Università degli Studi di Napoli Federico II

Facoltà di Ingegneria

Corso di laurea in Ingegneria Informatica

Titolo Tesi:

Una soluzione Open Source per lo sviluppo di Web Applications: il framework “Struts”

Relatore
Prof. Antonio d’Acierno

Candidato
Raffaele Musto
Matr. 41/3076

Anno Accademico 2004/2005

CAPITOLO 1

CAPITOLO 2

WEB APPLICATIONS E STRUTS

1.1 Introduzione

Sebbene il Web sia nato come architettura per la disseminazione di contenuti digitali multimediali, la sua rapida evoluzione ne ha reso evidente l'efficacia per la fornitura di servizi e applicazioni avanzate, sia in ambito Internet sia nel contesto di sistemi informativi aziendali in architettura intranet. La maturazione dei protocolli e dei formati di base, infatti, ha reso possibile implementare con tecnologie Web tutti quei requisiti di sicurezza, efficienza e affidabilità che tradizionalmente erano assicurati solo da piattaforme chiuse e proprietarie.

Volendo essere minimalisti, una applicazione web è un programma che risiede su di un Web server e fornisce pagine statiche o dinamiche in un linguaggio a marcatori

(tipicamente HTML) in risposta alla richiesta di un utente, il quale interagisce con essa, inviando richieste tipicamente attraverso un browser.

Naturalmente quando si parla di applicazioni web non ci si riferisce solo alla gestione dei contenuti, ma alla esecuzione di procedure complesse, spesso delicate o, per usare il gergo degli addetti ai lavori, *mission critical*, che possono richiedere l'integrazione tra sistemi software eterogenei preesistenti. I domini in cui le applicazioni Web vengono ormai ampiamente utilizzate sono numerosi e variegati: commercio elettronico, home-banking e servizi finanziari, e-government, editoria digitale, marketing e servizi post-vendita, groupware e knowledge management.

Jakarta Struts è un framework open source per lo sviluppo di applicazioni web, basato, come riportato sul sito ufficiale http://struts.apache.org , su tecnologie standard ed oramai consolidate, come: Java Servlets, JavaBeans, ResourceBundles, and XML. Pensato per facilitare sviluppatori di web application pienamente aderenti alla piattaforma J2EE della SUN, fornisce una infrastruttura unica, versatile e solida che permette di costruire applicazioni web di grosse dimensioni e di alta qualità con una forte riduzione dei tempi di sviluppo, agevolando la suddivisione dello sviluppo del progetto fra vari sotto-team che potranno lavorare in parallelo. È facile immaginare, come ciò si traduca in grossi vantaggi economici per chi lo adotta ed il perché in breve tempo stia diventando il framework più usato dalla comunità di sviluppatori Java.

Il framework Struts è stato realizzato per la prima volta da Craig R. McClanahan ed è stato poi donato alla Apache Software Foundation (ASF) nel 2000. Il progetto inizialmente inquadrato come sotto-progetto di Apache Jakarta, dato il suo sviluppo, è diventato oggi un progetto a se che coinvolge in maniera cospicua diversi soggetti in tutto il mondo che contribuiscono alla buona riuscita del framwork.

Scopo di questo lavoro di tesi da me svolto, è una analisi del core di questo potente framework, per capire le ragioni del suo successo. Lavoro che sicuramente lascierà molte strade aperte, data la continua evoluzione di Struts. A supporto della mia trattazione e per fornire una guida pratica all'utilizzo del framework, il lavoro presente lavoro si conclude con l'esame di un caso di studio e la realizzazione di una web application dimostrativa.

In una definizione più dettagliata, potremmo dire che Struts è un MVC web application framework, ovvero è un framework per lo sviluppo di applicazioni web J2EE basato sul pattern Model-View-Controller. Prima di addentrarci nel mondo di Struts, per capire meglio di cosa si tratta e la sua valenza nel mondo della progettazione delle web application, diamo una breve spiegazione di alcuni concetti introdotti fino ad ora.

1.2 L'architettura J2EE

L'architettura proposta dalla piattaforma J2EE divide le applicazioni enterprise in tre strati applicativi fondamentali: componenti, contenitori e connettori.
Il modello di programmazione prevede lo sviluppo di soluzioni utilizzando componenti a supporto delle quali fornisce quattro tecnologie fondamentali:

Enterprise Java Beans;
Servelet ;
Java Server Pages ;
Applet.

La prima delle tre denotata con EJB fornisce supporto per la creazione di componenti server-side che possono essere generate indipendentemente da uno specifico database, da uno specifico transaction server o dalla piattaforma su cui gireranno. La seconda, servlet, consente la costruzione di servizi web altamente performanti ed in grado di funzionare sulla maggior parte dei web server ad oggi sul mercato. La terza, JavaServer Pages o JSP, permette di costruire pagine web dai contenuti dinamici utilizzando tutta la potenza del linguaggio java. Le applet, anche se sono componenti client-side rappresentano comunque tecnologie appartenenti allo strato delle componenti.
Il secondo strato è rappresentato dai contenitori ovvero supporti alle tecnologie appartenenti al primo strato logico della architettura. La possibilità di costruire

contenitori rappresenta la caratteristica fondamentale del sistema in quanto fornisce ambienti scalari con alte performance.

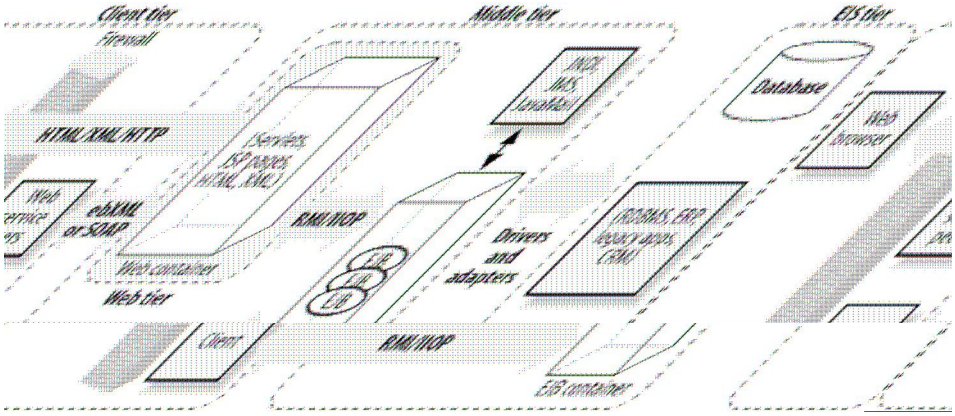
Infine i connettori consentono alle soluzioni basate sulla tecnologia J2EE di preservare e proteggere investimenti in tecnologie già esistenti fornendo uno strato di connessione verso applicazioni-server o middleware di varia natura: dai database relazionali con JDBC fino ai server LDAP con JNDI.

Gli application-server compatibili con questa tecnologia riuniscono tutti e tre gli strati in un una unica piattaforma standard e quindi indipendente dal codice proprietario, consentendo lo sviluppo di componenti in grado di girare in qualunque container compatibile con J2EE indipendentemente dal fornitore di software, e di interagire con una vasta gamma di servizi pre-esistenti tramite i connettori.

Ad appoggio di questa soluzione, la Sun mette a disposizione dello sviluppatore un numero elevato di tecnologie specializzate nella soluzione di “singoli problemi”. Gli EJBs forniscono un modello a componenti per il “server-side computing”, le Servlet offrono un efficiente meccanismo per sviluppare estensioni ai Web Server in grado di girare in qualunque sistema purché implementi il relativo container. Infine Java Server Pages consente di scrivere pagine web dai contenuti dinamici sfruttando a pieno le caratteristiche di java.

Un ultima considerazione, va fatta sul modello di approccio al problem-solving: la suddivisione netta che la soluzione introduce tra logiche di business, logiche di client, e logiche di presentazione consente un approccio per strati (tier) al problema, garantendo ordine nella progettazione e nello sviluppo di una soluzione, fornendo una separazione delle responsabilità, il riuso dei componenti, una migliore scalabilità, e molti altri aspetti vantaggiosi.

Uno schema degli strati funzionali di un applicazione web è mostrato in figura, anche se non tutte le applicazioni conterranno tutti gli strati:



1.2.1 Lo strato client

Appartengono allo strato client le applicazioni che forniscono all'utente una interfaccia semplificata verso il mondo enterprise e rappresentano quindi la percezione che l'utente ha della applicazione J2EE. Tali applicazioni si suddividono in due classi di appartenenza : le applicazioni web-based e le applicazioni non-web-based.

Le prime sono quelle applicazioni che utilizzano il browser come strato di supporto alla interfaccia verso l'utente ed i cui contenuti vengono generati dinamicamente da Servlet o Java Server Pages o staticamente in HTML.

Le seconde (non-web-based) sono invece tutte quelle basate su applicazioni stand-alone che sfruttano lo strato di rete disponibile sul client per interfacciarsi direttamente con la applicazione J2EE senza passare per il Web-Tier.

Nel caso di Struts, il tipo di client più comune è il browser web, ma precisiamo fin da ora che non è una regola.

1.2.2 Lo strato web

Lo strato web consente allo strato client di comunicare e interagire con la logica dell'applicazione, la quale risiede su altri strati. In web applications tradizionali, non è raro che parte della logica sia presente in questo strato, ma in applicazioni di scala più ampia, a livello "enterprise", il web tier agisce come "traduttore" effettuando un

mapping delle richieste HTTP verso invocazioni di servizi sullo strato successivo, il middle tier.
Il web tier è il collante che lega le applicazioni client al nucleo fondamentale di sistemi business. I componenti che risiedono nello strato web consentono agli sviluppatori di estender le funzionalità fondamentali di un web service. Nel caso di Struts, ciò avviene tramite coponenti dell'infrastruttura che vanno in esecuzione all'interno di un servlet container.

1.2.3 Lo strato business

Nell'ambito di una applicazione enterprise, è questo lo strato che fornisce servizi specifici: gestione delle transazioni, controllo della concorrenza, gestione della sicurezza, ed implementa inoltre logiche specifiche circoscritte all'ambito applicativo ed alla manipolazione dei dati. Mediante un approccio di tipo Object Oriented è possibile decomporre tali logiche o logiche di business in un insieme di componenti ed elementi chiamati "*business object*".
La tecnologia fornita da Sun per implementare i business object è quella che inprecedenza abbiamo indicato come EJBs (Enterprise Java Beans). Tali componenti si occupano di:

- Ricevere dati da un client, processare tali dati (se necessario), inviare i dati allo strato EIS per la loro memorizzazione su base dati;
- Acquisire dati da un database appartenente allo strato EIS, processare tali dati (se necessario), inviare tali dati al programma client che ne abbia fatto richiesta.

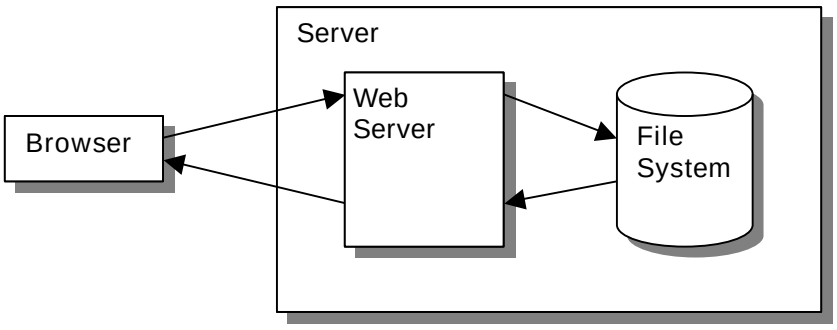
1.2.4 Lo strato EIS (Enterprise Information System)

Le applicazioni enterprise implicano per definizione l'accesso ad altre applicazioni, dati o servizi sparsi all'interno delle infrastrutture informatiche del fornitore di servizi. Le informazioni gestite ed i dati contenuti all'interno di tali infrastrutture

rappresentano la “ricchezza” del fornitore, e come tale vanno trattati con estrema cura garantendone la integrità.
Il business tier cominica con i componenti presenti nello strato EIS utilizzando protocolli specfici per la risorsa. Per esempio, per comunicare con un database relazionale normalmente utilizzerà un driver JDBC.

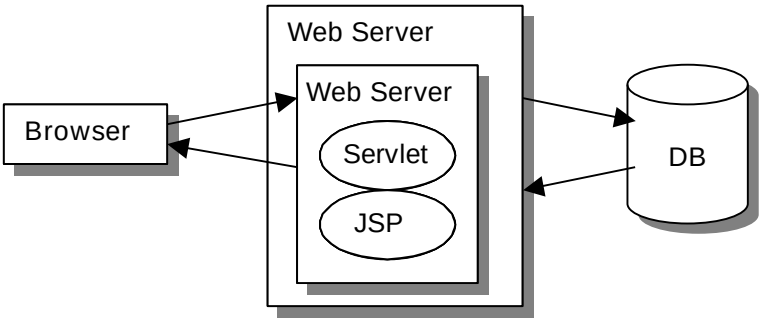
1.2.5 ...e Struts?

All’interno di questo scenario appena descritto, l’infrastruttura di Struts si colloca all’interno del web tier. Le applicazioni Struts sono ospitate da un web conteiner e possono far uso di servizi forniti dal container stesso, quali la gestione delle richieste tramite i protocolli HTTP e HTTPS. Questo dà agli sviluppatori la libertà di concentrarsi sulla realizzazione della logica di business che risolve i problemi reali della loro applicazione, costituendo il valore aggiunto delle web applications Struts.
All’interno dello strato web, il web-server gioca un ruolo fondamentale. In ascolto su un server, riceve richieste da parte del browser (client), le processa, quindi restituisce al client una entità o un eventuale codice di errore come prodotto della richiesta.



Le entità prodotte da un web-server possono essere entità statiche o entità dinamiche. Una entità statica è ad esempio un file html residente sul file-system del server. Rispetto alle entità statiche, unico compito del web-server è quello di recuperare la risorsa dal file-system ed inviarla al browser che si occuperà della visualizzazione dei contenuti.
Quello che più ci interessa sono invece le entità dinamiche, ossia entità prodotte dalla esecuzione di applicazioni eseguite dal web-server su richiesta del client.

Il modello proposto nella immagine precedente ora si complica in quanto viene introdotto un nuovo grado di complessità nella architettura del sistema che, oltre a fornire accesso a risorse statiche dovrà fornire un modo per accedere a contenuti e dati memorizzati su una Base Dati, dati con i quali un utente internet potrà interagire da remoto.



Le informazioni inviate dal web server viaggiano in rete trasportate dal protocollo HTTP, dunque diventa importante dare uno sguardo al protocollo request/response HTTP, di cui Struts fa molto uso.

1.3 Request/Response

HTTP è l'acronimo di HyperText Transfer Protocol (protocollo di trasferimento di un ipertesto). Usato come principale sistema per la trasmissione di informazioni sul web. Le specifiche del protocollo sono attualmente in carica al W3C (World Wide Web Consortium).

L'HTTP funziona su un meccanismo richiesta/risposta: il client apre una connessione verso un server ed esegue una richiesta, il server esegue la richiesta del client e restituisce la risposta chidendo la connessione. Nell'uso comune il client corrisponde al browser ed il server al sito web. Vi sono quindi due tipi di messaggi HTTP: messaggi richiesta e messaggi risposta.

Il messaggio richiesta è composto di tre parti: linea di richiesta (request line), sezione header (informazioni aggiuntive), Body (corpo del messaggio).

La linea di richiesta è composta dal metodo, URI e versione del protocollo.
l'URI sta per Uniform Resource Identifier ed indica l'oggetto della richiesta (ad esempio la pagina web che si intende ottenere)
I metodi HTTP più comuni sono GET e POST. Utilizzando il metodo GET, i dati vengono appesi alla request-URI nella forma di coppie *chiave=valore* separati tra di loro dal carattere "&". La stringa seguente è un esempio di campo request-URI per una richiesta di tipo GET.

http://www.java-net.tv/servelt/Hello?nome=Massimo&cognome=Rossi

"http://www.java-net.tv" rappresenta l'indirizzo del web-server a cui inviare la richiesta. I campi *"/servlet/Hello"* rappresenta la locazione della applicazione web da eseguire. Il carattere "?" separa l'indirizzo dai dati e il carattere "&" separa ogni coppia chiave=valore.
Questo metodo utilizzato per default dai browser a meno di specifiche differenti, viene utilizzato nei casi in cui si richieda al server il recupero di informazioni mediante query su un database.
Il metodo POST pur producendo gli stessi effetti del precedente, utilizza una modalità di trasmissione dei dati differente impacchettando i contenuti dei campi di un form all'interno di un message-header. Rispetto al precedente, il metodo POST consente di inviare una quantità maggiore di dati ed è quindi utilizzato quando è necessario inviare al server nuovi dati affinché possano essere memorizzati all'interno della Base Dati.
Il messaggio di risposta è composto dalle seguenti tre parti: linea di stato (status-line), sezione header, body (contenuto della risposta).
La linea di stato riporta un codice a tre cifre catalogato nel seguento modo:

- 1xx : Informational
- 2xx: Success
- 3xx: Redirection
- 4xx: Client error
- 5xx: Server error

Nel caso più comune il server risponde con un codice 200 (OK) e fornisce contentuto nella sezione body. Altri casi sicuramente incontrati da tutti sono:

302 Found. La risorsa è raggiungibile con un altro URI indicato nel header Location. Di norma i browser eseguono la richiesta all'URI indicato in modo automatico senza interazione dell'utente.

404 Not Found. La risorsa richiesta non è stata trovata e non se ne conosce l'ubicazione. Di solito avviene quando l'URI indicato in modo incorretto od è stato rimosso il contenuto dal server.

500 Internal Server Error. Il server non è in grado di rispondere alla richiesta per un suo problema interno.

Gli header della risposta più comuni sono:

Server. Indica il tipo la marca , la versione del server.

Content-Type. Indica il tipo di contenuto restituito. La codifica di tali tipi (detti Media type) sono registrati presso IANA (Internet Assigned Number Authority) e sono detti tipi MIME (Multimedia Internet Message Extensions), la loro codifica è descritta nel documento RFC1521. I principali sono:

- text/html. Documento HTML)
- text/plaintext. Documento di testo non formattato)
- image/jpeg.Immagine di fromato Jpeg

Dal momento che tutto il traffico HTTP è anonimo e *in chiaro*, sono state sviluppate diverse alternative per garantire differenti livelli di sicurezza. Ad oggi quello più usato è il protocollo HTTPS. Il meccanismo HTTPS, inventato da Netscape usa il sottostante canale cifrato a livello di trasporto mediante SSL o TLS per impedire l'intercettazione di qualsiasi parte della transazione. In pratica è un normale HTTP inglobato in un Secure Sockets Layer (SSL), un sistema di comunicazione che garantisce la privacy quando si comunica con altre applicazioni abilitate all'SSL. Una connessione SSL può essere stabilita tra un client e un server solo quando entrambi i sistemi sono in esecuzione in modalità SSL e hanno la capacità di autenticarsi reciprocamente.

1.4 Le basi di Struts

A partire dai primi siti internet dai contenuti statici, siamo oggi in grado di poter effettuare qualsiasi operazione tramite Web da remoto, grazie alla programmazione lato server. Questa enorme crescita, come una vera rivoluzione, ha voluto le sua vittime. Nel corso degli anni sono nate e poi completamente scomparse un gran numero di tecnologie a supporto di un sistema in continua evoluzione.

Quelli che in origine erano statici documenti HTML, sono divenuti, grazie al server-side programming, pagine dinamiche costruite on-the-fly per processare moduli e visualizzare i risultati di interrogazioni di database. Il primo linguaggio usato per rendere interattive le pagine Web è stato il Perl, sfruttando la rudimentale, limitata e inefficiente specifica nota come Common Gateway Interface. La CGI è rimasta per anni il principale strumento per la server-side web programming. Anche se oggi gli script CGI sono abbastanza diffusi, sono da tempo comparse soluzioni tecnicamente più solide e pratiche per lo sviluppo di applicazioni Web di qualità professionale e respiro industriale, anche se già arrivate al loro tramonto. Citiamo ad esempio il Server JavaScript di Netscape, le Active Server Pages di Microsoft o l'IntraBuilder di Borland. Java deve gran parte del suo iniziale successo alle applet, che insieme ai plugin/ActiveX e al JavaScript/VBScript rappresentano la parte client-side del web programming. Tuttavia, sviluppare in ambiente Internet significa nella maggior parte dei casi dover gestire entrambe le parti, tanto il server quanto il client. Dunque è naturale che molti sviluppatori desiderino utilizzare un solo linguaggio per lo sviluppo di applicazioni sia lato client che lato server e quindi JavaSoft ha dato una eccellente risposota proponendo nuove tecnologie che si sono imposte tra la comunità di sviluppatori.

