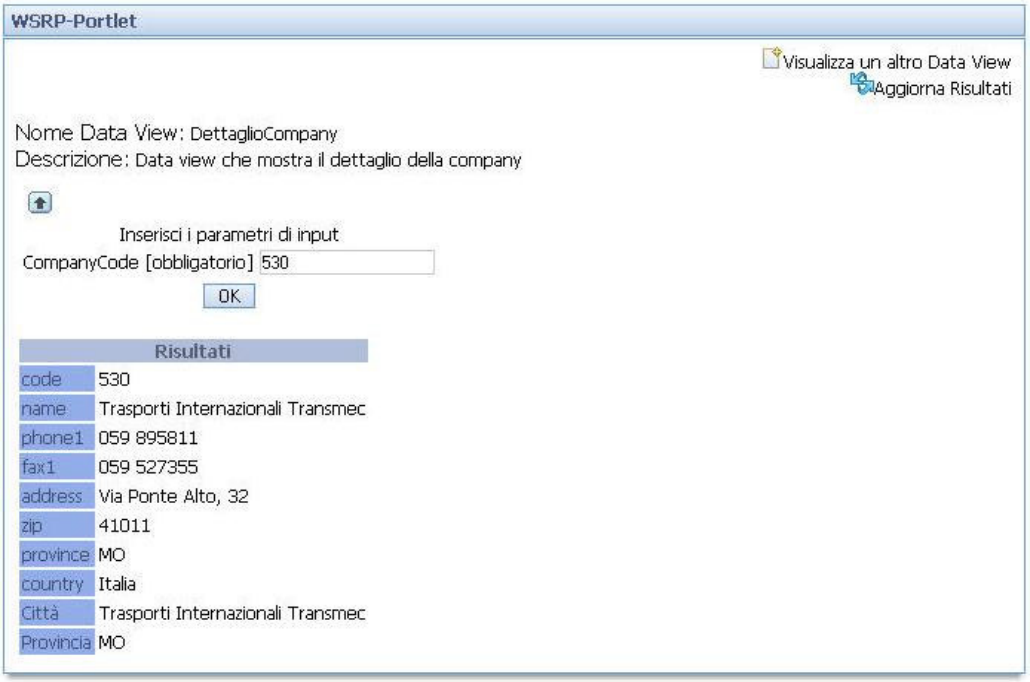

Figura 69: Parametri di input visualizzati

4.3.6 Azioni

Una peculiarità di questa applicazione sta proprio nella presenza delle azioni che tramite la loro definizione permetteranno di instaurare la comunicazione tra Portlet utilizzando come strumento la sessione (a livello `APPLICATION_SCOPE`).
Prima di entrare nel merito di questo argomento è doveroso segnalare una limitazione riscontrata in Liferay nell'uso della sessione a livello `PORTLET_SCOPE`.
Come già detto, le Portlet hanno due tipologie di sessione:

- `PORTLET_SCOPE` permette di inserire gli oggetti che saranno visibili solamente da ogni singola Portlet
- `APPLICATION_SCOPE` contenente gli oggetti che saranno visibili da tutte le Portlet della medesima applicazione

Utilizzando Liferay come portale Consumer, si è notato che non veniva fatta differenza tra i due tipi di sessione, infatti, gli oggetti venivano comunque inseriti a livello di `APPLICATION_SCOPE`. Questo comportava il fatto che gli oggetti in sessione a livello di Portlet venivano condivisi con le altre istanze della Portlet presenti nella pagina provocando malfunzionamenti nell'esecuzione.
La soluzione che è stata adottata prevede l'utilizzo dell'handle della Portlet il quale è un codice che permette l'identificazione univoca di ogni instanza.
Di seguito viene riportato il frammento di codice che permette di reperire l'handle della Portlet.

```
PortletWindow win =
 PortletUtils.getPortletWindow(actionRequest);
PortletEntity entity=win.getPortletEntity();
ObjectID obj = entity.getId();
String portletHandle = obj.toString();
```

La parte di codice sopra riportata è riferita al metodo `processAction` poiché per reperire la `portletWindow` viene utilizzato l'oggetto `actionRequest`. Negli altri metodi (`doView`, `doEdit`, `doHelp`) andrà utilizzato l'oggetto `renderRequest`.
Quando viene inserito un oggetto in sessione va associato ad esso un nome per identificarlo, la soluzione alla limitazione riscontrata il Liferay consiste nel fatto di concatenare l'handle della Portlet con il nome dell'oggetto così da avere univocità nei nomi utilizzati ed evitare di avere i problemi sopra esposti.

Come esempio viene riportato un frammento di codice che mostra ciò che è appena stato esposto:

```
renderRequest.getPortletSession().setAttribute(handle+"Name",
    testObject, PortletSession.PORTLET_SCOPE);
```

Di sopra è riportato il caso in cui viene inserito in sessione l'oggetto `testObject` associato ad un nome composto dall'handle e una stringa "Name".
La Portlet potrà reperire questo oggetto dalla sessione nel seguente modo:

```
Object obj = (Object)
 renderRequest.getPortletSession().getAttribute(handle+"Name",
 PortletSession.PORTLET_SCOPE);
```

Da notare che si è cercato di utilizzare la sessione il minor numero di volte possibile, preferendo, se possibile, passare gli oggetti tramite la `renderRequest` come mostrato di seguito.

```
renderRequest.setAttribute("Name",object);
```

Questo perché gli oggetti passati in questo modo saranno utilizzabili solamente per la durata della richiesta, alla successiva invocazione non saranno più presenti.
Viene riportato un esempio per fare chiarezza.
Supponiamo venga eseguito il metodo `doView` della Portlet e si abbia l'inserimento di un oggetto utilizzando la `renderRequest` (analogamente a quanto riportato precedentemente). Come di solito succede si avrà il dispatch verso una pagina jsp la quale sarà l'unica ad essere in grado di reperire tale oggetto infatti, nella successiva esecuzione di un qualunque metodo della Portlet la `renderRequest` non conterrà più l'oggetto inserito in precedenza.
Riprendiamo ora il discorso riguardante le azioni.
Un DataView potrà aver associato più azioni, la visualizzazione in modo contestuale ai risultati potrà essere di due tipologie:

- se in fase di definizione dell'azione è stata inserita un'icona, si avrà la seguente visualizzazione (il tool-tip rappresenta il nome associato all'azione)

Figura 70: Azione con icona

- altrimenti si avrà direttamente un link con il nome dell'azione

Figura 71: Azione senza icona

Come già detto nel precedente capitolo, nella definizione di un'azione andranno specificati i parametri associati, i quali saranno messi in sessione se l'azione verrà

selezionata. Nella fase di configurazione dell'azione, oltre a selezionare quali campi del DataView si vogliono associare, è possibile specificare opzionalmente un nome alternativo ad ogni parametro.
Questa è una fase molto delicata poiché il nome del campo o se specificato il nome alternativo saranno utilizzati per mettere in sessione i dati a loro associati.
I link e le icone associate alle azioni sono degli `ActionUrl` con associati tre parametri quali:

- `idAction` che identifica in modo univoco l'azione associata
- `numeroEntry` che indica quale entry della pagina corrente è stata selezionata
- `pagina` che indica la pagina corrente

```
ActionVO action = (ActionVO) listAction.get(i);
PortletURL url = renderResponse.createActionURL();
url.setParameter("pagina",Integer.toString(page));
url.setParameter("idAction",
                  Long.toString(action.getIdAzione()));
url.setParameter("numeroEntry",Integer.toString(j));
```

Alle azioni sono stati associati degli `ActionUrl` perché così facendo ogni volta che si avrà la selezione di un'azione verranno eseguite le seguenti operazioni:

- per prima cosa verrà eseguito il metodo `processAction` della Portlet associata all'azione selezionata. In questo metodo dopo avere stabilito che è stata selezionata un'azione si avrà:
  - grazie all'`idAction` presente come parametro associato al link dell'azione verranno reperite le informazioni associate, in particolare i parametri associati
  - grazie al `numeroEntry` e alla `pagina` verrà calcolata quale record del result set (del DataView) è stato selezionato
  - come ultima operazione, verranno settati in sessione i parametri con i relativi valori
- terminata l'esecuzione del `processAction` si avrà l'esecuzione dei metodi di render opportuni (`doView`, `doEdit`, `doHelp`) per ogni Portlet presente nella pagina. Da notare che se un DataView ha dei parametri di input associati, il cui

valore non è ancora stato inserito dall'utente, nel metodo `doView` vengono cercati in sessione.

I parametri associati all'azione vengono inseriti in sessione nel `processAction` poiché in caso di `ActionUrl` è il primo metodo ad essere eseguito seguito successivamente dall'esecuzione dei metodi di render opportuni. In questo modo, poiché il metodo `doView` è successivo al `processAction`, tutte le Portlet "vedranno" lo stesso contenuto in sessione (contenente anche i parametri associati all'azione selezionata).

Se invece di associare ad ogni azione un `ActionUrl` si fosse associato un `RenderUrl`, non verrebbe eseguito il `processAction` in più non è garantito che il primo metodo di render ad essere eseguito sia quello della Portlet associata all'azione selezionata. Così facendo si avrebbero due visioni diverse della sessione da parte delle Portlet:

- Portlet renderizzate prima dell'inserimento dei parametri associati all'azione in sessione
- Portlet renderizzate dopo l'inserimento dei parametri associati all'azione in sessione

Questo come si può intuire può portare a malfunzionamenti dell'applicazione:

- prendendo come validi dati non aggiornati
- vanificando la comunicazione tra Portlet. Questo può accadere quando una Portlet è in "ascolto" su parametri associati all'azione selezionata ma che ancora non sono stati inseriti in sessione.

Viene riportato di seguito un esempio di comunicazione tra Portlet tramite l'utilizzo delle azioni. In particolare, si nota che l'azione ha come nome "Detail", e permette di visualizzare (in un'altra Portlet) il dettaglio della società che viene selezionata.