Servlet e JavaServer Pages, su cui si fonda Struts, rappresentano oggi lo stato dell’arte delle tecnologie web. Raccogliendo la pesante eredità lasciata dai suoi predecessori, la soluzione proposta da SUN rappresenta quanto di più potente e flessibile possa essere utilizzato per sviluppare applicazioni web.

1.4.1 Java Servlet API

Java Servlet sono oggetti Java con proprietà particolari che vengono caricati ed eseguiti dal web server che le utilizzerà come proprie estensioni. Qualcuno le ha definete delle applet lato server. Il web server di fatto mette a disposizione delle Servlet il container che si occuperà della gestione del loro ciclo di vita, della gestione dell'ambiente all'interno delle quali le servlet girano, dei servizi di sicurezza. Il container ha anche la funzione di passare i dati dal client verso le servlet e viceversa ritornare al client i dati prodotti dalla loro esecuzione.

Dal momento che una servlet è un oggetto server-side, può accedere a tutte le risorse messe a disposizione dal server per generare pagine dai contenuti dinamici come prodotto della esecuzione delle logiche di business. E' ovvio che sarà cura del programmatore implementare tali oggetti affinché gestiscano le risorse del server in modo appropriato evitando di causare danni al sistema che le ospita.

Le servlet Java sono diventate l'elemento principale per estendere e migliorare le web application usando la piattaforma Java. Forniscono un metodo component-based e indipendente dalla piattaforma, per costruire web application. Le servlet hanno soppiantato lo standard CGI per la loro efficienza e caratteristica di scalabilità. Queste sono più efficienti del modello di thread dello standard CGI, poichè creano un solo processo e consentono a ciascuna richiesta utente di utilizzare un thread molto più leggero, che viene gestito dalla Java Virtual Machine. Una servlet viene mappata a uno o più URL (Uniform Resource Locator) e, quando il server riceve una richiesta ad uno degli url della servlet, viene invocato il metodo di servizio nella servlet che risponde. Dal momeno che ciascuna richiesta è associata con un thread separato, thread o utenti multipli posso invocare il metodo service() contemporaneamente. Questa natura multithread delle servlet è una delle ragioni principali per cui sono più scalabili rispetto alle applicazioni standard. Inoltre, dal momento che sono scritte in Java, esse non risultano limitate a una sola piattaforma o ad un unico sistema operativo.

Un altro vantaggio significativo della loro natura Java è che le servlet sono in grado sfruttare l'intera serie delle API Java (application programming interfaces) tra cui JDBC (Java DataBase Connectivity) e EJB.

Le servlet non vengono eseguite direttamente da un web server, ma hanno necessità di un loro contenitore, detto servlet-container.

Per le proprie servlet, gli sviluppatori sono liberi di scegliere tra i molti container disponibili. Le servlet sono portabili e possono migrare attraverso diversi container, senza il bisogno di ricompilare il codice sorgente.

Il package di base delle Servlet API è javax.servlet e contiene le classi per definire Servlet standard indipendenti dal protocollo. Tecnicamente una Servlet generica è una classe definita a partire dall'interfaccia Servlet contenuta all'interno del package *javax.servlet*. Questa interfaccia contiene i prototipi di tutti i metodi necessari alla esecuzione delle logiche di business, nonché alla gestione del ciclo di vita dell'oggetto dal momento del suo istanziamento, sino al momento della sua terminazione.

I metodi definiti in questa interfaccia devono essere supportati da tutte le servlet o possono essere ereditati attraverso la classe astratta GenericServlet che rappresenta una implementazione base di una servlet generica. Il package include inoltre una serie di interfacce che definiscono i prototipi di oggetti utilizzati per tipizzare le classi che saranno necessarie alla specializzazione della servlet generica in servlet dipendenti da un particolare protocollo.

Http servlet rappresentano una specializzazione di servlet generiche e sono specializzate per comunicare mediante protocollo http. Il package javax.servlet.http mette a disposizione una serie di definizioni di classe che rappresentano strumenti utili alla comunicazione tra client e server con questo protocollo, nonché forniscono uno strumento flessibile per accedere alle strutture definite nel protocollo, al tipo di richieste inviate ai dati trasportati.

Le interfacce ServletRequest e ServletResponse rappresentano rispettivamente richieste e risposte http.

Per uno studio attento della tecnologia Java Servlet si rimanda al sito ufficiale della SUN http://java.sun.com/products/servlet

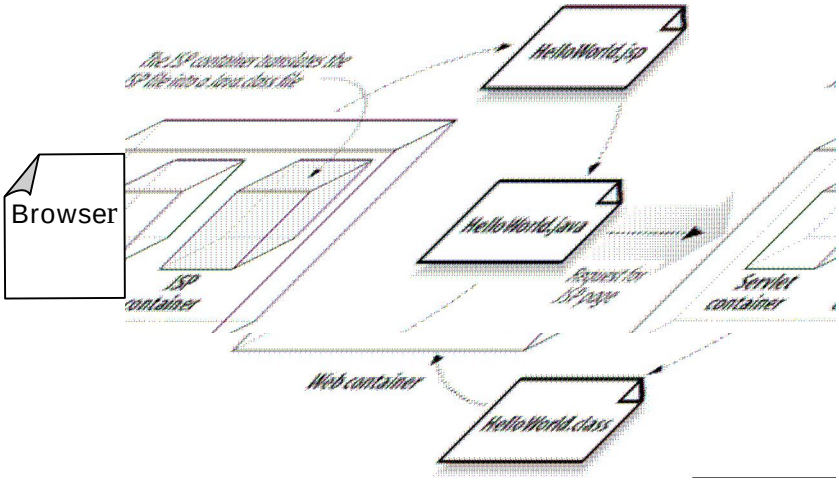
Sebbene le servlet sono risultate essere ottime nello svolgere il loro compito fin dalla loro introduzione, si è subito capito che incorporare i risultati delle richieste all'interno di semplici pagine statiche HTML, rappresentava una forte limitazione. Il

passo successivo nella progressione di sviluppo delle tecnologie web basate sulla piattaforma Java sono state le JavaServer Pages.

1.4.2 JSP - Java Server Pages

Il primo aspetto importante da sottolineare è che le JavaServer Pages sono la naturale estensione delle Java Servlet, ovvero in un certo senso, sono anche loro delle servlet.
Le pagine jsp sono documenti di testo con estensione *.jsp* contenenti una combinazione di HTML statico e tag tipo XML e di scriptlet. La possibilità di fondere codice html con codice Java senza che nessuno interferisca con l'altro consente di isolare la rappresentazione dei contenuti dinamici dalle logiche di presentazione. Il disegnatore potrà concentrarsi solo sulla impaginazione dei contenuti che saranno inseriti dal programmatore che non dovrà preoccuparsi dell'aspetto puramente grafico. I tag e gli scriptlet incapsulano la logica che genera il contenuto delle pagine. I file .jsp vengono elaborati e trasformati in file .java. A questo punto un compilatore java compila il sorgente e crea un file .class che può andare in esecuzione in un serlet container.
Dunque è il traduttore che ha il compito di crare il file .java di creare anche la servlet a aprtire dalla pagina jsp.
Jsp presenta diversi vantaggi rispetto le tecnologie concorrenti:

- JSP è una specifica non un prodotto, dunque gli sviluppatori posso scegliere la loro implementazione ideale.
- Le JSP sono compilate e non iterpretate, con un miglioramento delle prestazioni.
- Le pagine JSP supportano sia script che un accesso completo al linguaggio Java e possono essere estese tramite custom tag.
- Sono basate sul modello "write once run anywhere" della tecnologia Java.



1.5 Cos'è un framework?

Un framework è una architettura generica che costituisce l'infrastruttura per lo sviluppo di applicazioni in una determinata area tecnologica, nel caso di Struts, si tratta del mondo delle web-application. In parole povere, un framework è un insieme di classi ed interfacce di base, a disposizione del programmatore, che costituiscono l'infrastruttura di una applicazione. Non bisogna però pensare, che usare un framework, equivalga quindi, ad usare una libreria di classi, quali ad esempio le classi di base del linguaggio Java. Usare una libreria, vuol dire avere a disposizione dei metodi e delle funzionalità, ma resta a nostro carico, il compito di controllare il flusso applicativo. Adottare un framework, invece, vuol dire attenersi ad una specifica architettura, in cui saranno i componenti del framewrok ad occuparsi del flusso applicativo ed il programmatore avrà solo il compito di estendere le classi del framewrok e/o implementarne delle interfacce.
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 elaborativo 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.

1.5.1 Usare i frameworks

In genere i vantaggi dell'utilizzo di un framework vanno ben oltre gli svantaggi, anzi, quanto più il progetto è di grossi dimensioni, tanto più l'utilizzo di un framework è inevitabile. Anche se come detto, scegliere un framework, vuol dire adottare implicitamente una specifica architettura, questo non deve essere visto come vincolante, anzi, nel caso di framework validi, rappresenta 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 inventare tutto ex-novo come spesso purtroppo accade.

E' 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.

E' 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.

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.

1.5.2 Scegliere un framefork

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 sicuramente devono essere prese in considerazione.

- Maturità del progetto

  E' 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 costiuiscono 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 alle proprie esigenze.

Come vedremo di seguito 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

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.

1.6 I Patterns

La piattaforma J2EE ha costituito, sin dal suo primo apparire, una vera rivoluzione nell'ambito dello sviluppo di software a livello enterprise. Ciò è testimoniato sia dal gran numero di applicazioni che con essa sono state sviluppate, sia dalla continua e costante evoluzione della piattaforma medesima, dato che nel corso del tempo sempre più numerose sono le tecnologie che sono andate a confluirvi. D'altra parte, è anche vero che approcciare J2EE non è banale, e chi affronta la progettazione di una nuova applicazione si trova a confrontarsi con una serie di problematiche che derivano dalla complessità intrinseca di J2EE stessa.

A questo va aggiunto che apprendere le tecnologie J2EE viene spesso confuso con imparare a progettare con le tecnologie J2EE, e non si tratta di una differenza da poco. Esiste molta documentazione di qualità che permette di conoscere a fondo la piattaforma J2EE, ma esiste d'altra parte non molta chiarezza su come lavorarci in pratica.

Il problema di fondo è quindi essenzialmente di tipo pratico: come passare da una fase di design ad una implementazione affidabile, robusta e performante.

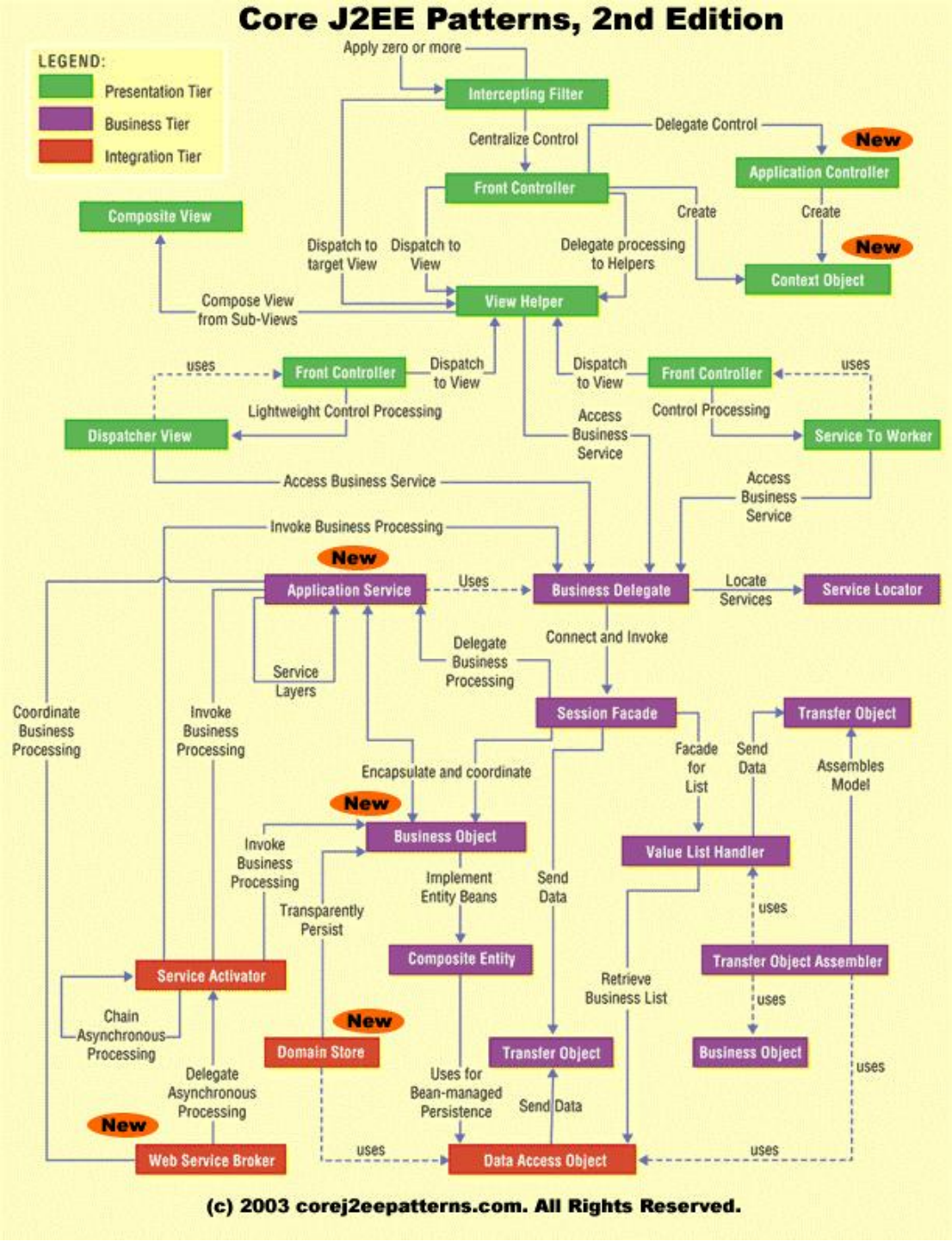
Quello che occorre è un insieme di regole che permettano di risolvere in modo sicuro i principali problemi che possono presentarsi, e che siano altresì riapplicabili in contesti analoghi.

Questa è, più o meno, la definizione di Pattern.

Nella letteratura specialistica si trovano diverse definizioni, alcune a volte quasi pittoresche. Un esempio fra tutte: "Un pattern è un'idea che si è rivelata utile in un dato contesto e probabilmente lo sarà ancora in altri". Se si legge con attenzione questa definizione, una delle prime cose che viene da pensare è che un Pattern non è affatto qualcosa di inventato ex-novo, è piuttosto qualcosa che discende dalla pratica e dall'esperienza. E' infatti proprio nella pratica e nella esperienza che è possibile affinare la ricerca e la messa in opera delle soluzioni migliori e, tra di esse, di quelle riutilizzabili. Dunque un Pattern è una soluzione comune ad un problema comune in un contesto ricorrente.

Viene quindi in mente di impiegare i Pattern in un ambito complesso come quello J2EE, proprio per avere a disposizione un ricco insieme di "best practices" e poter affrontare in modo affidabile le fasi di implementazione a partire da un progetto. Ecco dunque, perché una lunga serie di Pattern J2EE, pattern architetturali specifici per applicazioni complesse multi-tier.

Il risultato è il cosiddetto J2EE Pattern Catalog, che raggruppa i diversi Pattern sostanzialmente in tre categorie, Presentation Tier, Business Tier, Integration Tier. (http://corej2eepatterns.com/Patterns2ndEd/)

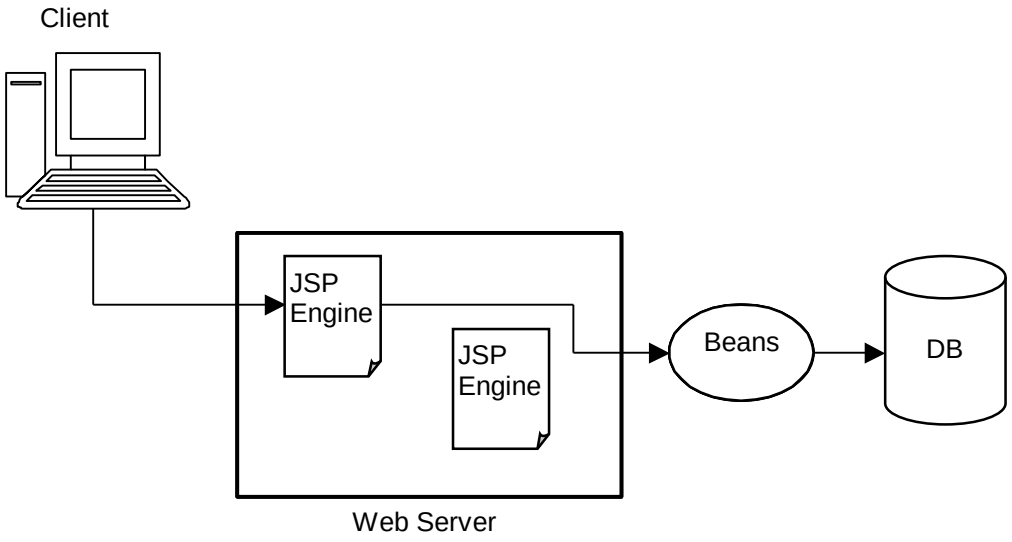


1.6.1 Il pattern MVC (Model-View-Controller)

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 di quei modelli applicativi che nella letteratura sono indicati spesso come "JSP Model 1" e “JSP Model 2”, introdotti con le prime specifiche delle JavaServer Pages.

Model 1

Da sole, JavaServer Pages consentono di realizzare applicazioni web dinamiche accedendo a componenti Java contenenti logiche di business o alla base dati del sistema. In questo modello il browser accede direttamente ad una pagina JSP che riceve i dati di input, li processa utilizzando eventualmente oggetti Java, si connette alla base dati effettuando le operazioni necessarie e ritorna al client la pagina html prodotta come risultato della processazione dei dati.

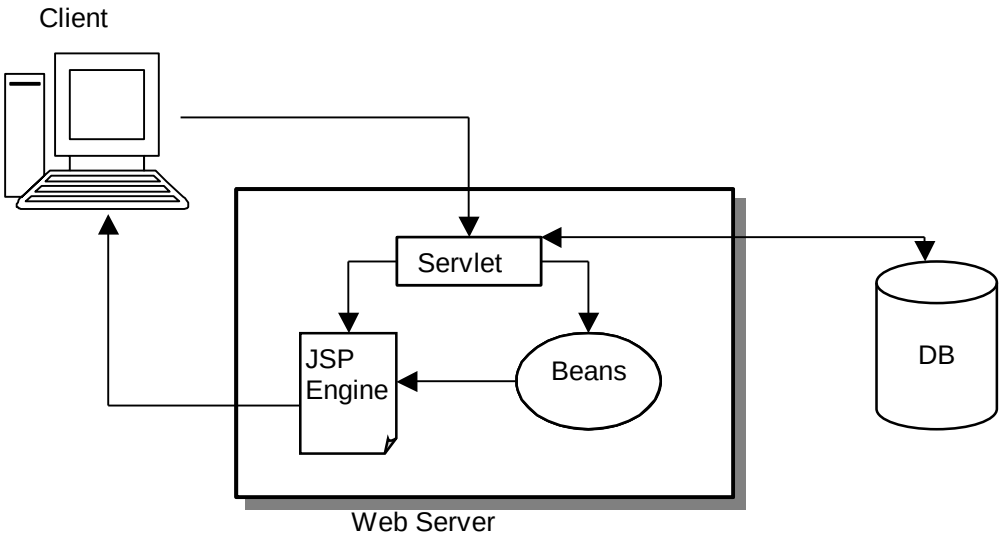


In base a questo modello l'applicazione è costruita secondo una logica "JSP centric" in base alla quale presentation , control e business logic dell'applicazione sono tutti a carico delle pagine JSP. Il web browser accede direttamente alle pagine JSP

dell'applicazione che al loro interno contengono logica applicativa e logica di controllo del flusso. Quando l'applicazione cresce in complessità non è pensabile svilupparla seguendo un simile approccio.

Model 2

Il secondo modello, utilizza JavaServer Pages come strumento per sviluppare template demandando completamente ad una particolare servlet la processazione dei dati di input.