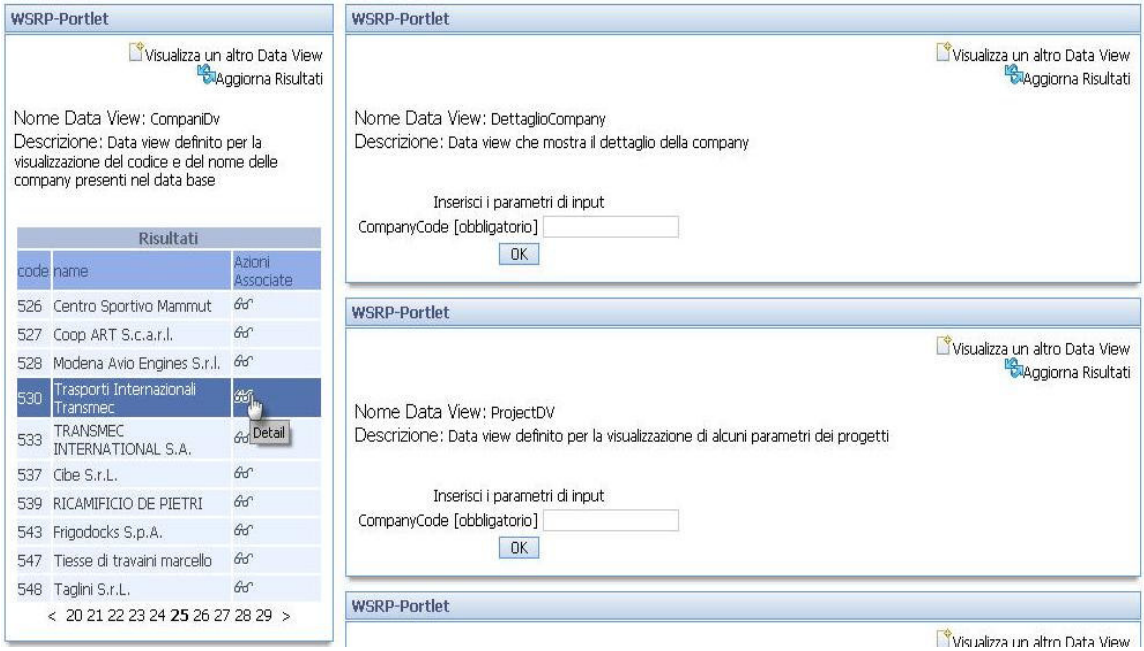

Figura 72: Selezione dell'azione

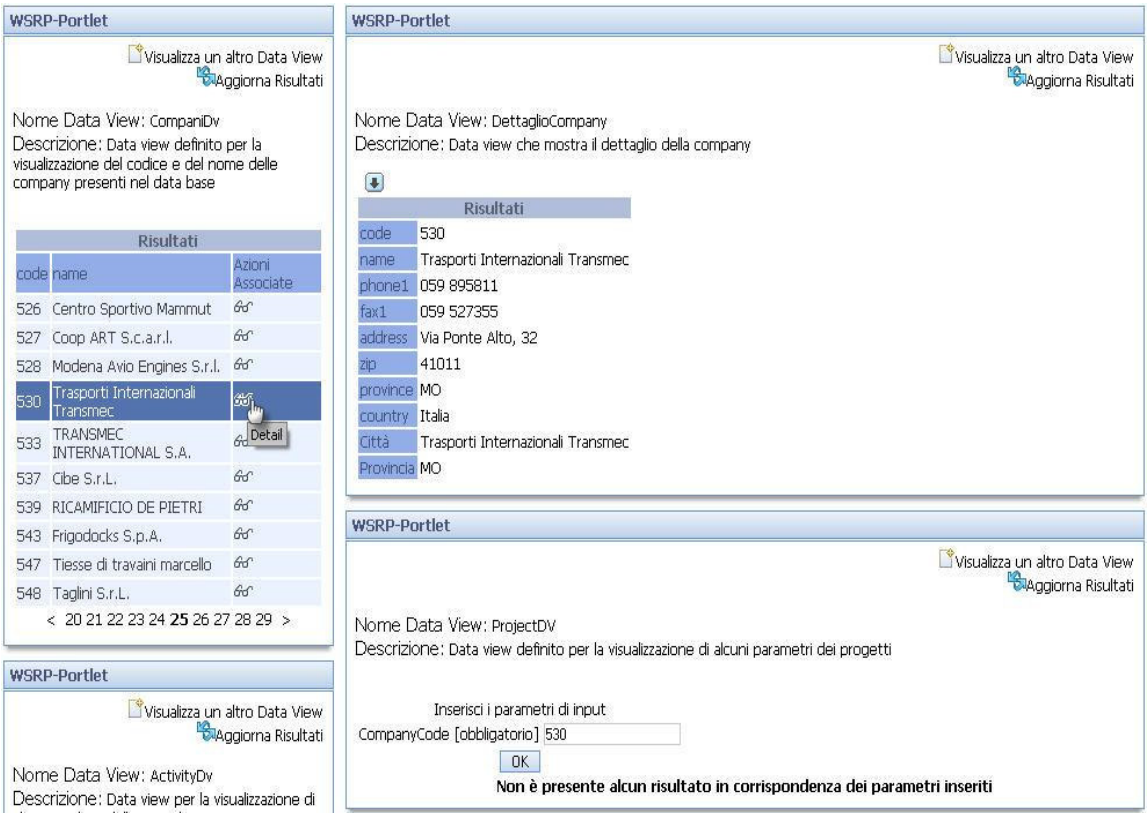

Figura 73: Ricezione dei parametri associati all'azione

Come si può notare dalla figura di sopra, l'azione ha determinato il cambiamento di due istanze della Portlet:

- in quella in alto a destra si ha il dettaglio della company
- in quella in basso a destra viene comunicato che in corrispondenza della company selezionata non è presente alcun risultato.

Per quanto riguarda la comunicazione tra Portlet utilizzando come portale Consumer uPortal è stata incontrata qualche difficoltà come riportato di seguito.

Come già detto in precedenza vi sono due tipi di eventi che coinvolgono le Portlet:

- `actionUrl` (ad esempio il submit di un form) per i quali viene eseguito prima il `processAction` e successivamente il metodo di render opportuno (`doView`, `doHelp`, `doEdit`)
- `renderUrl` per i quali viene solamente rinfrescata la Portlet eseguendo il metodo di render in base alla modalità di visualizzazione corrente.

Il problema riscontrato è che uPortal effettua le operazioni solamente sulla Portlet che ha ricevuto un evento mentre le altre vengono visualizzate senza eseguire il render prendendo le informazioni in cache. In generale questo comportamento può portare a vantaggi di natura prestazionale poiché vengono eseguite le operazioni solamente della Portlet interessata. Nel nostro caso invece questo comportamento vanifica la comunicazione tra Portlet poiché il Portale inserisce gli oggetti associati all'azione in sessione ma non rinfresca le altre Portlet presenti nella pagina.

Anche impostando nel file descriptor della Portlet l'`expiration-time` a zero per evitare il caching tale problema rimane.

Questa è una grande limitazione di uPortal perché potrebbe accadere che una Portlet si aspetti un parametro che è stato messo in sessione ma, poiché non è stata ri-renderizzata non riceva tale parametro e quindi venga vanificata la comunicazione inter-portlet.

Per cercare di risolvere tale problema è stato inserito un link (che implementa un `renderUrl`) per forzare manualmente il render della Portlet così da rendere possibile anche se in modo più macchinoso la comunicazione.

Viene riportata di seguito un esempio chiarificatore dei concetti sopra descritti.

Per prima cosa si ha la selezione da parte dell'utente di un'azione, in questo caso con nome detail.

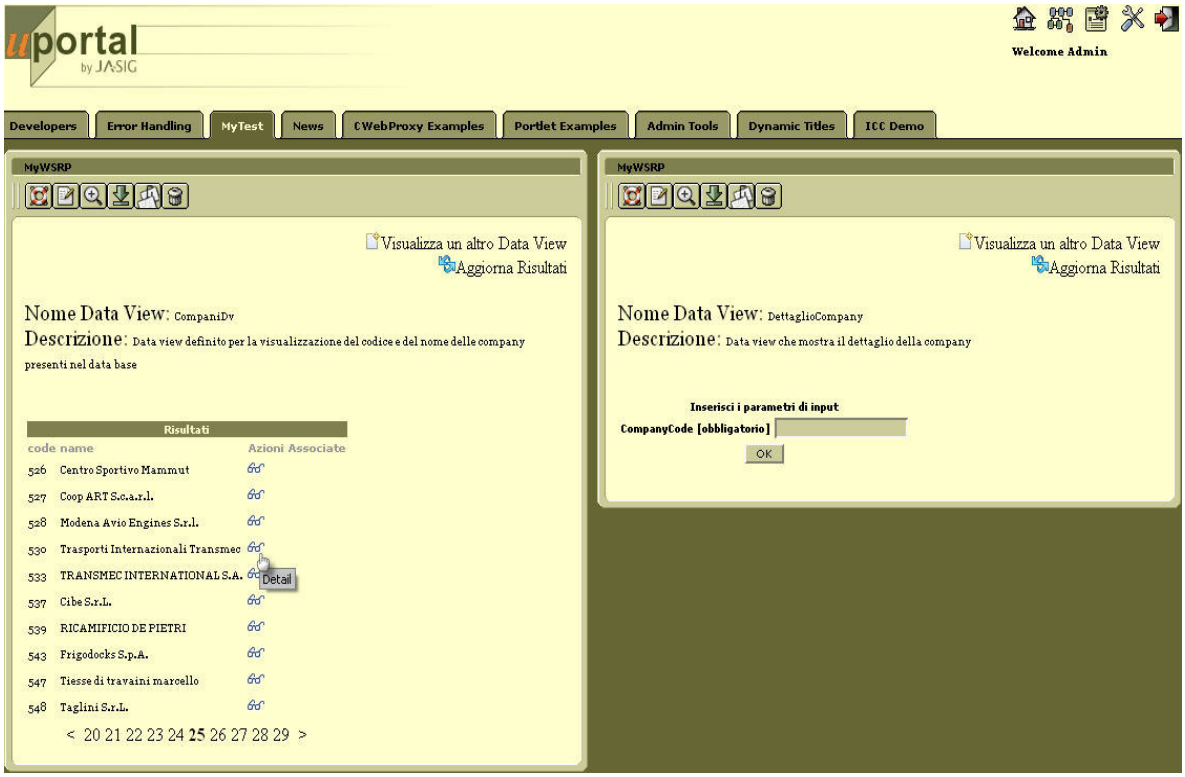

Figura 74: Selezione dell'azione in uPortal

Come si può notare dalla figura sottostante la Portlet di destra che doveva ricevere il parametro dall'azione, non è stata aggiornata e quindi non è possibile la visualizzazione dei risultati associati.