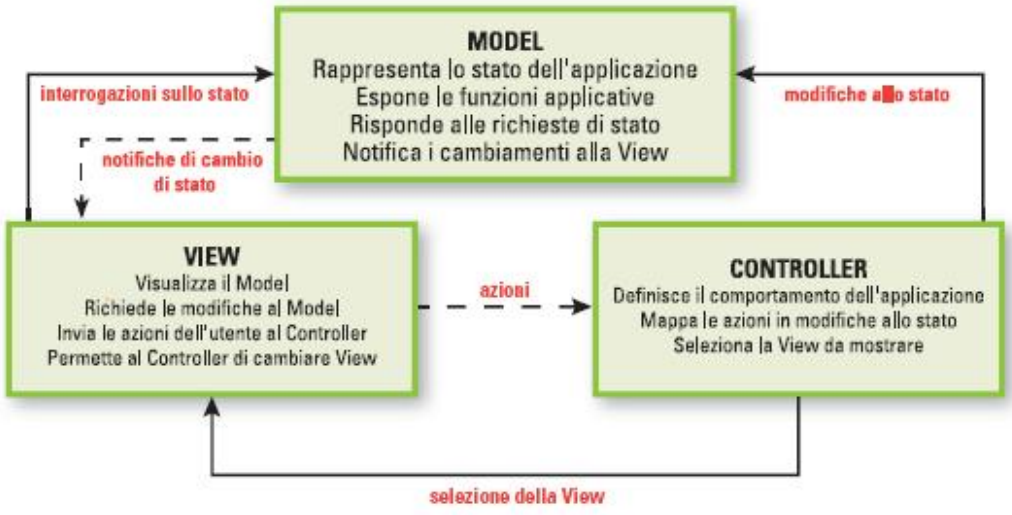
Nell'architetture Model 2 la richiesta del client viene intercettata prima da una servlet, detta controller servlet. Questa gestisce l'elaborazione iniziale della richiesta e determina quale pagina JSP deve essere visualizzata.

Dunque in questa architettura un client non invia mai direttamente una richiesta a una pagina jsp. Ciò permette alla servlet di effettuare una elaborazione di front-end comprendente autorizzazione ed autenticazione, logging centralizzato e internazionalizzazione. Una volta terminata l'elaborazione della richiesta, la servlet dirige la richiesta alla giusta pagina JSP. Come si vede la differenza principale tra le due architetture è la presenza di questo punto di accesso ai servizi singolo, rappresentato dalla servlet, il che incoraggia il riutilizzo e la separazione tra i vari strati dell'applicazione.

Proprio per questo il pattern Model 2 è più conosciuto come pattern MVC (Model-View-Controller) Secondo questo pattern, i componenti di un'applicazione vengono logicamente classificati a seconda che siano gestori di logiche di business e di accesso ai dati (model), gestori di visualizzazione dati e dell'input dell'utente (viewer o view) e gestori del flusso di controllo complessivo dell'applicazione (controller). L'accoppiamento tra i vari livelli è minimo, con evidenti vantaggi in termini di riusabilità, manutenibilità, estensibilità e modularità.
In pratica, si hanno innegabili vantaggi come:

- indipendenza tra i business data (model) la logica di presentazione (view) e quella di controllo (controller)
- separazione dei ruoli e delle relative interfacce
- viste diverse per il medesimo model
- semplice supporto per nuove tipologie di client: bisogna scrivere la vista ed il controller appropriati riutilizzando il model esistente

Di seguito è presentato un diagramma di interazione che evidenza le responsabilità dei tre componenti:



Precisiamo che tale pattern non è direttamente legato alle web application, anzi, venne introdotto nel mondo del software per la costruzione di interfacce grafiche con Smalltalk-80.

1.6.2 MODEL

Analizzando la figura precedente, si evince che il core dell'applicazione viene implementato dal Model, che incapsulando lo stato dell'applicazione definisce i dati e le operazioni che possono essere eseguite su questi. Quindi definisce le regole di business per l'interazione con i dati, esponendo alla View ed al Controller rispettivamente le funzionalità per l'accesso e l'aggiornamento. Per lo sviluppo del Model quindi è vivamente consigliato utilizzare le tipiche tecniche di progettazione object oriented al fine di ottenere un componente software che astragga al meglio concetti importati dal mondo reale. Il Model può inoltre avere la responsabilità di notificare ai componenti della View eventuali aggiornamenti verificatisi in seguito a richieste del Controller, al fine di permettere alle View di presentare agli occhi degli utenti dati sempre aggiornati.

1.6.3 VIEW

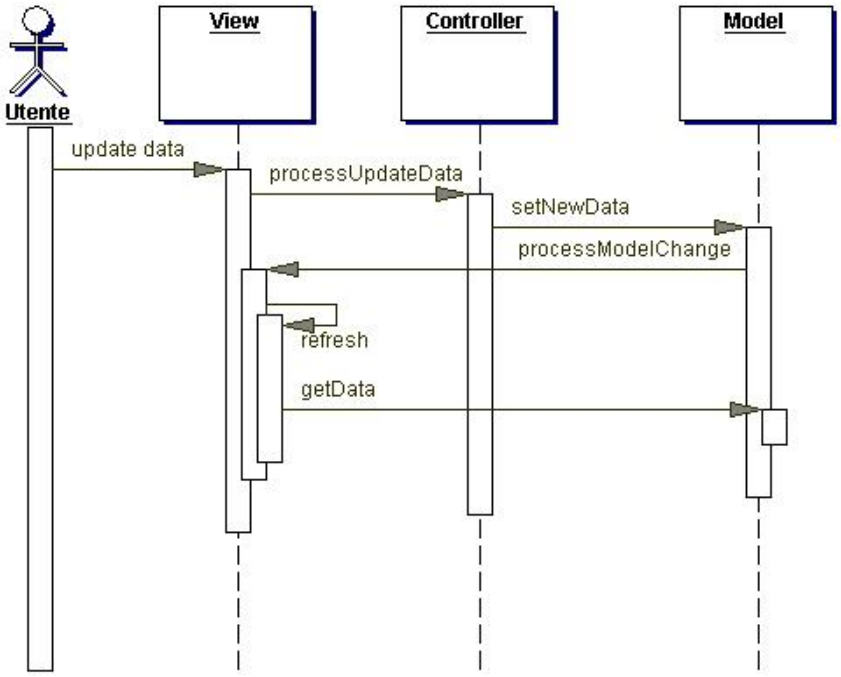
La logica di presentazione dei dati viene gestita solo e solamente dalla View. Ciò implica che questa deve fondamentalmente gestire la costruzione dell' interfaccia grafica (GUI) che rappresenta il mezzo mediante il quale gli utenti interagiranno con il sistema. Ogni GUI può essere costituita da schermate diverse che presentano più modi di interagire con i dati dell'applicazione. Per far sì che i dati presentati siano sempre aggiornati è possibile adottare due strategie note come "push model" e "pull model". Il push model adotta il pattern Observer, registrando le View come osservatori del Model. Le View possono quindi richiedere gli aggiornamenti al Model in tempo reale grazie alla notifica di quest'ultimo. Benché questa rappresenti la strategia ideale, non è sempre applicabile. Per esempio nell'architettura J2EE se le View che vengono implementate con JSP, restituiscono GUI costituite solo da contenuti statici (HTML) e quindi non in grado di eseguire operazioni sul Model. In tali casi è possibile utilizzare il "pull Model" dove la View richiede gli aggiornamenti quando "lo ritiene opportuno". Inoltre la View delega al Controller l'esecuzione dei

processi richiesti dall'utente dopo averne catturato gli input e la scelta delle eventuali schermate da presentare.

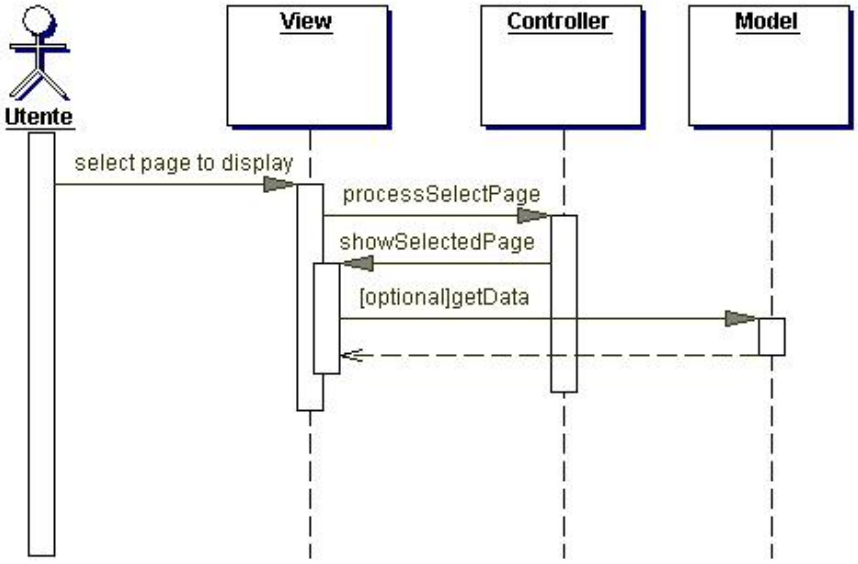
1.6.4 CONTROLLER

Questo componente ha la responsabilità di trasformare le interazioni dell'utente della View in azioni eseguite dal Model. Ma il Controller non rappresenta un semplice "ponte" tra View e Model. Realizzando la mappatura tra input dell'utente e processi eseguiti dal Model e selezionando la schermate della View richieste, il Controller implementa la logica di controllo dell'applicazione. 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.
Il controller è il “punto di accesso unico” di cui si parlava prima. Per capire la sua importanza consideriamo quest’ esempio.
Immaginiamo di voler integrare la sicurezza nel proprio sito web. La prima pagina è di solito una schermata di login, questa pagina dirige poi l’utente verso un’altra schermata che consente di continuare in maniera sicura. Però, se non c’è nulla che impedisca all’utente di raggiungere direttamente una pagina, ciascuna pagina (in una architettura non MVC) deve avere le sue funzioni di sicurezza, ovvero un controllo in ogni pagina, anche in quelle intermedie in cui non sarebbe necessario. Certamente è una tecnica poco pratica. Con il pattern MVC la sicurezza risiede all’interno dell’oggetto controller; dal momento che il punto di accesso ai servi è unico, diventa unico anche il punto in cui concentrare i controlli di sicurezza.

Evidenziamo ora solo due dei possibili scenari che potrebbero presentarsi utilizzando un applicazione MVC-based tramite diagrammi di sequenze: la richiesta dell'utente di aggiornare dati



e la richiesta dell'utente di selezionare una schermata.



Una possibile implementazione del paradigma Model-View-Control in ottica J2EE può essere la segunte:

- Model: componenti EJB che incapsulano la logica applicativa e implementano l'accesso agli Enterprise Integration System (DBMS, Host,…).
- Controller: entità modellabile come Servlet (o JSP) e classi dette RequestHandler per gestire le richieste dell'utente
- View: è costituita dalle pagine JSP, questa volta prive di logica applicativa.

I COMPONENTI DI STRUTS

Di seguito, passiamo in rassegna i vari elementi di Struts prima menzionati, dando un taglio per lo più concettuale, e rimandando più avanti per un approccio pratico alla programmazione con Struts, in quanto dallo studio fatto, ritengo che sia sicuramente utile separare lo studio di un framework, dallo studio della sua programmazione, in quanto otteniamo le basi per una buona progettazione della nostra web-application, sfruttando a pieno le risorse che ci sono messe a disposizione, il che rappresenta sicuramente una carta vincente.

2.1 Come Struts implementa il pattern MVC

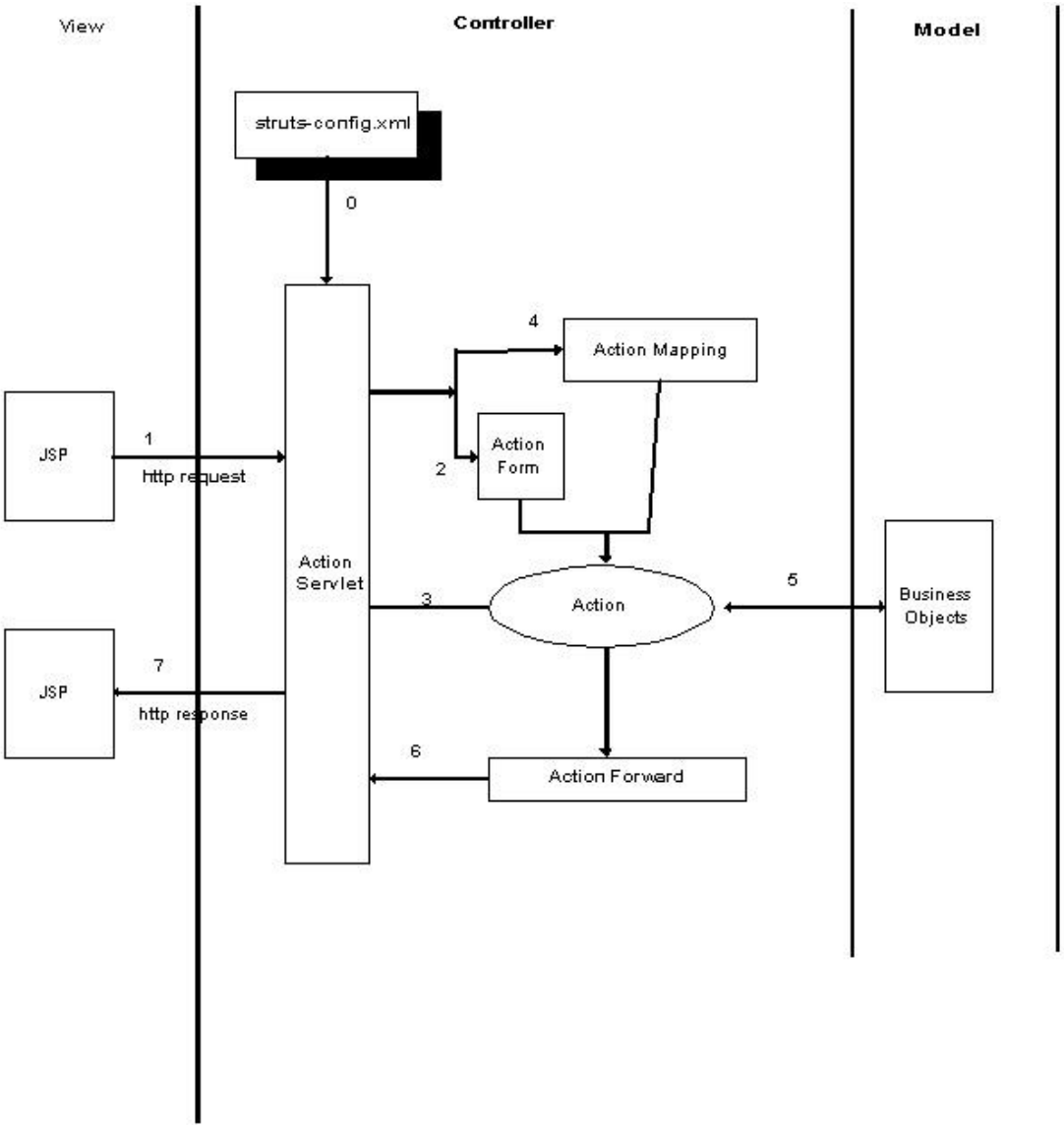
Come già detto Struts è un framework per la realizzazione di web application che aderisce a pieno al pattern MVC. Rappresenta uno strato software di controllo flessibile, che utilizza tecnologie standard Java (Servlet, Beans, ResourceBundles) e Xml e mette a disposizione del programmatore un controller sofisticato per l'implementazione pratica del pattern Model2. Il notevole vantaggio di questa astrazione è che Struts è essenzialmente agnostico rispetto alle scelte di chi faccia poi il model ed il view. In pratica vengono utilizzati componenti Ejb oppure Jdbc o plain Jdo per costruire il model, mentre per il view si lavora ancora con le Jsp, a questo punto viste come componenti di un framework più "robusto" e non come costituenti l'intero framework applicativo a disposizione del programmatore, come avveniva per il model1

I componenti fondamentali di Struts sono:

- *struts- config.xml* E' il file XML di configurazione di tutta l'applicazione. In questo file vengono definiti gli elementi dell'applicazione e le loro associazioni.
- *ActionServlet*: e' la servlet di controllo centralizzata che gestisce tutte le richieste dell'applicazione.
- *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.
- *File properties*: contiene i messaggi di errore, label, etc. e necessario per implementare il processo di internazionalizzazione.

A questi, come in tutte le web application che fanno uso delle servlet (non solo con Struts), si aggiunge il file web.xml, in cui viene specificato al server qual è la servlet controller della nostra applicazione web.

Coerentemente con quanto detto finora, si tratta di un insieme di classi ed interfacce che costituisco l'infrastruttura di un a web application J2EE, più un deployment descriptor (descrittore dell'impiego dell'applicazione: fisicamente si tratta di un documento Xml) per inizializzare l'interazione tra le risorse dell'applicazione.
C'è da osservare che tutti i componenti di Struts elencati 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.
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.
Nella figura seguente è rappresentato schematicamente il flusso elaborativo nella 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 e verrà dettagliato in seguito. ma fornisce una indicazione di base su come viene gestito il flusso elaborativo in una applicazione sviluppata con 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 che hanno il compito di elaborare le singole richieste.
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.
Coerentemente con quanto illustrato fin ora, risulta che Struts rispecchia a pieno le caratteristiche di un framework per il pattern MVC. Infatti, come atteso, troviamo che:

- 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.

2.2 Il file web.xml

È il descrittore di deploy della web-application, trasmette le informazioni di configurazione tra chi sviluppa l'applicazione, chi ne effettua il deploy e chi l'assembla. Questo file è descritto in dettaglio nelle specifiche delle Servlet Java, essendo un file di configurazione necessario per tutte le web application, non solo per quelle realizzate con il framework Struts. Esistono però in tale file delle informazioni specifiche per Struts che devono essere configurate quando si costruiscono applicazioni con tale framework. I web container utilizzano il descrittore per caricare e configurare la web-application quando questo viene avviato.Il file viene letto all'avvio del Web container e specifica dove si trova la nostra applicazione e come deve essere eseguita, in altre parole il file specifica ogni richiesta dove deve andare.

Sia per il descrittore di deploy dela web application, che per il file di configurazione di Struts, il formato si basa su un *Document Type Definition (DTD)*, che definisce i blocchi di costruzione che posso lecitamente essere utilizzati nei file XML. I DTD aiutano a specificare documenti XML *well-formed* (ben formati). Il DTD per il descrittore di deploy di applicazioni versione 2.3 può essere scaricato dall'indirizzo http://java.sun.com/dtd/index.html

Le seguenti dichiarazioni del DTD mostrano gli elementi top-level che costituiscono il descrittore di deploy per una web applicazion:

```
<!ELEMENT web-app (icon?, display-name?, description?,
   distributable?, context-param*, filter*, filter-mapping*,
   listener*, servlet*, servlet-mapping*, session-
   config?,mimemapping*,welcome-file-list?, error-page*,
   taglib*,resourceenv-ref*, resource-ref*, security-
   constraint*, loginconfig?,security-role*, env-entry*, ejb-
   ref*, ejb-local-ref*)
>
```

L'elemento web-app è la root del descrittore, gli altri elementi all'interno delle parentesi sono elementi figli, che devono essere collocati all'interno dell'elemento root nel file XML. Una illustrazione di tutti gli elementi figli, sarebbe in questo luogo eccessiva, dunque passiamo subito a vedere e commentare un esempio completo di file web.xml.

```
1  <?xml version="1.0" encoding="UTF-8"?>

2  <!DOCTYPE web-app
3     PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
4     "http://java.sun.com/dtd/web-app_2_3.dtd">

5  <web-app>
6    <servlet>
7      <servlet-name>action</servlet-name>
8      <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
9      <init-param>
10       <param-name>config</param-name>
11       <param-value>/WEB-INF/struts-config.xml</param-value>
12     </init-param>
13     <init-param>
14       <param-name>debug</param-name>
15       <param-value>3</param-value>
16     </init-param>
17     <init-param>
18       <param-name>validating</param-name>
19       <param-value>false</param-value>
20     </init-param>
21     <load-on-startup>1</load-on-startup>
22   </servlet>
23   <servlet-mapping>
24     <servlet-name>action</servlet-name>
25     <url-pattern>*.do</url-pattern>
26   </servlet-mapping>
27   <welcome-file-list>
```

```
28          <welcome-file>login.jsp</welcome-file>
29      </welcome-file-list>

30   <taglib>
31  <taglib-uri>/WEB-INF/tld/struts-bean.tld</taglib-uri>
32  <taglib-location>/WEB-INF/tld/struts-bean.tld</taglib-location>
33  </taglib>

34  <taglib>
35  <taglib-uri>/WEB-INF/tld/struts-html.tld</taglib-uri>
36  <taglib-location>/WEB-INF/tld/struts-html.tld</taglib-location>
37  </taglib>

38  <taglib>
39  <taglib-uri>/WEB-INF/tld/struts-logic.tld</taglib-uri>
40  <taglib-location>/WEB-INF/tld/struts-logic.tld</taglib-location>
41  </taglib>

42  <taglib>
43  <taglib-uri>/WEB-INF/tld/struts-template.tld</taglib-uri>
44  <taglib-location>/WEB-INF/tld/struts-template.tld</taglib-location>
45  </taglib>

46  </web-app>
```

All'interno del tag <servlet> (6), viene specificata la servlet di controllo della nostra applicazione web, che nel caso di Struts, sarà la ActionServlet, ed una serie di parametri che vengono inviati a questa dal web container per l'inizializzazione.

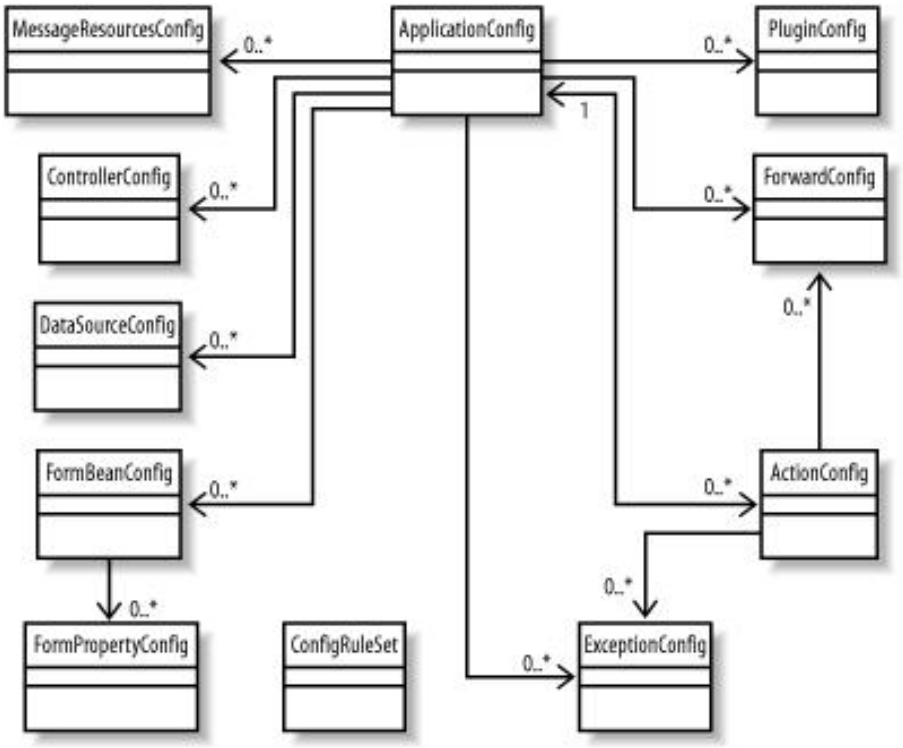
La ActionServlet, è referenziata attraverso il nome 'action' (7) e all'interno del tag <servlet-class> specifichiamo il nome della classe. Nelle righe 10-11, passiamo alla servlet un parametro di nome 'config' con il valore '/WEB-INF/struts-config.xml' ovvero stiamo dicendo alla servlet qual è e dove si trova il file di configurazione che deve leggere, per il mappaggio delle azioni.

Il parametro di debug (14), serve per stabilire il livello di debug che si vuole per l'applicazione. Sono possibili 4 livelli, il valore di default, nel caso non venga specificato è 0. Il parametro validating (18-19) è un booleano che specifica se si vuole che al file di configurazione di Struts, venga fatto il parsing per vedere se si

tratta di un file valido. Il valore di default è true. In (21) è settato ad 1 il parametro <load-on-startup>, in base al quale il container istanzia la ActionServlet allo start-up della web application. Nelle righe 23-26, viene specificato al container quando inviare il controllo alla servlet 'action', in pratica ogni volta che il container, nel contesto di questa applicazione, incontra un URL che finisce in .do invia la richiesta alla servlet action. È importante notare che in questo modo è possibile integrare il framework con una vecchia applicazione, specificando al container di utilizzare la servlet di Struts solo quando si trova di fronte ad un url con questa estensione. Le righe 27-29 definiscono la pagina di default della web-application che viene mostrata se l'utente non ne specifica una. Infine 30-45, viene definito l'uso e la locazione di alcune librerie disponibili con Struts e di cui si parlerà meglio in seguito.

2.3 Il file di configurazione di Struts

Se è vero che la classe ActionServlet è la spina dorsale del framework , è pur vero che nulla può funzionare senza il file struts-comfig-xml. Mentre il file web.xml specifica dove una richiesta deve andare, quest'ultimo, invece, determina esattamente cosa accadrà ad essa. Quindi il controller utilizza questo file per determinare quali oggetti deve chiamare. Il file può essere letto una sola volta in fase di start-up e le relazioni sono immagazzinate nella memoria per ottenere migliori performance. Queste informazioni, a partire dalla versione 1.1 vengono tenute all'interno di JavaBeans, contenuti nel package org.apache.struts.config. dove in pratica, abbiamo una classe per ogni settore del file.



La figura mostra le principali classi del package. All'interno, troviamo anche due classi particolarmente importanti:

ConfigRuleSet: è una classe con tutte le regole di parsing del file di configurazione; si occupa anche di istanziare le classi che contengono i dati del file di configurazione quando l'applicazione viene fatta partire.

ApplicationConfig: è la classe centrale del package e contiene tutte le informazioni che descrivono l'applicazione.

In pratica, all'atto della lettura del file. Viene creata una istanza appropriata per ogni settore alla quale sarà possibile accedere in seguito.
Come per il file di deploy della web application, anche il file di configurazione di struts, possiede un *Document Type Definition (DTD)* necessario per validarlo, scaricabile dal sito: http://jakarta.apache.org/struts/dtd/

La seguente dichiarazione del DTD di Struts indica che l'elemento root per il file XML è *<struts-config>* e che possiede otto elementi figli:

```
<!ELEMENT struts-config (data-sources?, form-beans?,
       globalexceptions?,globalforwards?,action-mappings?,
       controller?, messageresources*,plug-in*)
>
```

notiamo che i nomi di questi elementi, sono per lo più al plurale, questo per sottolineare che all'interno di ognuno di questi tag, potremo trovare uno o più tag (con lo stesso nome, ma al singolare), che rappresentano i singoli contenitori.

Andiamo ora ad analizzare i vari settori del file.

Datasources

Nel tag datasources é possibile impostare una rudimentale datasource che può essere utilizzata dall'interno del framework. Molte implementazioni di datasource usano un meccanismo di pool delle connessioni per migliorare prestazioni e scalabilità. Tutti i produttori dei più diffusi database forniscono delle implementazioni di datasource.
L'elemento *<datasources>* può contenere zero o più elementi *<datasource>*, il quale permette di specificare molteplici elementi *<set-property>*,che consentono di configurare proprietà specifiche per la propria implementazione.
Gli attributi per l'elemento datasource sono:

className: la classe di implementazione del bean di configurazone che manterrà le informazioni della datasource;

key: l'attributo del contesto della servlet sotto il quale sarà salvato la datasource;

type: il nome della classe Java di implementazione della datasource.

I principali elementi property che possiam torvare nel tag <datasource> sono:

description - la descrizione del datasource;

autoCommit - un booleano che specifica se impostare l'autocommit dopo ogni operazione sul database ;

driverClass - il driver da utilizzare per la connesione;

password e username;

url per specificare il percorso al database.

Un esempio di settaggio, potrebbe essere il seguente:

```
<!-- tag per il settaggio dei datasources -->
<datasources>
      <datasource>
        <set-property property="description" value="Postgred
        Data Source"/>
        <set-property property=”autoCommit” value=”false” />
        <set-property property=”url”
        value=”jdbc:postgresql://localhost/mydatabase” />
        <set-property property=”driverClass”
        value=”org.postgresql.Driver” />
        <set-property property="user" value="admin"/>
        <set-property property="password" value="admin"/>
        <set-property property="maxActive" value="150" />
        <set-property property="maxIdle" value="30" />
        <set-property property="maxWait" value="5000" />
      </datasource>
</datasources>
```

Si posso specificare datasource multiple, assegnare ciascuna di esse a una chiave unica e accedere ad una particolare datasource nel framework tramite la sua chiave.

Form-beans

I form-beans sono dei beans che contengono i dati di una form, inviata, ad esempio, da un utente. All'interno del tag <form-beans> abbiamo la possibilità di inserire più elementi <form-bean>, ovvero elementi che rappresentano un singolo form-bean. Ognuno di questi, può avere i seguenti attributi:

type: il nome della classe che estende la ActionForm (con tutto il package);

name: il nome che rappresenta la chiave che identifica il bean all'interno del file struts-config e necessari per refenziarlo in tutto il framework;

dynamic: un attributo booleano che indica se il form bean ha delle proprietà di tipo dinamico;

className: quando non si vuole utilizzare il bean di configurazione standard org.apache.struts.config.FormBeanConfig, è possibile specificare qui la propria classe che deve estendere quella appena menzionata.

```
<form-beans>
      <form-bean name="esempioForm"
      type="package.della.classe.ActionFormEsempio" />
</form-beans>
```

Global-exceptions

Questo tag è stato introdotto a partire dalla versione 1.1 e permette di configurare in maniera dichiarativa la gestione delle eccezioni rilanciate da più actions. Il tag può contenere nessun o più tag <exception>. In realtà , questo elemento può essere configurato anche nell'elemento action, che si vedrà più avanti. L'elemento exception descrive un mapping tra una eccezone java che può verificarsi durane l'elaborazione di una request e una istanza di org.apach.struts.action.ExceptionHandler responsabile di trattare l'eccezione lanciata. L'elemento exception, possiede altri elementi figli e soprattutto importanti attributi:

handler: il nome della classe che si occupa di processare le eccezioni (la classe di default è org.apache.struts.action.ExceptionHandler);

key: la chiave che ci permette di determinare il messaggio di errore associato con questa eccezione presente ne resource bundle;

path: il percorso alla classe o pagina JSP a cui indirizzarsi nel caso avvenga questa eccezione;

scope: specifica se la classe ActionError contenente gli errori generati venga messa nella request o nella session;

type: la classe che rappresenta l'eccezione catturata;

bundle: l'attributo ServletContext che identifica un resource bundle dal quale l'attributo key dovrebbe provenire;

className: la classe di implementazione del bean di configurazione che manterrà le informazioni sull'eccezione. Se specificata, deve essere una classe derivante dalla org.apache.struts.config.ExceptionConfig, che è la classe predefinita quando nessun valore è specificato.
Riportiamo un esempio di un lemento global.exceptions:

```
<global-exceptions>
      <exception
        key="chiave.messaggio.errore"
        path="/paginaDiErrore.jsp"
        type="java.io.IOException"/>
</global-exceptions>
```

Global-forwards

Ogni action che viene eseguita, termina sempre con un forwarding o un redirecting verso un view, che è solitamente una pagina jsp (potrebbe essere una qualsiasi altra risorsa). Anziché fare diretto riferimento alla vista, il framework Struts usa il concetto di forward per associare un nome logico con la risorsa.

Il tag per il settaggio di forwards globali visibili a tutte le actions viene utilizzato nel caso si abbiano delle forward comuni tra le varie actions. All'interno troveremo l'elemento figlio forward, che specifica la singola associazione. L'elemento forward effettuai l mapping di un nome logico a un URI relativo ad una applicazione., ciò garantisce un maggior disaccoppiamento tra la logica del controller e del model dalla view. Questo elemento può essere definito anche nell'elemento action e presenta i seguenti attributi:

name: il nome logico con cui cercare l'istanza all'interno dell'ActionMapping e referenziare questo forward;

path: il percorso relativo al contesto a cui il controllo deve essere reindirizzato;

redirect: è un booleano valorizzato a true se si vuole utilizzare il metodo sendRedirect per arrivare alla risorsa, valorizzato a false se si intende utilizzare il metodo forward della classe RequestDispatcher;

className: la classe di implementazione del bean di configurazione che manterrà le informazioni sul forward. Se specificata, deve essere una classe derivante dalla org.apache.struts.action.ForwardConfig, che è la classe predefinita quando nessun valore è specificato.
Ecco un esempio di un elemento global-forward:

```
</global-forwards>
      <forward
        name="success"
        path="/paginaJSP.jsp" />
</global-forwards>
```

Action-mappings

Tag per il mapping delle Action. Al suo interno contiene nessun o più tag <action>.
L'elemento action descrive un mapping tra il path di una specifica request e la corrispondente classe Action che dovrà gestirla ed in che modo, specificando se è

associato uno dei JavaBeans (form) descritti prima per il trattamento dei dati. Il controller seleziona un mapping particolare facendo corrispondere il path dell'URI della request con l'attributo path in uno degli elementi action.
L'elemento action, contiene vari elementi figli, tra cui i più importati, sodo già visti in precedenza: exception e forward.
I numerosi attributi dell'elemento sono:

attribute: il nome dell'attributo con scope di request o di session sotto le quali è possibile l'accesso al form bean di questa action. È permesso un valore solamente quando vi sia un form bean specificato nell'attributo name;

className: la classe di implementazione del bean di configurazione che manterrà le informazioni dell'action. La classe predefinita è org.apache.struts.action.ActionMapping;

forward: il path relativo all'applicazione di nua servlet o di una risorsa JSP verso cui verrà effettuato il forward, invece di istanziare e invocare la classe Action;

include: il path relativo all'applicazione di nua servlet o di una risorsa JSP che saranno incluse nella response, invece di istanziare e invocare la classe Action;

type: la classe Action utilizzata; in alternativa a questo attributo si possono utilizzare o forward (che contiene il path della Servlet o della JSP che processerà la request) o include;

input: il percorso al form che e' stato invocato; viene utilizzato nel caso si presenti un errore di validazione dei dati, così che l'applicazione sappia da chi è stata fatta la richiesta e possa tornare alla pagina di partenza elencando gli errori trovati che l'utente può così correggere;

name : nome del form bean definito nel tag form-beans che questa action utilizzerà;

path: l'URI associato con quella determinata action (può essere settato a true se quella determinata action sarà quella di default per tutta l'applicazione, in modo che gestisca tutte le richieste non gestite dalle altre action. Solo una action può essere settata come quella di default);

validate : é settato a true se si vogliono validare i parametri passati nel form bean;

scope: usato per identificare lo scope nel quale viene posto il form bean (se è request o sesson), può essere specificato solo se è presente l'attributo name;

type: un nome di classe Java che estende la org.apache.struts.action.Action, usato per elaborare la request se mancano i parametri forward o include;

parameter: un attributo utilizzato per passare eventualmente un parametro; nel caso si stia invocando una action che estende la classe DispatchAction questo attributo si rileverà molto utile per specificare il nome del parametro inserito nella request il cui valore non sarà nient'altro che il nome del metodo da invocare.

```
<action
      path="/signin"
      type="com.oreilly.struts.storefront.security.LoginAction
      "
      scope="request"
      name="loginForm"
      validate="true"
      input="/security/signin.jsp">
      <forward name="Success"
              path="/index.jsp"
              redirect="true"/>
      <forward name="Failure"
              path="/security/signin.jsp"
              redirect="true"/>
</action>
```

il tag <forward> alla fine dell'Action, reindirizza il tutto ad un'altra Action o ad una JSP.

Controller

Il tag controller, contiene per lo più informazioni che sono state già specificate nel file web.xml. A partire dalla versione di Struts 1.1, da quando alla classe principale ActionServlet è stata affiancata la classe RequestProcessor (come si vedrà meglio avanti), è stato necessario replicarli anche nel file di configurazione di Struts.
Dal momento che il controller ed i suoi attributi sono definiti nel file struts-config.xml, si possono definire elementi controller separati per ciascuna sottoapplicazione. L'elemento controller ha un solo elemento figlio set-property.
Troviamo quindi i principale attributi dell'elemento controller:

cIassName: la classe di implementazione del bean di configurazione che manterrà le informazioni dell'action. La classe predefinita è org.apache.struts.config.ControllerConfig;

contentType: il tipo do contenuto e la codifica caratteri opzonale che viene impostata per ciascuna response. Il valore di default è text/html;

debug: il livello di debug così come viene impostato nel file web.xml;

locale: un booleano che indica se vogliamo che sia messo in sessione la classe Locale(che indica il paese di appartenenza) preferita dall'utente;

maxFileSize: nel caso di upload specifica le dimensioni massime per accettare un file;

multipartClass: il nome della classe da utilizzare nel caso di download/upload (la classe di default è org.apache.struts.upload.DiskMultipartRequestHandler);

nocache: un booleano che se impostato a true inserisce in ogni response l'header che specifica che la pagina non deve essere presa dalla cache;

processorClass: la classe che processa le request (il valore di default è la classe org.apache.struts.action.RequestProcessor)

Ecco come configurare l'elemento controller

```
<controller
      contentType="text/html;charset=UTF-8"
      debug="3"
      locale="true"
      nocache="true"
      processorClass="com.struts.framework.CustomRequestProc
      essor"/>
</controller
```

Messages-resources

L'elemento *message-resources* specifica caratteristiche dei resource bundle che contengono i messaggi localizzati per una data applicazione. Il resource bundle, come si vedrà in seguito, è uno dei componenti fondamentali per il successo di Struts, in quanto fornisce uno strumento validissimo per gestire l'internazionalizzazione e per costruire delle views generiche e facilmente gestibili nel tempo. Ciascun file di configurazione di Struts, può definire uno o più resource bundel. Questo elemento contiene solamente un elemento set-property ed al solito, una serie di attributi:

className: la classe di implementazione del bean di configurazione che manterrà le informazioni del message-resources. Se specificata, deve essere una classe derivante dalla org.apache.struts.config.MessageResourcesConfig, che è la classe predefinita quando nessun valore è specificato;

key: l'attributo coni l quale sarà immagazzinato questo resource bundle di messaggio;

parameter: il nome base del resource bundle (ovvero senza estensione .properties);

null: specifica come la sottoclasse MessageResources deve comportarsi quando viene specificata una chiave non presente.

```
<message-resources
      null=”false”
      parameter=”ApplicationResources”/>
</message-resources
```

Plug-in

Nel tag plug-in vengono specificati i nomi di quelle classi, con i loro eventuali parametri di configurazione, che vengono create o distrutte quando viene fatta partire l’applicazione o viene fermata. Per creare un plug.in, si crea una classe Java che implementa l’interfaccia org.apache.struts.action.Plugin e si aggiunge un elemento plug-in al file di configurazione.

```
<plug-in
      className=”nomeDellaClasse”>
      <set-property property=”debug” value=”true” />
</plug-in>
```

2.3.1 Suddividere in più parti lo struts-config.xml

Dalla versione 1.1 viene fornita la possibilità di creare più files di configurazione, suddividendo l’applicazione in tante sotto-applicazioni; ciò si è reso necessario perché nei grandi progetti con numerosi programmatori si creava non poca difficoltà nel condividere questo file vitale per l’applicazione e che deve essere modificato continuamente.

Per configurare questi files all'interno dell'applicazione basta aggiungerli al file web.xml come parametri iniziali:

```
<init-param>
      <param-name>config/sottoapplicazione1</param-name>
      <param-value>/WEB-INF/struts-sottoapplicazione1-
      config.xml</param-value>
</init-param>
```

in questo modo specifichiamo al controller di caricare la sotto-applicazione utilizzando il file di configurazione specifico per questa.