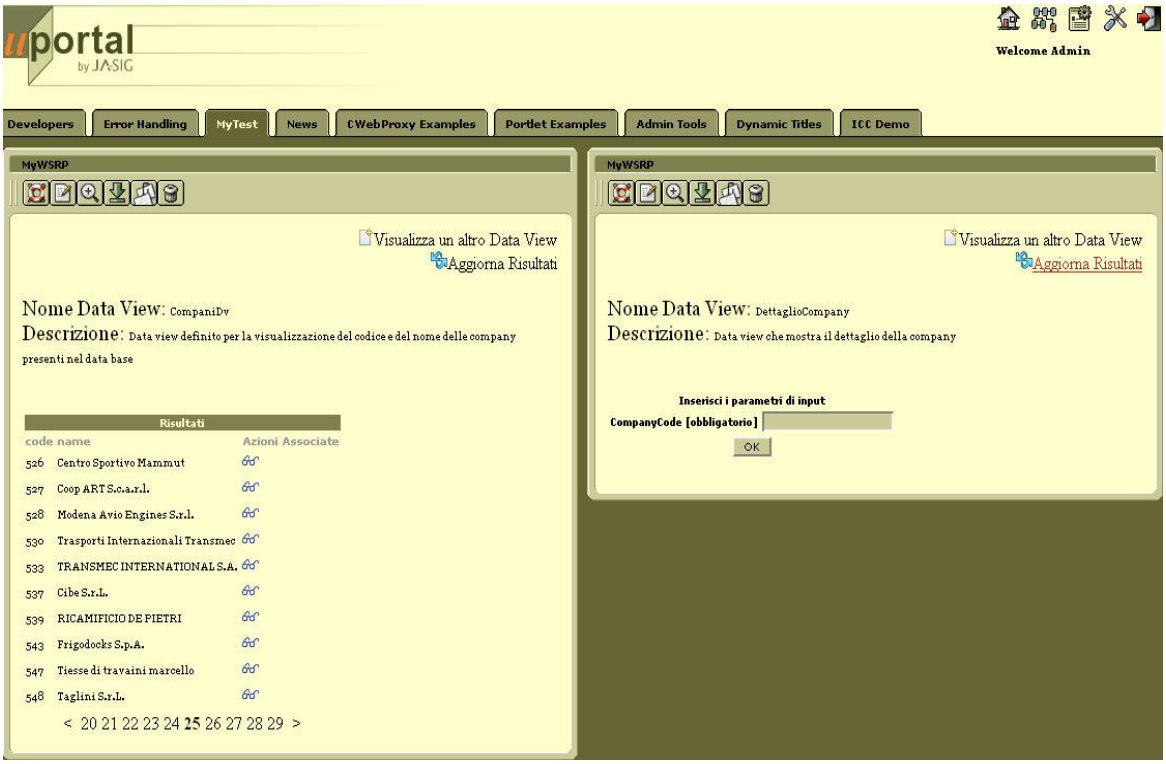

Figura 75: Mancato aggiornamento delle Portlet

Solamente cliccando sul link “Aggiorna Risultati” è possibile la visualizzazione dei risultati associati.