2.4 La classe ActonServlet

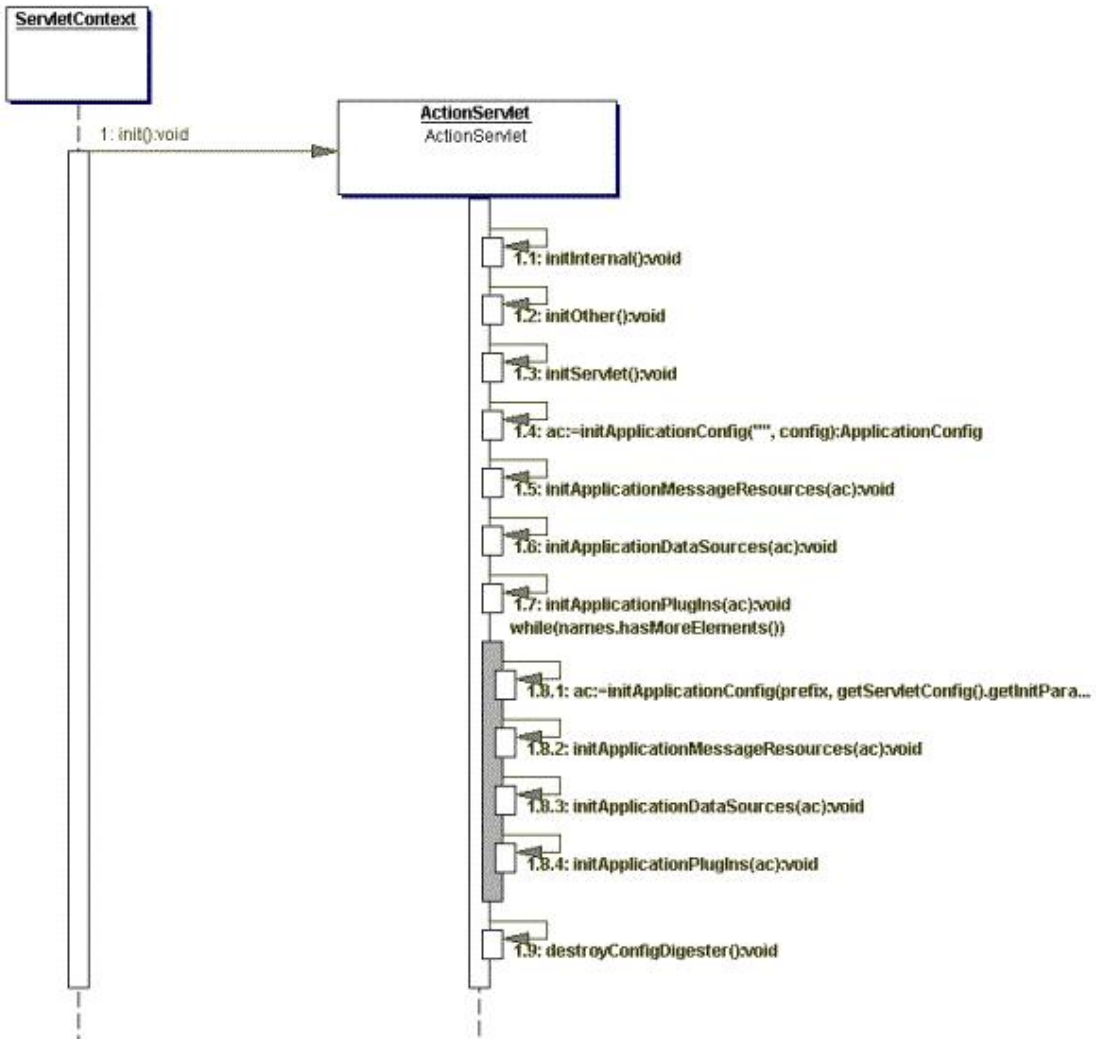
La org.apache.struts.action.ActionServlet è la servlet di controllo di Struts. Come già accennato, è la servlet che gestisce tutte le richieste http che provengono dai client e indirizza il flusso applicativo in base alla configurazione presente nel file XML struts-config.xml.

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.

In realtà, a partire dalla versione 1.1, la servlet principale e' stata affiancata da una nuova classe di controllo org.apache.struts.action.RequestProcessor, in cui è stato spostato gran parte del codice. Questa si occupa di processare le richieste, dando maggior libertà al programmatore. In questo modo, separando le due azioni di ricevimento e di processo, l'utente non e' più vincolato al processore interno della ActionServlet, ma può utilizzarne uno scritto da lui, estendendo la classe RequestProcessor. In particolare, è di uso comune fare l'override del metodo processPreprocess() che viene eseguito dal RequestProcessor prima dell'elaborazione

di ogni richiesta. Questo metodo che solitamente non contiene codice, è il punto ottimale per inserire controlli di validità della sessione, dell'utente o simili.

Dunque ricapitolando, la ActionServlet ha il compito di inizializzare la web-application ed intercettare tutte le richieste http, le quali verranno poi passate alla RequestProcessor per essere processate.
Come tutte le servlet, per poter utilizzare la ActionServlet, questa deve essere configurata nel file web.xml, come abbiamo già visto.
Vediamo cosa succede quando la ActionServlet viene inizializzata, cosa che avviene o all'avvio del container, nel caso in cui abbiamo inserito il tag <load-on-startup> nel file web.xml, oppure quando questa viene invocata per la prima volta.



Al momento dell'inizializzazione della servlet il container invocherà il metodo init() (questo vale per tutte le servlets) che si occuperà di effettuare i seguenti passi:

1. Invoca il metodo initInternal() che inizializza la classe MessageResource, utilizzata per far ritornare messaggi specifici a seconda della lingua impostata nell'applicazione.
2. Imposta alcuni parametri utilizzati all'interno dell'applicazione (lo struts-config.xml, il livello di debug, la validazione dei files XML) invocando il metodo initOther().
3. Invoca il metodo initServlet() che preleva il valore impostato nel file web.xml con cui la servlet verrà invocata (questo valore verrà utilizzato nel tag html:form per impostare l'url esatto a cui inviare i dati inseriti all'interno del form) ed inoltre registra tutti i files DTDs che ci possono servire per la validazione.
4. Chiama il metodo initApplicationConfig() che inizializza alcuni dati di configurazione del framework per l'applicazione di default, effettuando il parsing del file struts-config.xml impostato nel web.xml con il parametro config. Crea quindi un oggetto di tipo ApplicationConfig, con all'interno tutte le informazioni che ci interessano(actions, formbeans, forwards, exceptions etc.), che verrà immagazzinato nel ServletContext.
5. Viene invocato il metodo initApplicationMessageResources() che carica ed inizializza ogni MessageResource impostato nello struts-config.xml per l'applicazione di default. Una volta caricato viene immagazzinato anche questo nel ServletContext con la chiave indicata all'interno del tag <message-resource>.
6. Inizializza i datasources impostati all'interno dello struts-config.xml per l'applicazione di default (initApplicationDataSources()). Questo metodo sostituisce il metodo initDataSources() che e' stato lasciato solo per compatibilità con le vecchie versioni ed in futuro verrà eliminato.
7. Inizializza i plug-in specificati nello struts-config.xml per l'applicazione di default.(initApplicationPlugIns()).

8. (ciclo while)Dopo aver inizializzato l’applicazione di default la servlet vedrà se esistono delle sotto-applicazioni e provvederà ad inizializzarle una ad una ripetendo i procedimenti visti prima.

Terminato il processo di inizializzazione, la ActionServlet è pronta per ricevere tutte le richieste provenienti dal livello view. Come dicevamo, la ActionServlet è una estensione di javax.servlet.http.HttpServlet e dunque sarà pronta per ricevere dal client, richieste di tipo GET o POST. Quello che accade è questo:

1. I metodi doGet() e doPost() invocano allo stesso modo 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.

2.5 La classe RequestProcessor

Come abbiamo già detto prima questa classe è stata aggiunta nella nuova versione di Struts soprattutto per permettere agli sviluppatori di poter gestire il comportamento di un’applicazione secondo i propri bisogni quando questa riceve delle richieste. Creare dei processori personalizzati è molto semplice, basta aggiungere all’interno dello struts-config.xml il tag <controller> specificando il percorso della classe da

invocare più altri parametri che hanno altre funzionalità come abbiamo già visto in precedenza.

Ma adesso vediamo cosa avviene invece nel processore di default quando dalla ActionServlet viene invocato il suo metodo process() :

1. Viene invocato il metodo processMultipart() che controlla se il contentType della richiesta inizia con “multipart/form-data” ed in tal caso viene utilizzata una classe di Struts (org.apache.struts.upload.MultipartRequestWrapper), creata proprio nel caso in cui l’applicazione debba gestire degli uploads, invece dell’HttpServletRequest method
2. Viene chiamato il metodo processPath() che si occupa di determinare il percorso che verrà utilizzato per identificare la Action da utilizzare.
3. Il metodo processLocale() determina la localizzazione dell’utente e immagazzina l’oggetto Locale generato nella sessione.
4. Chiamando i metodi processContent() e processNoCache() determina il content type e se l’attributo noCache è impostato.
5. Esiste inoltre un metodo processPreprocess(), che non fa assolutamente nulla. Questo metodo viene invocato prima della chiamata alla Action ed è stato messo proprio per permettere allo sviluppatore per effettuare dei possibili controlli (di validazione, di sessione etc.) prima di entrare nella parte di business.
6. Invoca il metodo processMapping() che determina la classe ActionMapping da utilizzare.
7. Controlla se l’utente che ha fatto la richiesta è in possesso dei permessi (processRoles()).
8. Invoca i metodi che determinano la ActionForm, cioè il bean che si deve occupare del trasporto dei dati, lo popolano, invocano la action etc. etc.
9. Se nell'elemento <action> è presente l'attributo validate al valore true chiama il metodo validate() dell'ActionForm per il controllo dei dati forniti dalla request.

10. Se il controllo è ok a questo punto il RequestProcessor chiama il metodo execute() dell'Action configurata nell'elemento <action> delegandole l'elaborazione della richiesta.
11. Il metodo execute() dell'Action al termine dell'elaborazione restituisce un oggetto ActionForward che consente al RequestProcessor di inoltrare il flusso elaborativo.

2.6 La classe ActionMapping

La org.apache.struts.action.ActionMapping contiene le informazioni con cui determinare a quale Action è associato un particolare evento.
Queste informazioni vengono fornite in fase di startup, prelevandole dallo struts-config.xml, e trasformate in un set di ActionMapping inserite all'interno di un container (la classe ActionMappings : nel framework le classi che finiscono con una s sono dei contenitori) da cui verranno riprese quando bisognerà avere accesso ad ogni tipo di parametro di configurazione inserito nel tag <action>.
Riprendendo quindi una parte di codice dallo struts-config.xml possiamo vedere che all'elemento qui sotto corrisponde una classe ActionMapping che mappa il percorso "/Path" alla Action "percorso.della.classe.NomeAction".

```
<action path="/Path" type="percorso.della.classe.NomeAction">
     <forward name="success" path="/AltraAction.do" />
     <forward name="failure" path="/jspDiFailure.jsp" />
</action>
```

Quindi quando il controller riceverà una richiesta con il percorso contenente la stringa "/Path" invocherà il metodo perform() della Action "*percorso.della.classe.NomeAction*", inoltre a seconda dell'esito reinderizzerà il tutto

o ad un'altra Action associata al percorso "*/AltraAction*" o ad una pagina di errore "*/jspDiFailure.jsp*".
A partire dalla versione 1.1 questa classe estende *org.apache.struts.config.ActionConfig*, già vista in precedenza, che avrebbe dovuto prendere il suo posto rendendola deprecata, ma per problemi di compatibilità con le precedenti versioni di Struts questa classe viene ancora utilizzata. Difatti viene ancora passata come parametro nel metodo execute() della Action.

2.7 La classe Action

La classe Action può avere compiti diversi a seconda del tipo di applicazioni in cui viene utilizzata. Nelle applicazioni più semplici si occupa di gestire la parte di business logic associata con una richiesta, mentre in altri casi la classe Action invocherà un altro oggetto (EJBs, oggetti CORBA, classi DAO) che si occuperà lui della parte di business logic; compito della Action sarà allora quello di gestire gli errori e di interpretare i dati nelle *HttpServletRequest* per passarli come variabili Java alle classi che si occupano delle varie operazioni.

Quindi la Action non viene utilizzata propriamente per gestire la business logic, ma si occupa di altre funzioni, come il logging, il controllo della sessione, l'autenticazione ed autorizzazione di una richiesta, demandando gli altri servizi a delle classi esterne al framework. In questo modo possiamo riutilizzare con maggiore semplicità queste classi esterne anche in altre applicazioni ed inoltre, nel caso venga modificato qualcosa nel servizio, la classe Action non dovrà essere re-implementata. Un esempio sostanziale potrebbe essere quello delle chiamate al Database; se scriviamo il codice che tira su la connessione e chiama le query all'interno della classe Action, allora dovremo riscriverlo per ogni Action e non potremmo riutilizzarlo all'interno di un'altra applicazione. Invece se ad occuparsi delle

connessioni c’è un’altra classe invocata dalla Action allora potremmo riutilizzarla anche in altri parti ed in caso di modifiche (non so utilizzo di un DataSource al posto di una connessione singola etc.) la classe Action rimarrà tale e quale.

A differenza delle classi viste in precedenza, la classe action è quella che deve essere sicuramente modificata dal programmatore Struts, infatti per ogni funzione realizzata con Struts, bisogna creare una propria classe che estende la classe Action e ne implementa il metodo execute() che è fatto come segue (nelle versioni precedenti alla 1.1 al posto di execute c’era il metodo perform(), che attualmente viene richiamato da execute).

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
                             throws Exception{
//codice di esempio
//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() ha come parametri request e response http, un'istanza dell'oggetto ActionForm prima descritto, e un oggetto ActionMapping che contiene le infornazioni 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.
E' 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:

1. 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.
2. 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:

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

Le Action costiuiscono 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.

Ricordiamo che una volta scritta la nostra classe e la business logic, non ci resta che specificare nello struts-config.xml quando questa Action deve essere chiamata in causa e questo lo si fa all’interno del tag <action>.

2.8 La classe ActionForm

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 da 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'immissione 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 , istanziare 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 costiuiscono 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 costiuisce 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.

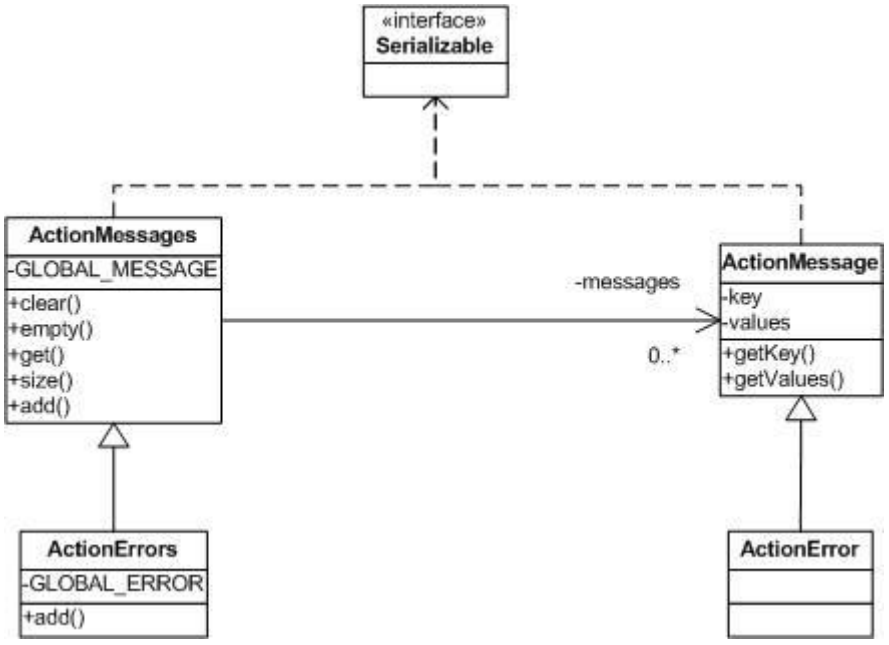
Nel metodo validate() va inserito il codice per la validazione formale dei dati del form. Ciò garantisce di avere un punto standard nel codice nel quale questa validazione viene effettuata e che i dati che arrivano alla Action siano già stati formalmente validati.
Questo metodo, non obbligatorio, solitamente ritorna un null, mentre se trova degli errori tra i dati ritorna un oggetto di tipo ActionErrors (di cui parleremo tra un pò).
Ricordiamo che scritta la nostra classe che estende la ActionForm, per usarla, dobbiamo configurarla nel file srtuts-config.xml, all'interno del tag <form-bean> e associarla poi ad una o più action nel tag <action>.
Logicamente pur utilizzando il framework Struts non è obbligatorio catturare i dati che ci vengono dal form HTML utilizzando la classe ActionForm; infatti lo sviluppatore può decidere liberamente di non associare nessun bean ad una Action e quindi di gestire manualmente tutte le operazioni che questa ci fornisce : trasferimento dei dati dalla request al bean, gestire il processo di validazione, gestire gli errori all'interno della Action (ma chi se la sentirebbe di rinunciare a questi vantaggi per implementarsi tutto da solo?).

2.9 La classe ActionForward

Abbiamo visto che i metodi execute() e perform() delle classi Action ritornano un oggetto di tipo ActionForward.
Si tratta nient'altro di una destinazione a cui deve puntare la ActionServlet una volta terminate le operazioni all'interno delle classi Actions. Se è andato tutto a buon fine, possiamo tranquillamente invocare una pagina di view o un'altra Action, oppure se ci sono state delle eccezioni, puntare ad una pagina di errore. La ActionForward contiene dunque, le informazioni che ci servono per proseguire il flusso dell'applicazione. Queste informazioni vengono impostate nel tag <forward> del file struts-config.xml, come visto nei paragrafi precedenti.

2.10 La classe ActionMessage

A partire dalle versione 1.1 di Struts è stata introdotta la classe *ActionMessage*.
Questa classe ha il solo compito di contenere dei messaggi che verranno in seguito visualizzati nelle pagine di view. Ogni istanza della classe ActionMessage con il messaggio da mostrare all'utente verrà inserita all'interno di un contenitore rappresentato dalla classe ActionMessages.
Questo contenitore verrà salvato all'interno della request (tramite il metodo *saveMessages()* della classe Action) e visualizzato con l'utilizzo del tag *<html:messages/>*.



2.11 La classe ActionError

Se i messaggi che vogliamo visualizzare nelle pagine di view sono dei messaggi particolari (messaggi di errore), allora non ci resta che utilizzare una classe che estenda la classe ActionMessage e che specifichi all'utente la gravità

dell'informazione ricevuta. Infatti sicuramente un messaggio di errore è bloccante rispetto ad un warning che serve solo per dare un'informazione all'utente su una possibile anomalia da correggere, ma che non vincola il regolare svolgimento dell'applicazione.

Ad esempio, 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.

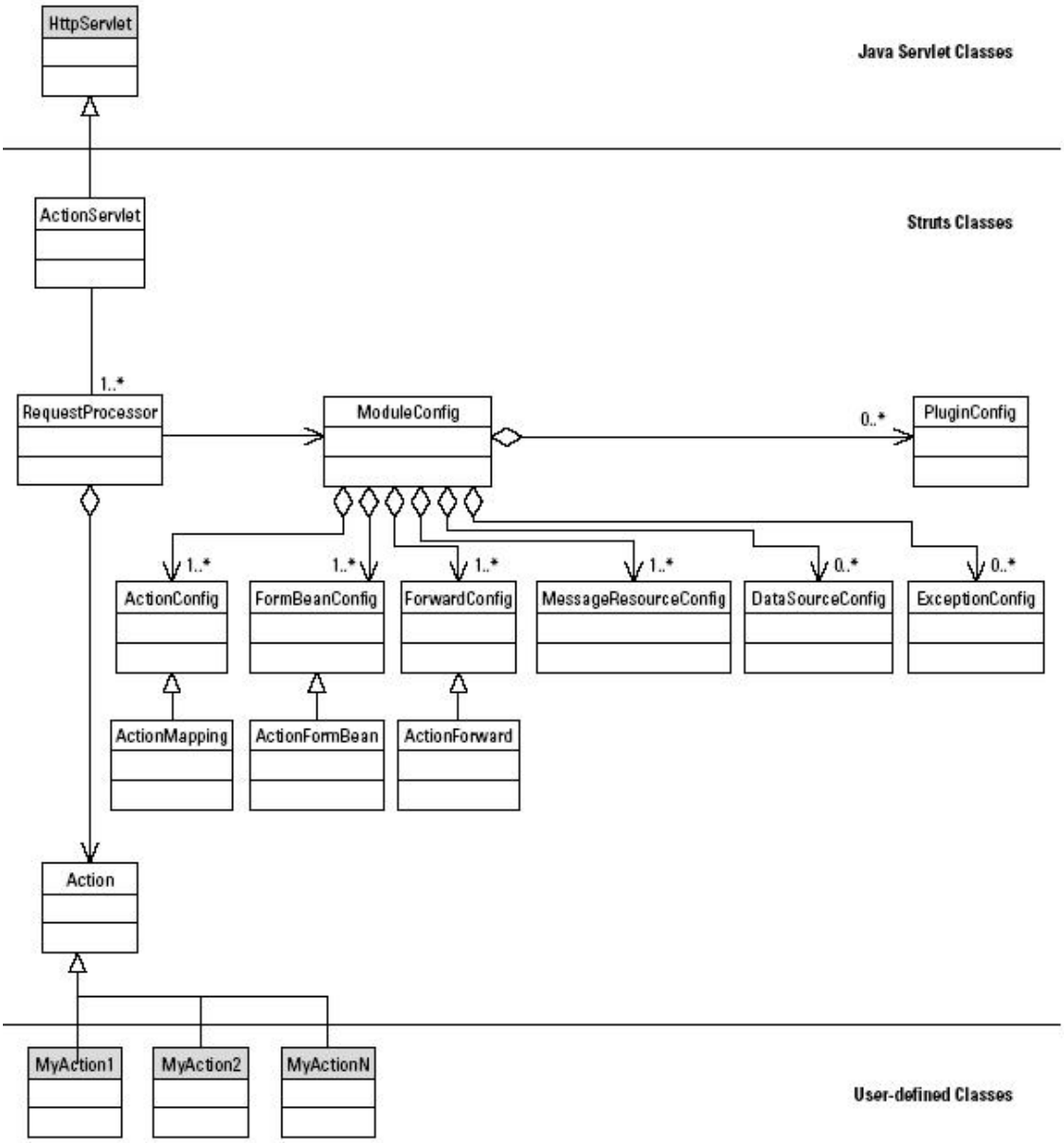
Il messaggio di errore viene reperito automaticamente dal framework dal resouce bundle dell'applicazione (il file properties); la chiave del messaggio è fornita nel costruttore dell'oggetto ActionError associato all'errore in questione. Di seguito è riportato l'esempio di un metodo che esegue la validazione 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.