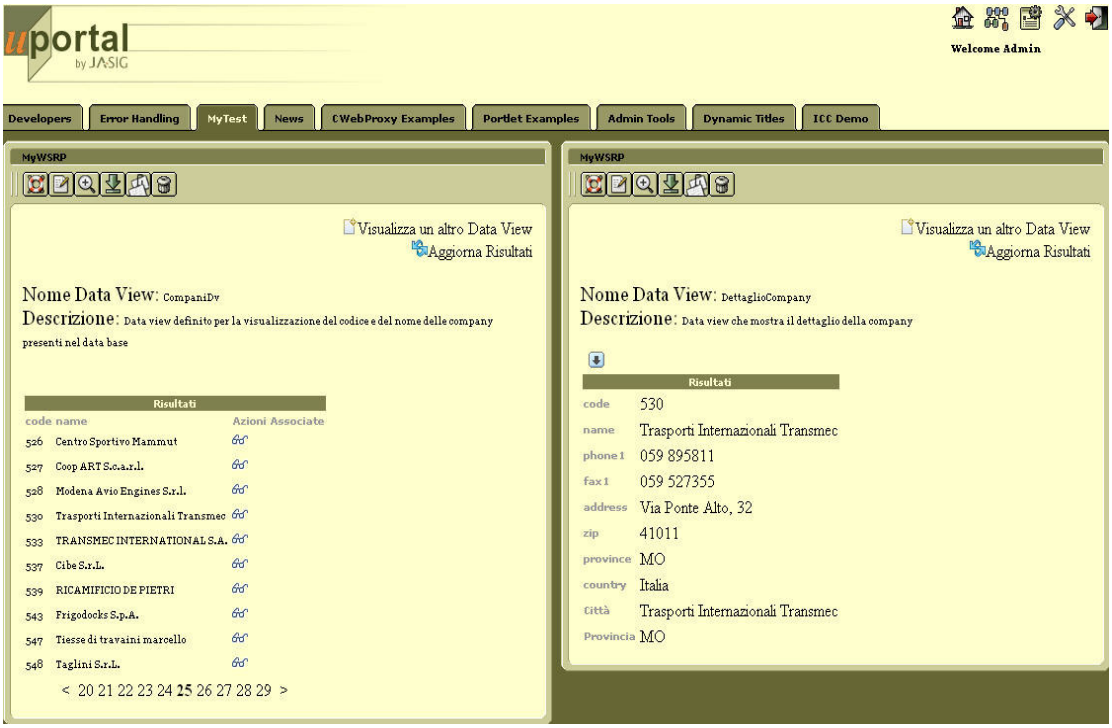

Figura 76: Risultati aggiornati

In questo paragrafo si è dato risalto alla comunicazione tra Portlet utilizzando come mezzo la sessione.
Nello svolgimento del tirocinio è stata sviluppata una Portlet con le medesime funzionalità della Portlet di pubblicazione WSRP ma con l'obiettivo di dover essere eseguita solamente in modo locale ed utilizzando come portale Liferay 4.0.
In questo caso per instaurare la comunicazione tra Portlet non è necessario l'utilizzo della sessione poiché è possibile utilizzare una tag-lib nativa di Liferay.
Questa particolare tag-lib permette di avere nelle pagina JSP degli `actionUrl` e `renderUrl` indirizzati ad una o più specifiche Portlet. In questo modo sarà possibile instaurare una comunicazione punto-punto tra Portlet evitando di utilizzare come mezzo di comunicazione la sessione (ovviamente a livello `APPLICATION_SCOPE`) che di natura viene condivisa da tutte le Portlet appartenenti alla medesima applicazione. Questo, da un lato permette una migliore comunicazione tra Portlet, dall'altro è un elemento di forte dipendenza del portale (Liferay), infatti se utilizzassimo la medesima Portlet in un altro portale come ad esempio uPortal, tale comunicazione non sarebbe possibile poiché la tag-lib che si sta utilizzando è specifica per il portale Liferay.
Di fatto, il `liferay:actionURL` è simile al canonico `portlet:actionURL` eccezione fatta per alcuni attributi addizionali come:

- `portletName` permette di specificare il nome della Portlet a cui riferire l'`actionUrl`;
- `encrypt` se settato a true permette di cifrare tutti i valori dei parametri;
- `anchor` se settato a false per non avere il refresh della Portlet.
- `WindowState` che permette di settare lo stato della finestra associata alla Portlet (`maximized`, `minimized`, `normal`);
- `PortletMode` che permette di settare la modalità della Portlet (`edit`, `view`, `help`).

Viene riportato di seguito un frammento di tag-lib che definisce i parametri riferiti all'`ActionUrl`.

```
    <tag>
         <name>actionURL</name>
         <tagclass>
              com.liferay.taglib.portlet.ActionURLTag
         </tagclass>
```

```
          <teiclass>
               com.liferay.taglib.portlet.ActionURLTei
          </teiclass>
          <bodycontent>JSP</bodycontent>
          <attribute>
               <name>windowState</name>
               <required>false</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
               <name>portletMode</name>
               <required>false</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
               <name>portletName</name>
               <required>false</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
               <name>anchor</name>
               <required>false</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
          <attribute>
               <name>encrypt</name>
               <required>false</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
     </tag>
```

Del tutto analogo è il discorso per il `liferay:renderURL`:

```
<tag>
          <name>renderURL</name>
          <tagclass>
               com.liferay.taglib.portlet.RenderURLTag
          </tagclass>
          <teiclass>
               com.liferay.taglib.portlet.RenderURLTei
```

```
        </teiclass>
        <bodycontent>JSP</bodycontent>
        <attribute>
             <name>windowState</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
             <name>portletMode</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
             <name>portletName</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
             <name>anchor</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
             <name>encrypt</name>
             <required>false</required>
             <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
```

Infine si ha la definizione dei parametri che verranno passati nell'`actionURL` o `renderURL`, basterà indicare il nome del parametro e il valore corrispondente:

```
<tag>
        <name>param</name>
        <tagclass>com.liferay.taglib.util.ParamTag</tagclass>
        <bodycontent>JSP</bodycontent>
        <attribute>
             <name>name</name>
             <required>true</required>
             <rtexprvalue>true</rtexprvalue>
```

```
          </attribute>
          <attribute>
               <name>value</name>
               <required>true</required>
               <rtexprvalue>true</rtexprvalue>
          </attribute>
 </tag>
```

Come esempio viene riportato un frammento di codice che permette di instaurare questa comunicazione tra due Portlet:

```
String
portlet=”PublicationPortlet_WAR_PublicationLiferay_Instance_NO
Ci”;
<a href="<liferay:renderURL portletName="<%=portlet%>">
  <liferay:param name="nome" value="Parametro"/>
  </liferay:renderURL>">Invia Parametro</a>
```

Come si può notare il nome della portlet è composto da varie parti:

- PublicationPortlet è il nome della Portlet;
- WAR viene inserito dal portale quando viene deployata la Portlet come war file;
- PublicationLiferay è il nome dell’applicazione;
- Instance_NOCi è l’identificativo che il portale inserisce alla Portlet (per poter indirizzare le richieste alle Portlet opportune) quando è possibile avere più istanze della medesima Portlet nella stessa pagina del portale.

Nella Portlet che riceve il parametro basterà inserire nella pagina JSP di ricezione il seguente frammento di codice:

```
String parametro = request.getParameter(“Parametro”);
```

4.3.7 CSS

Come si è potuto notare nei precedenti screen-shot, la grafica associata alla Portlet è molto semplice, si è scelto, in conformità con lo standard WSRP di utilizzare le classi

definite nella specifica così da avere una visualizzazione uniforme dei contenuti tra vari Consumer.

L'aggregazione di pagine ha come obiettivo ottenere un comune look-and-feel delle Portlet contenute in una pagina. Questo non va ad influire solamente nelle decorazioni ma anche nei contenuti stessi. Utilizzando un CSS comune per ogni Portlet e definendo un insieme di stili standard si ottiene un comune look-and-feel senza richiedere che le Portlet generino markup dipendente dal Consumer.

5. Conclusioni e Sviluppi futuri

Il periodo di tirocinio è stato molto utile per toccare con mano ciò che è il mondo del lavoro, inoltre, è stata un'occasione per applicare sul campo le nozioni e i concetti appresi durante questi cinque anni di Università.
Un aspetto a mio riguardo molto importante è che, pur avendo fatto uno stage aziendale, è stato possibile affrontare un argomento (WSRP) nuovo di cui è scarsa o incompleta la documentazione. Questo ha permesso di mettermi alla prova nell'affrontare questi problemi ed avere maggiori soddisfazioni nel superamento degli stessi. Molto utile se non indispensabile è stato l'utilizzo di Forum e Newsletter al fine di interagire direttamente con persone competenti. Affrontando lo studio dello standard WSRP si è reso necessario l'utilizzo dei Portali, questo mi ha permesso di selezionarne alcuni e di studiarne il funzionamento cogliendone pregi e difetti.
Per quanto riguarda l'argomento Struts e Struts Bridge si è imparato a sviluppare Web-Application utilizzando il pattern MVC e ad approfondire il linguaggio di programmazione JAVA.
Gli sviluppi futuri associati all'applicazione sviluppata seguendo lo standard WSRP sono i seguenti:

- *testare altri Portali "Consumer" e " Producer"* al fine di verificarne appieno le capacità di integrazione in modo indipendente dall'implementazione e dall'architettura
- *integrare nell'applicazione l'utilizzo dello standard UDDI* così da avere il discovery del servizio in modo automatico da parte dei Portali "Consumer"
- *integrare il supporto per l'autenticazione degli utenti* così da poter associare privilegi ed, in base ad essi, avere diversi tipi di accesso all'applicazione
- *integrare nell'applicazione lo standard WS-Security*, per ottenere lo scambio di messaggi SOAP in modo "sicuro"
- *verificare se nella nuova specifica WSRP 2.0* che verrà presto rilasciata permangono le lacune della versione attuale
- *sviluppo di applicazioni che operino come "Producer" in uno scenario WSRP rendendo fruibili le proprie funzionalità attraverso un'interfaccia di portale.*

6. Glossario

Apache Struts	Apache Struts è un framework open source per lo sviluppo di applicazioni web su piattaforma J2EE seguendo il pattern MVC.
AXIS	AXIS (Apache eXtendable Interaction System) il quale è l'implementazione di SOAP per Java fornita dall'Apache Group.
Css	CSS (Cascading Style Sheet) sono un insieme di regole redatte dal W3C (World Wide Web Consortium) per definire l'aspetto delle pagine HTML e XHTML
DBMS	Database Management System è un sistema informatico progettato per gestire un Database, ovvero un insieme di numerosi dati strutturati
Deployare	Azioni mediante le quali è possibile inserire l'applicazione (es: Portlet) nel contesto di un Portale.
JSR-168	Java Specification Request è la specifica Java riguardante le Portlet
MVC	Model-View-Controller è il nome di un design pattern fondamentale nello sviluppo di sistemi software object-oriented; è il pattern a cui fa riferimento Struts.
Portale	Un portale Web è un sito web che costituisce un punto di partenza, una porta di ingresso ad un gruppo consistente di risorse di Internet o di una Intranet.
Portals Bridges	I Portals Bridges permettono ad applicazioni sviluppate con i più comuni web framework come Struts, JSF, PHP, Perl e Velocity di essere utilizzate come Portlet JSR-168 compliant
Portlet	Componente web basato sulla tecnologia Java, gestito da un Portlet Container che esegue richieste da parte dell'utente e genera contenuti dinamici
Portlet Container	Componente che si occupa si ricevere le richieste dal portale e di reindirizzarle alle Portlet opportune.
Portlet Handle	Codice che identifica in modo univoco ogni instanza della Portlet
SOA	Service-Oriented Architecture viene indicata un'architettura software atta a supportare l'uso di servizi (come ad esempio i