In questo capitolo ho dunque descritto i componenti principali del controller Strtus, già sufficienti per realizzare una semplice applicazione. Una sintesi è fornita dal seguente diagramma delle classi del controller, tratto dal riferimento [3].

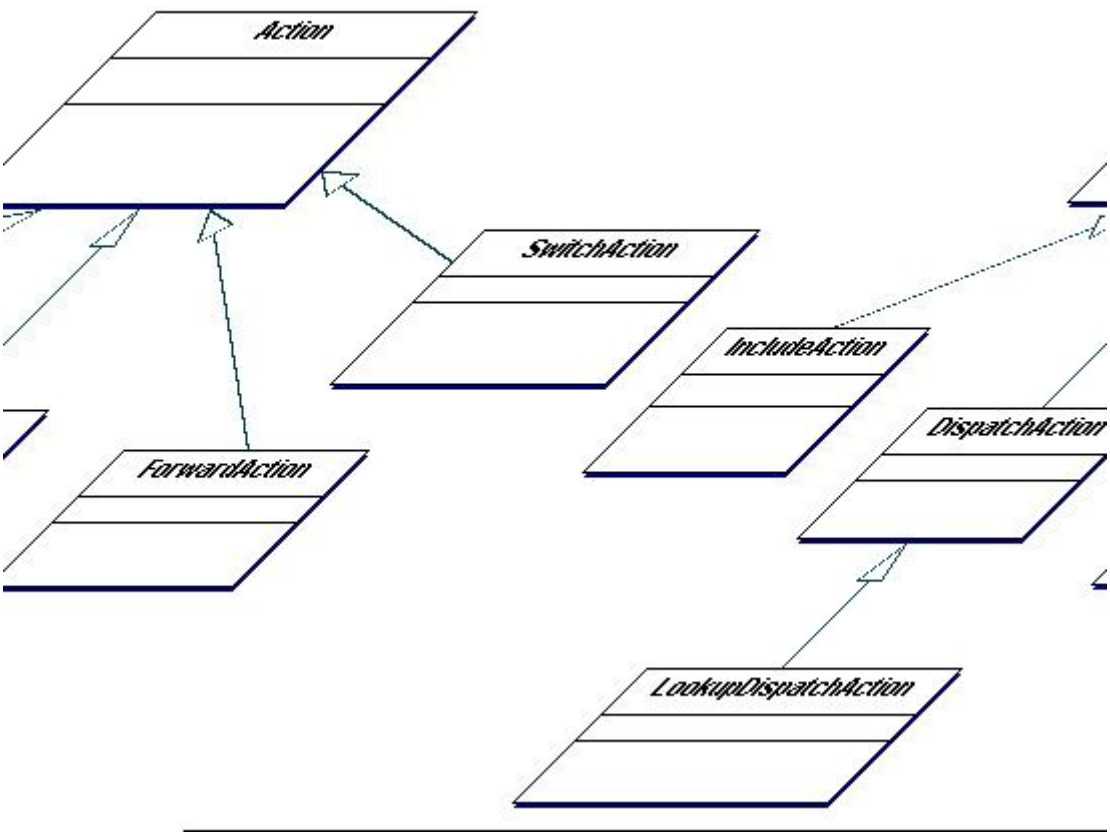


ALTRE FUNZIONALITA' E COMPONENTI AGGIUNTIVI DI STRUTS

Quelli visti nel capitolo precedente sono i componenti fondamentali del framework Struts e che già da soli forniscono tutti gli strumenti per realizzare una web-application completa. Ovviamente, nel corso del suo sviluppo, data la maturità del progetto, sono state aggiunte tante altre classi e sotto progetti apportando tante nuove funzionalità che semplificano ulteriormente il compito del programmatore e fanno accrescere la validità e la robustezza di questo framework, caratteristiche che come spiegato, sono alla base del suo successo.

3.1 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 visto che la classe Action si occupa di gestire una singola operazione all'interno del metodo perform(), quindi per gestire n operazioni dovremo allora creare n classi Actions. Questo tipo di approccio può risultare molte volte svantaggioso sotto alcuni punti di vista : eccessiva produzione di classi in applicazioni grandi, scrittura di codice che porta via molto tempo etc. Per venire incontro a questo tipo di problematica è stata creata la classe DispatchAction, il cui unico obiettivo è quello di poter comprendere all'interno di un'unica classe più operazioni che abbiano un collegamento logico. Tipico è l'esempio di operazioni di

inserimento, cancellazione , lettura e aggiornamento su una stessa tabella di un database. Sarebbe abbastanza 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 org.apache.struts.actions.DispatchAction.

La DispatchAction è assolutamente analoga ad una Acion base ma fornisce la possibilità di invocare diversi metodi della stessa purchè il client specifichi il metodo da chiamare. In pratica è come una Action che non ha un solo metodo execute() ma ne ha n 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;
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.esempio.GestioneTabellaAction"
name="gestioneTabellaForm"
scope="request"
input="/tabella.jsp"
parameter="metodo"/>
```

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

```
http://servername/context-root/gestioneTabella?metodo=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 la submit() del form.
L'importante è che è possibile 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. Nella pratica comune la DispatchAction è effettivamente molto utile e a mio parere salvo eccezioni è quella da utilizzare come classe base per le Action della propria applicazione invece della semplice Action standard.

3.2 La LookupDispatchAction

La org.apache.struts.actions.LookupDiapatchAction è utile quando la scelta del metodo da eseguire in una Action di tipo 'dispatch' è effettuata mediante i button di un form ma si ha la necessità di avere le label dei button localizzate e non ci si vuole affidare a codice JavaScript per la selezione del metodo da attivare.
La LookupDiapatchAction è quindi del tutto analoga alla DispatchAction per quel che riguarda la sua struttura, quello che cambia è il modo con il quale viene selezionato il metodo da mandare in esecuzione.
In questo caso le label dei button vengono asscociate alle loro key contenute nel resource bundle dell'applicazione, e queste key, che molto probabilmente non sono nomi di metodo validi, vengono mappate dallo sviluppatore ai metodi della Action mediante la definizione di un metodo così fatto:

```
protected Map getKeyMethodMap(ActionMapping mapping,
ActionForm form,
                              HttpServletRequest request) {
  Map map = new HashMap();
  map.put("bottone.leggi", "leggi");
  map.put("bottone.inserisci", "inserisci");
  map.put("bottone.cacnella", "cancella");
  map.put("bottone.modifica", "modifica");
  return map;
}
```

Nel metodo viene definita una HashMap nella quale ad ogni key è associato il nome di un metodo della Action. Questo metodo è usato dal framework per determinare la corrispondenza tra la label localizzata del bottone cliccato ed il metodo della LookupDispatchAction da eseguire.
In base a questo codice la definizione dei bottoni nella pagina JSP sarà del seguente tipo:

```
<html:form action="/gestioneTabella">
<html:submit property="method">
<bean:message key=" bottone.leggi ">
</html:submit>
```

```
<html:submit property="method">
<bean:message key=" bottone.inserisci ">
</html:submit>
<html:submit property="method">
<bean:message key=" bottone.cacnella ">
</html:submit>
<html:submit property="method">
<bean:message key=" bottone.modifica ">
</html:submit>
</html:form>
```

In questo modo è possibile avere una Action di tipo 'dispatch' i cui metodi sono attivabili da button di un form di una applicazione localizzata semplicemente scrivendo un metodo getKeyMethodMap() come descritto.

3.3 Action per il controllo del flusso

3.3.1 La ForwardAction

In molte occasioni quando ci spostiamo da una pagina di view all'altra non abbiamo bisogno di passare attraverso una Action per effettuare delle operazioni di business, quindi non facciamo nient'altro che collegare le due pagine con un link. Questa non è una soluzione molto corretta, perché in questo modo andiamo a violare un passaggio del pattern MVC, eliminando la parte di controller. Ciò potrebbe creare dei problemi nella visualizzazione dei messaggi nelle pagine di view, anche perché è proprio il controller che si occupa di gestire la richiesta e di inserire all'interno di questa le classi *ApplicationConfig* e *MessageResources*.

Per cercare di evitare questo tipo di problema, senza nello stesso tempo costringere lo sviluppatore a costruire una classe Action che abbia come unico scopo il forward

verso una pagina di view, si è creata la classe *ForwardAction* con l'unico scopo di collegare due pagine di view passando per il controller.
A differenza delle altre Actions il percorso a cui indirizzare l'applicazione dopo l'esecuzione del metodo perform() non viene indicato nel tag <*forward*> annidato nel tag <*action*>, ma nell'attributo "*parameter*"di quest'ultimo.

3.3.2 La IncludeAction

La IncludeAction è la corrispettiva della ForwardAction per l'operazione di include della risposta generata da un'altra risorsa. Consente di aggiungere alla propria elaborazione l'elaborazione effettuata da un'altra risorsa quale ad esempio una servlet. Non è molto utilizzata ma è comunque fornita da Struts qualora potesse servire.

3.4 La DynaActionForm

Utilizzare la classe ActionForm, come visto nel capitolo precedente, può causare una serie di problemi, soprattutto in progetti molto grandi. Infatti uno dei problemi maggiori è la creazione di un numero eccessivo di classi (molti infatti creano una classe per ogni form HTML), che rende difficoltosa la manutenzione e la gestione dell'applicazione. Una soluzione potrebbe essere quella di associare una classe per più di un form, che però si va a scontrare con un altro problema, la condivisione dei files. Infatti andando a creare un punto comune a molte parti dell'applicazione si rischia di creare anche una contesa tra gli sviluppatori sull'utilizzo di quella classe, con tutti i problemi che ciò comporta : troppe persone toccano quella classe rendendo

spesso il codice illeggibile, tempi di attesa oppure modifiche effettuate da altri che influiscono anche sulla nostra parte di codice etc..

Un altro problema (come se non bastassero già questi) è che qualunque modifica al bean comporta che la classe modificata sia ricompilata.

Per cercare di risolvere tutto ciò si è deciso di aggiungere nella versione 1.1 di Struts la classe org.apache.struts.action.DynaActionForm, che estende la classe ActionForm e che differisce da quest'ultima nel modo di definire le sue proprietà e nei metodi validate() e reset().

Le proprietà del bean in questo caso vengono specificate all'interno dello struts-config.xml, evitando in questo modo di modificare delle classi Java che dovrebbero essere ricompilate e sono proprio loro che rendono questo bean dinamico perché vengono inserite al suo interno con i corrispondenti metodi get e set in fase di runtime.

Il metodo reset() invece viene invocato al momento in cui si processa la richiesta, mentre nell'ActionForm questo veniva invocato al momento della creazione o quando viene ripresa una sua vecchia istanza. Inoltre non abbiamo un controllo diretto sul metodo (che infatti non deve essere implementato) a meno che non decidessimo di creare una classe che estenda DynaActionForm e che faccia l'overriding del metodo.

Per quanto riguarda il metodo validate() merita un discorso a parte che affronteremo più avanti, vi basti soltanto sapere che la validazione dei dati del form avviene in maniera un po' più complicata e non soltanto tramite l'implementazione di un metodo.

Vediamo adesso come possiamo utilizzare questa classe.

Per prima cosa, come in tutte le altre classi, va configurata all'interno del nostro struts-config.xml semplicemente aggiungendo un tag <form-bean> all'interno del tag <form-beans> (d'altronde questa classe altro non è che un ActionForm e quindi va impostata nella stessa maniera):

```
<form-beans>
  <form-bean name="provaDynaForm"
          dynamic="true"
          type="org.apache.struts.action.DynaActionForm">
    <form-property  name="campoStringa"
```

```
                    type="java.lang.String "/>
    <form-property  initial=”10”
                    name="campoIntero"
                    type="java.lang.Integer"/>
  </form-bean>
<form-beans>
```

Notiamo che a differenza del tag *<form-bean>* del paragrafo precedente c’è un attributo *“dynamic”* impostato a “true”. E’ proprio questo che specifica al framework di gestire questa ActionForm in maniera dinamica. Inoltre possiamo vedere che le proprietà del bean sono impostate grazie al tag *<form-property>* : in questo caso abbiamo inserito nella classe soltanto due proprietà che verranno caricate dalla classe DynaActionForm all’interno di un *java.util.Map* ponendo come chiave l’attributo *“name”* e come valore quello di default o quello da noi impostato nell’attributo *“initial”*.

L’attributo *“initial”* quindi serve per impostare i valori delle proprietà quando l’applicazione verrà avviata, ma anche quando verrà invocato il metodo *reset()* del bean che li risistemerà nel loro stato originale.

Nell’attributo *“type”* invece andremo a specificare il tipo del parametro (cioè se si tratta di una Stringa, di un intero etc.) ricordandoci di utilizzare sempre le classi di wrapper per i tipi primitivi (*java.lang.Integer* per gli interi, *java.lang.Boolean* per i booleani etc.) anche perché come abbiamo detto prima questi parametri andranno inseriti all’interno di un Map che accetta solo oggetti.

Come si diceva, la validazione dei dati diventa un pò più complicata. La DynaActionForm non offre alcun comportamento predefinito per il metodo validate(). A meno di non porla come sottoclasse (ma si perderebbero alcuni dei vantaggi citati), non esiste un metodo semplice per la validazione. Per fortuna, ancora na volta il framework fornisce una caratteristica adatta chiamata Validator.

3.5 Il framework Validator

Struts Validator è stato creato da David Winterfeldt ed è ora disponibile nella distribuzione principale di Struts. Questo permette la configurazione dichiarativa di routine di validazione per un'applicazione basata su Struts senza bisogno di scrivere alcuna logica speciale di validazione, suppportando già regole di validazione fondamentali come il controllo dei campi richiesti, email, data e ora e altri ancora, dando ovviamente, anche la possibilità di creare con facilità regole di validazione personalizzate.
Tale framwork è sicuramente uno dei punti forti di Struts per le sue potenzialità e per i problemi che va risolvere all'interno della programmazione classica di Struts ed in particolare nella scrittura del metodo validate() della classe Action Form.

Il primo problema, come sappiamo, sta nel fatto che scrivere logica di validazione all'interno di ogni classe ActionForm, immette sicuramente codice ridondante per tutta l'applicazione. Infatti all'interno di una web-application il tipo di validazione che si effettua nei vari form è sicuramente molto simile.
Il secondo problema, ugualmente molto importante, riguarda ci aggiornamenti e le modifiche delle regole di validazione. È chiaro che se la validazione viene svolta nel metodo validate() delle singole ActionForm e nel tempo c'è necessità di cambiare tali regole di validazione, sarà necessario ricompilare le singole classi.

Il framework Validator risolve questi primi problemi, spostando tutta la logica di validazione completamente al di fuori dell'ActionForm e di configurarla in maniera dichiarativa per un'applicazione attraverso l'uso di file di testo esterni scritti in XML. Quindi non è più necessaria alcuna logica di validazione nelle proprie ActionForm, rendendo la web-application più facile da gestire e da aggiornare.
Un altro beneficio sta nel fatto che il Validator è facile da estendere. Offre molte routine standard di validazione, ma se c'è bisogno di nuove regole, il framework può essere facilmente esteso.

Come detto, il framework Validator sposta tutta la logica di validazione in files XML esterni all'applicazione. In particolar modo i files fondamentli sono due: *validation-rules.xml* e *validation.xml*.

3.5.1 Configurare il Validator per l'uso con Struts

Il Validator viene configurato come un plug-in di Struts. Per abiliare il Validator bisogna aggiungere nello struts-config.xml le seguenti righe;

```
<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 e inizializza il Validator; si può notare come vengano definiti i percorsi dei due file XML di configurazione del Validator. Stiamo specifichanodo al framework che quando l'applicazione verrà avviata nel metodo *initApplicationPlugIns()* della classe *RequestProcessor* verrà invocato il metodo *init()* della classe *ValidatorPlugIn* che permetterà di caricare in memoria i files 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.

Si noti che il framework Validator si può utilizzare anche quando si fa uso delle ActionForm classichè (anzichè le DynaActionForm), in tal caso 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.

Nel caso in cui le regole contenute non soddisfino tutte le nostre richieste possiamo decidere di estendere questo file, legendo il dtd (*validation-rules_1_1.dtd*) per la sintassi, con le regole da noi scritte: la cosa migliore da fare è inserirle all'interno di un altro file XML di modo che se ci saranno delle modifiche al framework Validator non saremo costretti a riscrivere tutto nel nuovo file validator-rules.xml.
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.

```
<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>
</global>
</form-validation>
```

name : assegna un nome in maniera univoca alla regola di validazione. Questo valore verrà utilizzato sia all'interno di questo file per referenziare le dipendenze tra le varie regole sia all'interno dell'altro file di configurazione del Validator (che vedremo più avanti);

classname : contiene il nome della classe che si occupa della validazione;

method: il metodo della classe che deve essere invocato per effettuare questa validazione;

methodParams : i tipi dei parametri con i relativi percorsi da passare al metodo separati da virgola;

msg : indica la chiave all'interno del nostro file di properties da cui prendere il messaggio di errore nel caso la validazione non vada a buon fine. Nel caso in cui stessimo utilizzando delle regole già presenti all'interno del validator-rules.xml dovremo o modificare il nome all'interno di quest'ultimo con uno presente nell'ApplicationResources.properties (cosa sconsigliabile perché incontreremmo grosse difficoltà in caso di aggiornamento della versione del file XML del Validator) o inserire queste chiavi all'interno del file di properties della nostra applicazione o utilizzando l'attributo msg all'interno dell'altro file di configurazione;

Il tag javascript, contiene il codice che può essere utilizzato se si vuole eseguire una validazione client-side (di cui si parlerà a breve).
Come possiamo vedere nell'attributo "classname" del tag <validator> è stata inserita la classe org.apache.struts.validator.FieldChecks, una classe di utility specifica per il framework Struts (creata come alternativa alla classe GenericValidator contenuta all'interno del Validator) con all'interno i metodi invocati nel validator-rules.xml. Quando questi vengono chiamati e la validazione non va a buon fine viene creato un ActionError inserito nel request e questo con il suo messaggio di errore viene reso visibile nella parte di view.

Il secondo file di configurazione richiesto dal framework Validator è validation.xml. Questo file associa le regole di validazione contenute all'interno del validation-rules.xml alle varie classi ActionForms.
Andiamo ad analizzare la struttura di questo file, che dovremo creare completamente noi seguendo le indicazioni forniteci dal file validation_1_1.dtd

Un frammento di file validation.xml potrebbe essere il seguente, dove si specificano le regole di validazione per un form di login:

```
<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 (di cui si parlerà). In questo file vengono definiti i messaggi di errore associati a ciascun criterio di validazione definito nel validator-rules.xml ad esempio:

```
errors.required={0} is required.
```

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.

3.5.2 Validazione client-side

Il Validator foirnisce anche il supporto alla validazione client-side dei form dell'applicazione. Le funzioni JavaSctipt associate ai diversi criteri di validazione sono definite nel file validator-rules.xml come già visto. Per abilitare la validazione client-side è sufficiente inserire nelle pagine JSP dell'applicazione il seguente tag:

```
<html:javascript formName="logonForm"/>
```

Il validator genererà dinamicamente le routine di validazione per il form specificato nell'attributo formName del tag <html:javascript>.
Nel nostro caso genererà la funzione validateLogonForm() che effettuerà la validazione del form purchè la si colleghi al submit dello stesso come segue:

```
<html:form action="logonAction" onsubmit="return
validateLogonForm (this)">
```

Chiaramente il submit del form non avverrà se i controlli sui campi non hanno successo. La validazione JavaScript consente di evitare i round-trip con il server dovuti a successivi submit del form conseguenti ad errori di validazione sui campi digitati nel caso in cui i controlli vengano eseguiti esclusivamente lato server. In ogni caso comunque il Validator esegue anche i controlli lato-server una volta che il submit del form viene effettuato. Ciò garantisce che i controlli formali vengano effettuati anche nel caso in cui i controlli JavaScript lato client vengano disattivati, cosa non da poco visto che in genere nelle applicazioni web che eseguono i controlli formali sui campi lato client , le validazioni non vengono eseguite anche lato server come in realtà dovrebbe essere fatto. Ciò conferisce una maggiore robustezza ed affidabilità all'applicazione ed è uno dei principali benefici dell'uso del Validator.