	web service) per soddisfare le richieste degli utenti così da consentire l'utilizzo delle singole applicazioni come componenti del processo di business.
SOAP	Simple Object Access Protocol è un protocollo leggero per lo scambio di messaggi tra componenti software in linguaggio XML.
Struts Bridge	Permette il deploy di una Web-Application sviluppata con Struts come Portlet.
UDDI	L'UDDI (acronimo di Universal Description Discovery and Integration) è un registry (ovvero una base dati ordinata ed indicizzata), basato su XML ed indipendente dalla piattaforma hardware, che permette la pubblicazione di servizi offerti su Internet.
WAR	Web Application Archive è un semplice file compresso contenente le classi, le librerie ed i file di configurazione (scritti in XML di norma) utilizzati dalla Web-Application.
Web Service	Secondo la definizione data dal W3C un Web service è un sistema software progettato per supportare l'interoperabilità tra applicazioni diverse in rete
WSDL	WSDL (acronimo di Web Services Description Language) è un linguaggio formale in formato XML utilizzato per la creazione di "documenti" per la descrizione di Web Service.
WSRP	Acronimo di Web Services for Remote Portlet. È lo standard che definisce l'interazione tra Producer e Consumer al fine di permettere l'utilizzo da parte del Consumer di Portlet remote
WSRP4J	Implementazione in Java di un Producer secondo la specifica WSRP.
XML	Acronimo di eXtensible Markup Language permette la definizione (sintattica) della struttura dei documenti sul web. È lo standard di trasmissione dati tra applicazioni diverse.

7. Bibliografia

Portali

1	http://portals.apache.org/pluto/ sito di riferimento per il portale Pluto
2	http://www.developer.com/java/web/article.php/3554396 per lo sviluppo di applicazioni per il portale Pluto
3	http://www.liferay.com sito di riferimento molto ricco di documentazione e con un forum molto frequentato sul portale Liferay
4	http://www.uportal.org/ sito di riferimento per uPortal
5	http://www.fair-portal.hull.ac.uk/downloads/uPortalGuide1.pdf#search=%22uPortal%20portal%20%22 come guida di base per l'utilizzo di uPortal

Portlet

6	http://jcp.org/en/jsr come riferimento della specifica JSR-168
7	http://developers.sun.com/prodtech/portalserver/reference/techart/jsr168 come riferimento per la specifica JSR-168 e come tutorial per lo sviluppo di Portlet
8	http://www.javaworld.com/javaworld/jw-08-2003/jw-0801-portlet.html come introduzione alla specifica delle Portlet
9	http://web.princeton.edu/sites/isapps/jasig/2004WinterNewOrleans/Presentations/ come presentazione riguardo le Portlet
10	**"Portlet and Apache portals"** di Stefan Hepper, Peter Fisher, Stephan Hesmer, Richard Jacob, David Sean Taylor, Mike McCallister - Manning Publications -
11	http://www.mokabyte.it come riferimento in Italiano per alcuni articoli molto interessanti
12	**"Portal Development With Open Source Tools"** di W.Clay Richardson, Donald Avondolio, Joe Vitale, Peter Len, Kevin T Smith - WROX Publications -
13	http://www.onjava.com/pub/a/onjava/2005/09/14/what-is-a-portlet.html?page=1 come tutorial per lo sviluppo di Portlet JSR-168 compliant

WSRP

14	http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrp sito di riferimento per lo standard WSRP
15	http://www-128.ibm.com/developerworks/library/ws-wsrp/ come introduzione allo standard WSRP
16	http://portals.apache.org/wsrp4j/ come riferimento per il progetto WSRP4J
17	http://xml.coverpages.org/wsrp.html come raccolta di articoli sul WSRP
18	http://www.theserverside.com/news/thread.tss?thread_id=39077 come raffronto tra standard JSR-168 e WSRP

Altri

19	http://www.mokabyte.it/2004/01/jstruts-1.htm "Sviluppare applicazioni J2EE con Jakarta Struts" di Alfredo Larotonda per il capitolo dedicato al framework Jakarta Struts.
20	http://portals.apache.org/bridges/multiproject/portals-bridges-struts/index.html riferimento per l'utilizzo dello Struts Bridge
21	http://www.ja-sig.org/wiki/display/PLT/Struts+Bridge come tutorial per l'utilizzo dello Struts Bridge
22	http://dsc.sun.com/learning/javaoneonline/2005/webtier/TS-3374.pdf#search=%22struts%20bridge%22 come articolo per le peculiarità dello Struts Bridge

8. Ringraziamenti