3.6 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. E' 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. E' 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. E' un identificatore per ogni combinazione di lingua e regione geografica, una istanza di questa classe identifica una combinazione language/country. Per istanziare 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.

3.6.1 Internazionalizzazione e Struts

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 standard per le quali si rimanda alla documentazione ufficiale presente sul sito della Sun (http://java.sun.com).

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 public java.util.Locale 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 sie predisposta. Ad esempio

```
ApplicationResource_en_En.properties per inglese/regnoUnito
ApplicationResource_en_US.properties per inlese/Stati Uniti
ApplicationResources_fr_FR.properties per francese/Francia
```

e così via. Il file ApplicationResource_en_En.properties sarà ad esempio del tipo :

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

Nello struts-config.xml va 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.

3.6.2 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. E' 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 mediante il metodo
//getLocale(HttpServletRequest request)
//della classe org.apache.struts.action.Action che restiuisce
il locale corrente //dell'utente
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;
}
```

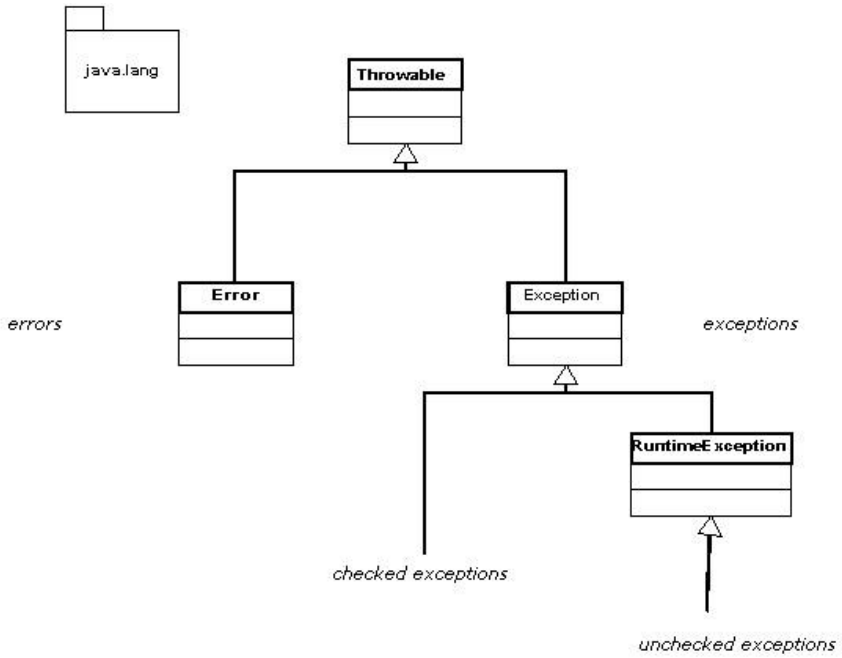
Per reperire le stringhe localizzate nelle pagine JSP si utilizza invece un custom-tag della libreria struts-bean, precisamente il tag <bean:message>. L'utilizzo è banale , basta fornire come attributo la chiave corrispondente alla label che si vuole acquisire come nell'esempio seguente:

<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.

3.7 Le eccezioni in Java

Le eccezioni sono il meccanismo che il linguaggio Java mette a disposizione per gestire eventuali situazioni anomale che possono verificarsi nell'esecuzione di una applicazione. Una eccezione è un oggetto di una determinata classe che viene creato quando una certa situazione di errore si è verificata in conseguenza dell'esecuzione di una applicazione. Le eccezioni fanno parte di una gerarchia di classi che ha come padre la classe java.lang.Throwable come rappresentato in figura.



Da essa discendono due classi fondamentali che sono la java.lang.Exception e java.lang.Error. Le classi che discendono da Error rappresentano situazioni anomale dalle quali non è possibile in genere ripristinare l'applicazione, quali un OutOfMemory o uno StackOverflowError. Le classi che ereditano da Exception, e alle quali facciamo riferimento come eccezioni, invece rappresentano quelle situazioni anomale che lo sviluppatore può intercettare e gestire in base alla logica della propria applicazione.

Le eccezioni si suddividono ulteriormente in checked exceptions ed unchecked exceptions. La suddivisione fa riferimento all'obbligo da parte dello sviluppatore di dover gestire in qualche modo una eccezione o meno. Le unchecked exceptions sono sottoclassi di java.lang.RuntimeException che a sua volta è una sottoclasse di java.lang.Exception. Le eccezioni che discendono da RuntimeException possono essere gestite allo stesso modo di quelle checked ma non vi è l'obbligo di farlo, cioè il compilatore non impone che le eccezioni che estendono la RuntimeException vengano gestite.

La gestione delle eccezioni a livello di linguaggio viene fatta mediante l'uso del blocco try/catch e della clausola throws. Il codice che potenzialmente può generare eccezioni può essere messo all'interno di un blocco try/catch . Qualora si verifichi

una eccezione il flusso elaborativo passa all'interno del blocco catch corrispondente all'eccezione generata, all'interno del quale viene messo il codice per la gestione dell'eccezione stessa, e prosegue con il codice seguente il blocco try/catch. Per le eccezioni di tipo checked questa gestione è obbligatoria, ovvero il compilatore non compilerà con successo la classe se il codice che può generare eccezione non è inserito all'interno di un blocco try/catch , a meno che il metodo stesso non dichiari che la gestione della eccezione è delegata al metodo chiamante. In tal caso si utilizza nella definizione del metodo la clausola throws. Mediante la clausola throws si dichiara quali eccezioni possono essere generate dal metodo in questione informando il chiamante sui propri obblighi in tale senso, tutto secondo la logica del design by contract . Nel caso in cui anche il metodo chiamante dichiari nella clausola throws l'eccezione in questione, l'elaborazione passerà all'ulteriore metodo chiamante e così via fino alla fine dello stack contenente la sequenza dei metodi.

3.7.1 Struts e la gestione delle eccezioni: approccio programmatico o dichiarativo

Dalla versione 1.1 di Struts sono state introdotte nel framework delle componenti che consentono di gestire le eccezioni in modo strutturato ed efficiente. I due approcci possibili nella gestione delle situazioni di eccezione, così come nella gestione di molte altre problematiche applicative ricorrenti nello sviluppo di applicazioni J2EE, sono quello programmatico e quello dichiarativo.
In generale l'approccio programmatico consiste nello scrivere all'interno della propria applicazione codice specifico per la gestione di una determinata problematica mentre l'approccio dichiarativo consiste nel configurare all'esterno dell'applicazione le modalità di gestione del problema stesso.
Nel caso delle eccezioni quindi, gestire una situazione di eccezione in modo programmatico significa inserire all'interno del proprio codice delle istruzioni specifiche per il trattamento dell'eccezione verificatasi.

L'approccio dichiarativo invece presuppone la configurazione all'esterno del codice del comportamento dell'applicazione a fronte del verificarsi di una eccezione.
Il primo approccio è quello tradizionale , utilizzabile in una qualsiasi applicazione Java e non presenta particolari differenze in una applicazione sviluppata con Struts. In questo caso lo sviluppatore deve inserire all'interno delle proprie classi il codice opportuno, ovvero i blocchi try/catch per la gestione delle eccezioni, né più né meno che in una qualsiasi applicazione Java.
Una Action contenente gestione programmatica delle eccezioni potrebbe avere un aspetto simile:

```
try{
  //codice che può generare eccezioni
}
catch(Exception e){
  ActionErrors errors = new ActionErrors();
  erorrs.add("eccezione", new ActionErorrs("eccezione"));
  saveErrors(request,errors);
  return mapping.findForward("eccezione");
}
```

In pratica si tratta di racchiudere il codice tra blocchi try/cath e nel catch corrispondente mettere il codice di gestione dell'eccezione che nell'esempio non è altro che la generazione di un errore da inviare al client, secondo la modalità già nota.
Questo approccio è possibile ed è sicuramente migliorabile personalizzando ad hoc il framework con classi generalizzate che accentrino gran parte della gestione necessaria, ma costringe comunque lo sviluppatore a scrivere codice ad hoc, spesso ridondante, per gestire queste situazioni. Inoltre in questo caso se si vuole modificare il comportamento dell'applicazione al verificarsi di una eccezione è necessario intervenire nel codice.
E' invece sicuramente più elegante ed efficiente sfruttare la possibilità offerta da Struts di gestire in modo dichiarativo le eccezioni. All'esterno del codice , quindi nel file XML di configurazione struts-config.xml, è possibile descrivere mediante opportune sezioni XML il comportamento che l'applicazione deve avere a fronte di una particolare eccezione. In questo modo, non solo si evita di scrivere codice

specifico nell'applicazione per la gestione delle eccezioni, ma si ha la possibilità di modificare il comportamento dell'applicazione senza intervenire nel codice stesso.

3.7.2 Gli strumenti forniti da Struts per la gestione dichiarativa delle eccezioni

Il principio di base è quello di definire esternamente al codice il comportamento dell'applicazione e il conseguente flusso applicativo al verificarsi di una eccezione.
Questa configurazione può essere fatta a livello di singola Action o a livello globale per tutta l'applicazione.
Per configurare la gestione a livello di singola action si utilizza il tag <exception> all'interno della configurazione della Action stessa.
Nel tag <exception> si dichiara la classe dell'eccezione da gestire, la risorsa a cui verrà inoltrato il flusso elaborativo ed una chiave per acquisire dal resource bundle dell'applicazione un messaggio di errore come nell'esempio seguente:

```
<action path"/esempio"
  type="it.esempi.EsempioAction"
  input="input.jsp"
  scope="request"
  name="myForm" >
  <exception key="label.eccezione.esempio"
    type="it.esempi.MiaException"
    path="errorpage.jsp" />
</action>
```

In questo esempio si definisce che a seguito di una ipotetica MiaException il flusso applicativo sarà inoltrato alla pagina JSP errorpage.jsp acquisendo dal resource bundle il messaggio di errore corrispondente alla label label.eccezione.esempio.
In questo modo è possibile configurare tutti i comportamenti in corrispondenza delle possibili eccezioni generate dal metodo execute() di una Action.
E' anche possibile definire un comportamento globale utilizzando il tag <global-exception>.

In tal caso si presuppone la definizione di una o più classi handler per le eccezioni da gestire, ovvero delle classi che estendono org.apache.struts.action.ExceptionHandler ed il cui metodo execute() viene eseguito dal framework al verificarsi della corrispondente eccezione.
La configurazione di una eccezione a livello globale è fatta come nell'esempio seguente:

```
<global-exceptions>
  <exception key="label.eccezione"
    type="it.esempi.MiaException"
    handler="it.esempi.ExceptionHandler"/>
</global-exceptions>
```

Nella sezione <global-exception> si dichiara la classe dell'eccezione da gestire, la classe handler che conterrà la logica di gestione ed una chiave per acquisire dal resource bundle dell'applicazione un messaggio di errore.
Nell'esempio riportato si definisce che la classe it.esempi.ExceptionHandler , che estende org.apache.struts.action.ExceptionHandler, gestisce le eccezioni di tipo it.esempi.MiaException. Ciò significa che al verificarsi di una eccezione di questo tipo il framework manderà in esecuzione il metodo execute() della classe handler così dichiarata.
In questo metodo in genere è opportuno inserire del codice per effettuare il log dell'eccezione, memorizzando tutte le informazioni utili alla comprensione del problema che la ha generata, quali il messaggio associato all'eccezione, lo stack-trace etc. Il metodo restituisce in output un oggetto della classe ActionForward che consente di effettuare il forward alla risorsa desiderata.
E' ovvio che in entrambi i casi non sarà necessario inserire all'interno dell'applicazione alcun codice di gestione delle eccezioni perché il tutto è delegato alle classi handler.
Nella pratica è sicuramente preferibile dichiarare delle global exceptions almeno per quelle eccezioni comuni a tutte le Action, lasciando alla dichiarazione della singola Action solo i casi particolari di eccezioni possibili solo in quel contesto. E' il caso ad esempio delle proprie eccezioni applicative definite per individuare particolari situazioni anomale riferite alla logica della propia applicazione.

E' comunque impensabile dover definire un numero elevato di classi handler per gestire tutte le possibili eccezioni. Se non sia necessaria una gestione estremamente particolareggiata è possibile dichiarare una classe handler per la java.lang.Exception che gestirà quindi in maniera centralizzata tutte le possibili eccezioni.

C'è infine da osservare che vi sono alcuni casi in cui non è possibile non inserire blocchi try/catch all'interno del codice. Basti pensare al caso di codice che esegue in modo programmatico una transazione su un database e che necessita di gestire le situazioni di eccezione per effettuare correttamente il commit o il rollback della transazione stessa.

In tal caso lo sviluppatore è obbligato ad inserire uno o più blocchi catch al cui interno eseguire l'operazione di rollback per annullare la transazione corrente.

Struts e le tecnologie per il livello View

4.1 Introduzione

Come abbiamo spiegato, quando abbiamo trattato il pattern MVC, il livello view è la rappresentzione di un modello tramite un'interfaccia utente. Metaforicamente, una view è una finestra che i client posso usare per vedere lo strato del modello e la prospettiva può essere diversa a seconda della finestra da cui il client sta guradando, in altre parole uno stesso model può avere, ed in un web application sicuramente ha più viste. Anche se come chiarito, la view non contiene in alcun modo logica applicativa né di controllo è un livello sicuramente molto importante, perchè è quello a più diretto contatto coni l client e dunque con l'utente il quale è lui a decretare il successo di una applicazione web. Ecco perché la sua progettazione e realizzazione è importante come tutto il resto dell'applicazione. Inoltre le tecnologie utilizzabile in

questo livello sono tante da poter interessare un successivo lavoro di tesi, limitandoci in questa sede solo ad una panoramica.
Come più volte detto, nell'implementazione del pattern MVC, Struts si occupa essenzialmente del controller, lasciando completa flessibilità nella scelta delle tecnologie da adottere nalla view, anche se è oramai abitudine adottare la tecnologia JSP, tanto che nello stesso pacchetto di Struts troviamo una serie di librerie di tag per facilitare la creazione di pagine JSP, ma ribadiamo, non è una regola.

4.2 Template Engines

Prima di parlare della tecnica più utilizzata, facciamo una breve panoramica sulle alternative.
Tra le più rilevanti, ci sono una serie di *Template Engines* caratterizzati da una separazione tra il design e i dati contenuti nelle pagine. Questa metodologia offre diversi vantaggi, come più flessibilità a livello design e maggiore facilità di sviluppo per gli addetti alla grafica che non hanno a che vedere con tutto ciò che c'è sotto.

4.2.1 Apache Cocoon e Cocoon plug-in

Cocoon è un framework open source facente parte del progetto Jakarta che come Struts implementa il pattern MVC basandosi su file esterni XML. Cocoon fa uso fi *pipelines* (particolari percorsi per il flusso di trasformazioni) per creare dati XML da varie sorgenti e poi trasferire questi dati in varie tecnologie di presentazione atteaverso l'uso di XSLT (Extensible Stylesheet Language Transformations)
In seguito alla grande espansione di Struts, Cocoon è passato dall'esserne un rivale ad esserene un plug-in, che da la possibilità a Struts di inoltrare i dati a questo framework per sfruttare i suoi pipelines.

4.2.2 Freemarker

Anche Freemarker è un prodotto open-source: si tratta di un motore che gestisce template HTML per servlet Java. Con Freemarker, si salva l'HTML in template, i quali alla fine in maniera dinamica vengono compilati in "oggetti template". Questi oggetti template generano poi l'HTML in maniera dinamica, utilizzando dati forniti dalle servlet. Freemarker Utilizza un suo proprio linguaggio per i template che ha una velocità prossima a quella delle pagine HTML statiche.

4.2.3 Velocity

Velocity è un altro progetto dell'ambito Jakarta come Struts. Si tratta di un template engine basato su Java. Simile a Freemarker ma capace di andare oltre la semplice creazione di contenuti dinamici per siti web. Velocity a partire dai template può generare SQL, PostScript e XML e può essere utilizzato sia come strumento standalone per generare codice sorgente e report che come componente integrato in altri sistemi. Difatti esiste VelocityStruts che rappresenta una estensione di Struts.

4.3 Le librerie di custom-tag di Struts

Struts sfrutta la caratteristica delle librerie di tag JSP con l'inclusione di svariate categorie di tag che contribuiscono a rendere lo strato di presentazione più facilmente gestibile e riutilizzabile. Usando le librerie di custom-tag di Struts, gli sviluppatori possono interagire con il resto dell'infrastruttura senza includere codice Java nelle pagine JSP, aspetto fondamentale per rispettare il pattern MVC.

Quando il browser legge un file HTML determina come elaborare e gestire i tag contenuti nel file sulla base di un set standard di tag. Il proposito dei custom-tag JSP consiste nel dare allo sviluppatore la possibilità di estendere il set di tag che possono essere usati all'interno di una pagine JSP. Con i semplici tag HTML, è il browser che contiene la logica per l'elaborazione dei tag e per la presentazione dell'output. Con i custom-tag JSP la funzionalità è collocata in una classe speciale Java detta *tag handler*.

Il tag handler è una classe che esegue il comportamento specifico del tag. Implementa una interfaccia dei diversi tag, a seconda del tipo di tag di cui si necessita. La classe handler ha un accesso a tutte le risorse JSP come ad esempio l'oggetto PageContext e gli oggetti request, response e session. Il tag è anche integrato con le informazioni dell'attributo, così da adeguare il proprio comportamento sulla base dei valori dell'attributo.

- I custom-tag JSP offrono notevoli vantaggi rispetto a scrivere scriptlet o codice Java nelle proprie pagine JSP:
- I tag sono riutilizzabili e permettono di risparmiare tempo prezioso nello sviluppo;
- I tag possono essere personalizzati con attributi sia statici che dinamici;
- I tag hanno accesso a tutti gli oggetti presenti in una pagina jsp, incluse le variabili request, response e output;
- I tag possono essere annidati, dando luogo ad interazioni più complesse all'interno di una pagina JSP;
- I tag semplificano la leggibilità di una pagina JSP.

In generale l'uso dei tag JSP porta avanti i concetti di riusabilità, dal momento che il comportamento è implementato in un solo luogo, il tag handler, e non è replicato in tutte le pagine JSP.

Una libreria di tag è un insieme di tag personalizzati JSP raggruppati secondo una logica di impacchettamento. Anche se non è abbligatorio, i tag all'interno di una libreria dovrebbero risolvere problemi di tipo simile.

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 istanze di bean:
- *Logic Tag* usati per logica condizionale, iterazioni e controllo di flusso ;
- *Nested Tag* che estendono le funzionalità dei tag base.

Annessi ad una libreria di custom tag ci sono:

- una classe handler la cui funzione si è già discussa;
- un file descrittore della libreria (TLD) che rappresenta la solito un file XML con metainformazioni sulla libreria, già presente nel pacchetto di istallazione di Struts, che va copiato nella nostra web application; dichiarato all'interno del descrittore di deploy dell'applicazione (il file web.xml di cui si è parlato) e dichiarato all'interno di ogni singola pagina JSP che faccia uso di uno dei tag che contiene.

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.

Non è questa la sede per una descrizione dettagliata dei vari tag , per la quale si rimanda comunque alla documentazione ufficiale del framework alla pagina:

http://jakarta.apache.org/struts/userGuide/

Va piuttosto ribadito che l'utilizzo dei custom tag nelle pagine JSP è altamente consigliato per evitare il proliferare di scriptlet Java nelle pagine e per rendere standard la scrittura delle stesse.

4.4 La JSP Standard Tag Library (JSTL)

Il gruppo che definì per primo le specifiche delle JSP, avevano come obiettivo quello di definire un set di tag standard delle JSP all'interno di tali specifiche. In questo modo i produttori non avrebbo potuto generare le loro versioni dei tag ma avrebbero permesso agli sviluppatori di contare sul fatto che questo insieme standard di tag fosse disponibile in tutti i containers a norma. Tuttavia a causa di limiti temporali, questa caratteristica non fu inclusa nelle prime specifiche delle JSP.

Da allora molti produttori hanno creato le proprie versioni di librerie di tag più comunenmente usate, ma queste versioni sono risultate tanto diverse da non permettere che gli sviluppatori possano trasferire le loro pagine JSP da un container ad un altro senza modificare le pagine stesse. La JSR 52, ovvero la JSP Standard Tag Library (JSTL) punta a risolvere questo problema, fornendo una libreria di action standard, con la quale fornire ai programmatori un patrimonio comune di tag in grado da un lato di garantire la sostanziale compatibilità delle pagine JSP al variare del Web Container e dall'altro di fornire quei costrutti di programmazione necessari per svincolare il framework di Sun dal continuo ricorso agli scriptlet Java.

Tale libreria di tag, sviluppata in seno al Java Community Process dall'Apache Group consiste in una collezione di Custom Tag Library (simili aquelli di cui is è parlato prima), organizzate e suddivise per categorie funzionali.

Si noti che JSTL utilizza funzionalità ed API delle Custom Tag Library introdotte a partire dalla versione 1.2 di JSP, e non risulta pertanto compatibile con Web Container che implementino versioni precedenti di questo framework.

JSTL si compone di quattro Custom Tag Library, ognuna delle quali viene mappata in namespace (detti anche prefix in gergo JSP) ed associata ad URI differenti e stabiliti per convenzione:

Libreria	*URI*	*Namespace*
core	http://java.sun.com/jstl/core	c
XML processing	http://java.sun.com/jstl/xml	x
I18N capable formatting	http://java.sun.com/jstl/fmt	fmt
relational db access (SQL)	http://java.sun.com/jstl/sql	sql

- : tale libreria, fornisce tag sostanzialmente paralleli ai costrutti strutturati di programmazione in Java, al fine di ridurre al minimo il bisogno di innestare scriptlet direttamente all'interno delle pagine JSP. La libreria core implementa inoltre alcune utility action volte alla manipolazione di URL e contenuti esterni;
- : libreria orientata alla gestione di documenti XML e XSLT;
- capable formatting: libreria di tag rivolta alla localizzazione delle applicazioni;
- libreria di tag rivolta all'utilizzo di connessioni JDBC dall'interno delle pagine JSP; si noti che tale libreria, in maniera piuttosto evidente, non rispetta la separazione dei ruoli tipici delle applicazioni multi-tier, dal momento che consente di integrare sezioni di business logic all'interno di pagine JSP, vale a dire del presentation layer. Tuttavia la disponibilità di facility per la gestione diretta di database dall'interno dello strato di interfaccia utente gioca un ruolo importante nell'implementazione di sistemi prototipo e di applicazioni di complessità elementare, oltre ad avere un preciso appeal a livello di marketing.

La libreria JSTL introduce anche un nuovo linguaggio di espressione (EL – *Expression Language*) per rendere più facile agli autori di pagine web scrivere codice che acceda ai dati delle applicazioni senza costringerli ad apprendere un linguaggio di programmazione completo come Java.

E' molto importante inoltre osservare che alcuni dei tag forniti nelle librerie di Struts si sovrappongono per la loro funzione con i tag presenti nelle JSTL.

In particolare i tag della libreria logic di Struts trovano corrispondenza in molti dei tag della libreria core delle JSTL. L'indicazione è di usare sempre questi ultimi in quanto sono già lo standard per la scrittura delle pagine JSP e sicuramente verranno utilizzati nelle versioni future.
Probabilmente la libreria di tag di Struts che avrà ancora largo utilizzo è la html almeno finchè la tecnologia delle Java Server Faces non si sarà diffusa ed affermata.

In molti casi si impiegano i tag delle varie librerie in associazione con i JavaBeans, le cui proprietà corrispondono ai campi di imput dei form HTML. In molti casi, tuttavia, i bean saranno oggetti dai valori comuni appartenenti al livello modello. Questi bean possono avere quasiasi scope: di pagina, di request di sessione o di applicazione.

UN CASO DI STUDIO: UN NEGOZIO DI DVD ON-LINE

5.1 Scopo

A questo punto del lavoro di tesi, è sembrato necessario introdurre l'analisi di un caso di studio, per un duplice motivo.

In primo luogo per avvalorare quanto detto fin ora riguardo alle potenzialità, alla facilità d'uso e alle caratteristiche di questo valido framework.

In secondo, per fornire una sorta di guida applicativa alla realizzazione di web application Struts, che possa essere un punto di partenza per futuri lavori.

In questo capitolo viene mostrato come creare un'applicazione completa usando Jakarta Struts a partire dalle specifiche di progetto applicando molti dei concetti di cui si è parlato. Ovviamente il risultato vuole essere una applicazione dimostrativa e non pretende di essere un'applicazione reale, sebbene completa.

Prima di addentrarci nel progetto e nell'implementazione della nostra applicazione ci sono ancora due punti da trattare, ovvero la struttura che deve avere una generica web-application e la preparazione dell'ambiente di sviluppo.

5.2 Struttura di una applicazione web

La nozione di applicazione web è stata formalizzata per la prima volta nella versione 2.2 delle specifiche delle servlet Java. Sebbene il concetto di applicazione web sia piuttosto ampio e dipendente dal contesto in cui viene esaminato, la definizione fornita nelle suddette specifiche risulta di portata sufficientemente generale.Dal punto di vista dello sviluppatore di applicazioni web basate su Java risulta particolarmente importante la standardizzazione della struttura di tali applicazioni e l'introduzione degli archivi WAR come strumento per il *deployment* delle applicazioni web.

Una applicazione web è una gerarchia di file e directory disposti secondo uno schema standard.

/webapp. Una applicazione web ha una propria directory radice (*root*). Il nome della cartella corrisponde a ciò che nella terminologia delle servlet viene chiamato *context path* (nel seguito supponiamo che tale nome di tale cartella sia `webapp`). La directory radice contiene tutti gli altri elementi che compongono l'applicazione web.

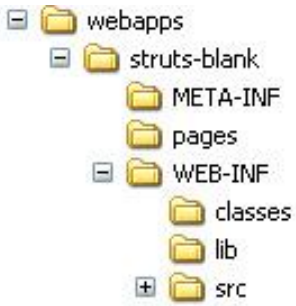
File HTML, JSP, ... La directory `/webapp` contiene i documenti HTML, le pagine JSP, le immagini e le altre risorse che compongono l'applicazione web. Tali file possono essere strutturati in directory esattamente come si farebbe per un sito web statico.

/webapp/WEB-INF. All'interno della cartella *root* si trova una directory speciale, denominata `WEB-INF`. La funzione di questa cartella è quella di contenere file riservati, come ad esempio il file di configurazione dell'applicazione, `web.xml`. Per questo motivo le specifiche impongono che la cartella ed il suo contenuto debbano risultare inaccessibili per gli utenti dell'applicazione.

/webapp/WEB-INF/web.xml. Il file `web.xml` comunemente denominato *deployment descriptor*. Si tratta del file di configurazione dell'applicazione web; in esso, ad esempio, si definiscono gli *alias* delle servlet, i parametri di inizializzazione, le mappature dei percorsi e così via come visto nel secondo capitolo.

/webapp/WEB-INF/classes e /webapp/WEB-INF/lib. Queste due directory sono destinate a contenere le classi e gli archivi Jar di cui la nostra applicazione web necessita. Se dobbiamo utilizzare delle classi Java compilate (file con estensione `.class` andremo a copiarle all'interno di `/webapp/WEB-INF/classes`, secondo l'usuale struttura di directory Java (così se abbiamo una classe `MiaServlet` contenuta nel package `sito.servlet`. il percorso del file compilato sarà `/webapp/WEB-INF/classes/sito/servlet/MiaServlet.class`). Se dobbiamo utilizzare classi contenute in archivi Jar, invece, sarà sufficiente copiarli in `/webapp/WEB-INF/lib`.

/webapp/WEB-INF/src. Questa cartella contine i file sorgenti Java o più in generale interi package non compilati.



5.2.1 Archivi WAR

Una volta sviluppata una applicazione web possiamo "impacchettarla" in un archivio WAR (Web Application Archive). Un archivio WAR non è altro che un archivio Jar,

una vecchia conoscenza dei programmatori Java, la cui estensione viene modificata in `.war` semplicemente per indicare che il contenuto dell'archivio è una applicazione web.

Gli archivi WAR hanno una duplice funzione: oltre a quella ovvia di comprimere tutte le componenti di una applicazione in un unico archivio, costituiscono il formato di *deployment* dell'applicazione stessa. Il termine inglese *deployment* indica la messa in opera dell'applicazione.

Uno scenario tipico è quello in cui l'applicazione viene sviluppata localmente per poi essere trasferita, ad ultimazione, sul server vero e proprio (detto anche *di produzione*). In questo caso è sufficiente creare il file WAR, copiarlo sul server di produzione e seguire le istruzioni per l'installazione del WAR relative al *servlet container* che si sta utilizzando. Nel caso di Tomcat, l'installazione di un WAR è estremamente semplice: è sufficiente copiare il file nella directory `webapps` e avviare (o riavviare) il server, che provvederà da solo a decomprimere l'archivio e ad avviare l'applicazione.

Un esempio di comando per la creazione di un archivio WAR (da digitare dopo essersi spostati all'interno della cartella `webapp`) è il seguente:

```
jar cvf webapp.war .
```

E' consigliabile, in ogni caso, fare riferimento alla documentazione ufficiale del tool `jar` per una spiegazione esaustiva delle opzioni disponibili.

Concludendo, organizzando le nostre applicazioni in tal modo otterremo diversi vantaggi:

- aumenteremo la portabilità dell'applicazione, rendendone più semplice la messa in opera;
- renderemo l'applicazione più comprensibile ad altre persone, in quanto sarà molto più semplice sapere dove si trovano le classi, gli archivi Jar e le altre componenti dell'applicazione stessa;
- potremo racchiudere l'intera applicazione in un archivio WAR, rendendone semplice l'installazione su tutti i servlet container conformi alle specifiche Sun.

Precisiamo che quanto appena detto, fa riferimento ad una web-application in senso lato e non specificatamente realizzata con Struts.

5.3 Sicurezza nelle web-applications

Quando si sviluppa una applicazione web anche se non con Struts, solitamente, questa avrà delle pagine o delle risorse non accessibili a tutti. Dunque diventa importante rendere la propria applicazione "sicura". Per fare questo, bisogna, essenzialmente, risolvere due problemi, quello dell'autenticazione e dell'autorizzazione dell'utente.
Il primo problema risponde all'esigenza di verificare se l'utente è chi dice di essere, il secondo all'esigenza di verificare se l'utente già autenticato ha i diritti di usare una particolare risorsa.
Per risolvere questi due problemi, ci sono fondamentalmente due tipi di approcci, il primo che delega il compito alla stessa web-application, il secondo che va a monte e che risolve questi problemi di sicurezza agendo sul web container; ovviamente ogni soluzione comporterà vantaggi e svantaggi.

Risolvere il problema all'interno della web application stessa, vuol dire andare ad implementare dei controlli sull'utente da inserire in punti opportuni.
Come sappiamo, l'autenticazione si intende l'operazione di riconoscimento di un utente come utente registrato nel sistema, richiesta nel momento in cui questo tenta di accedere ad una risorsa personale. Un protocollo di autenticazione solitamente avviene richiedendo all'utente di inserire degli identificativi, ad esempio username e password, che verranno poi ricercati all'interno di un DB. Se la ricerca non produce risultati, verrà segnalato un errore all'utente ed eventualmente verrà riportato nella pagina di log-in, se presente, la ricerca restituirà i dati dell'utente che solitamente verranno memorizzati in una variabile di sessione e vi resteranno fino al momento del logout o alla chiusura del browser. Effettuata l'autenticazione, ogni qual volta

l'utente tenta di accedere a risorse personali, l'applicazione andrà a verificare se l'utente è già autenticato, andando a verificare che esisti una sessione aperta nel suo browser e che contenga i sui dati personali.
Si capisce quindi, che per risolvere il problema dell'autenticazione in Struts, una soluzione è quella di inserire all'inizio di ogni action che fornisce un servizio personale, un controllo sulle variabili di sessione, in questo modo:

```
HttpSession session = request.getSession();
User user = (User) session.getAttribute("user");
if (user == null){
       return (mapping.findForward("login"));
}
```

In pratica se esiste una sessione aperta, andiamo a leggere la variabile di sessione user, se questa esiste, vuol dire che l'utente si è precedentemente autenticato e quindi si prosegue con l'esecuzione dell'action, viceversa, si invia l'utente alla pagina di login, dove effettuare l'autenticazione.

Effettuata l'autenticazione, il passo successivo è quello dell'autorizzazione dell'utente, necessaria quando nella nostra applicazione ci sono utenti con permessi differenti. L'esempio tipico è quello di distinguere un utente registrato comune dall'amministratore di sistema. Una soluzione banale per risolvere il problema, è quella di inserire tra i dati memorizzati dell'utente un ulteriore campo che indichi il ruolo dell'utente. Nel caso semplice in cui bisogna distinguere solo tra due ruoli, è sufficiente inserire una campo booleano che valga '1' per utenti con diritti di amministratore e '0' per gli altri. A questo punto, per garantire l'esecuzione di una funzione, nel caso di Struts di un Action, solo da parte di utenti con diritti di amministratori, sarà sufficiente introdurre un ulteriore controllo dopo quello per l'autenticazione.

```
HttpSession session = request.getSession();
User user = (User) session.getAttribute("user");
// autenticazione utente
...
//controllo sull'autorizzazione
```

```
if (!user.isAdministrator()){
       return (mapping.findForward(“unauthorized”));
}
// esecuzione action
```

La soluzione appena descritta per rendere la propria web-appication sicura, è sicuramente efficiente, ma subito si comprende che è applicabile nel caso di piccole applicazioni, semplici e costituite da poche action altrimenti il carico di lavoro diventa consistente.

In realtà una prima soluzione di questo problema è offerta sempre da Struts. Consiste nello sfruttare proprio la natura centralizzata del controllore che ci offre. Come abbiamo spiegato, a partire dalla versione 1.1 la ActionServlet (la action centrale) di Struts è stata affiancata dalla classe RequestProcessor, diventando anche esse una classe chiave del controllore. Se andiamo a rivedere il flusso delle informazioni quando avviene una richiesta, ci accorgiamo che per ogni richiesta viene richiamato il metodo processPreprocess () di questa classe.

```
  protected boolean processPreprocess(HttpServletRequest request,
                                      HttpServletResponse response)
```

Questo è un metodo che non fa nulla ed esiste solo per permetterne l’override da parte del programmatore. Appare chiaro che essendo un metodo attraverso cui transitano tutte le richieste, è il luogo ideale dove inserire controlli sull’autenticazione dell’utente una volta per tutte. Quindi non avremo più bisogno di andarli ad inserire all’interno di ogni action. Una possibile personalizzaizone del processPreprocess è la seguente:

```
public class CustomRequestProcessor extends RequestProcessor
{
protected boolean processPreprocess(
                            HttpServletRequest request,
                            HttpServletResponse response)
       {
       boolean continueProcessing = true;

       // verifica se la richiesta è una richiesta di log-in
       try{
              HttpSession session = null;
              //verifica se la sessione è valida
              if(request.isRequestedSessionIdValid())
```

```
                session = request.getSession();
            else
            response.sendRedirect("home.jsp?invalid=yes");

            // prelevai il path della richiesta corrente
            String path = processPath(request, response);
            //se non è una richiesta di login o di join
            //l'utente deve già essere autenticato
            if ((!path.equals((String) "/login"))&&
                  (!path.equals((String) "/join")))
            {
             Utente user
               =(Utente)session.getAttribute("user");
            // insure user has logged on
            if (user == null){
            try{
            response.sendRedirect("home.jsp?invalid=yes");
            }catch(Exception ioe)
            {
            continueProcessing = false;
            }
            return continueProcessing;
      }
}
```

Il codice risulta abbastanza chiaro, si effettua innanzitutto un controllo sul path della richiesta, all'utente è lasciata la possibilità di accedere senza autenticazione solo alla funzione di login e di registrazione, dunque se il path della richiesta è diverso, viene effettuata l'autenticazione così come visto precedentemente.

Si noti che riscrivendo il processPreprocess() risolviamo solo il problema dell'autenticazione, quello dell'autorizzazione, dovrà necessariamente essere risolto volta per volta come spiegato prima.

5.3.1 Specifiche di sicurezza J2EE

Le specifiche J2EE sulla sicurezza definiscono un meccanismo di sicurezza dichiarativo, nel quale i parametri di sicurezza vengono espressi in file di configurazione esterni letti all'avvio dal web container, dunque questa volta è lui a farsi carico della sicurezza dell'applicazione.

Secondo le specifiche sono necessari quattro passi per garantire caratteristiche di sicurezza alla propria applicazione.
Il primo passo consiste nel definire i ruoli di chi usa la nostra web application. I ruoli dunque sono un insieme di permessi che si danno ad un gruppo di utenti. Come prima, l'esempio tipico è la distinzione tra l'utente registrato che effettua gli acquisti e l'utente con i diritti di amministratore.Il secondo passo consiste nel definire dei "reami di autenticazione" (realms) all'interno del file *server.xml* del nostro web container. Un reame identifica un insieme di utenti, le loro passwords e i ruoli ad essi associati. La maggior parte dei Servlet Engine (come Tomcat) e degli application servers (come Jboss) fornisce essenzialmente i seguenti reami:

- JDBCRealm - Le informazioni di autenticazione sono memorizzate in un database relazionale accessibile con il relativo driver JDBC.
- JNDIRealm - Le informazioni di autenticazione sono reperibili da un LDAP server o un JNDI server.
- MemoryRealm - Le informazioni di autenticazione sono reperibili in un oggetto in memoria caricato automaticamente all'avvio dell'application server partendo da un file di configurazione XML (conf/tomcat-users.xml per Tomcat).

E' possibile inoltre specificare più reami e scegliere quale utilizzare per la singola Web Application indicandolo opportunamente nel deployment descriptor.

Se abbiamo un database che contiene i dati di tutti gli utenti, la soluzione più comune è utilizzare nel server.xml un JDBCRealm questo richiede di creare a partire dalla tabella degli utenti, una nuova tabella con solo due campi: l'username e il ruolo, se ad un username sono associati più ruoli, questo comparirà più volte. Fatto questo, basta definire i parametri del JDBCRealm nel file xml.

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
       debug="99"
       driverName="com.mysql.jdbc.Driver"
       connectionURL="jdbc:mysql://localhost/myDB"
       connectionName="muraf"
```

```
        connectionPassword=”18mur80”
        userTable=”users”
        userNameCol=”user_name”
        userCredCol=”user_pass”
        userRoleTable=”userRoles”
roleNameCol=”role_name” />
```

A questo punto il container ha le informazioni necessarie per poter gestire l’autenticazione e l’autorizzazione degli utenti. Si noti, però, che per un container è possibile definire solo un reame per volta, quindi se questo dovrà ospitare più aplicazioni, queste dovranno avere gli stessi reami.

Il terzo passo di questo processo teso a rendere sicura la propria applicazione è dichiarare delle aree dell’applicazione, che si intendono proteggere, andando modificare il descrittore di deploy (il file web.xml). In realtà, se si sta realizzando un’applicazione Struts, è possibile dichiarare le action da proteggere all’interno dello struts-config.xml, mentre le altre risorse, come le pagine Jsp, all’interno del web.xml.
Proteggere un’action, è molto semplice, basta aggiungere l’attributo *roles* nel tag action e specificare quali ruoli, definiti precedentemente, hanno diritto di accesso. Sarà compito del RequestProcessor effettuare gli opportuni controlli all’atto dell’invocazione dell’action.
Agendo sul file web.xml si possono invece proteggere anche altre risorse, ad esempio vietare l’accesso ad una intera directory dell’applicazione. Per fare questo basta inserire opportune dichiarazioni nel tag *security-constraint* , si rimanda al DTD del web.xml per i dettagli.
Ora che sono state fornite al sistema tutte le informazioni necessarie per l’autenticazione e autorizzazione, dobbiamo solo specificare queste ultime con che meccanismo devono avvenire.
Le modalità a disposizione degli application servers per accertare l'autenticità e le autorizzazioni dell'utente collegato sono definite nella specifica e si riassumono nelle seguenti 4 parole chiavi definite nel Document Type Descriptor (DTD) della generica Web Application:

- BASIC: Il server invia una richiesta di autenticazione che viene interpretata dal Browser. Quest'ultimo presenta all'utente una finestra di popup nella quale inserire username e password che vengono spediti al server e quest'ultimo può verficarne la rispondenza con la coppia username/password definita nel reame.

- FORM: In questo caso il server invia una pagina HTML configurabile contenente una Form Web dove vengono richieste username e password, la quale va a richiamare, una volta inviate le informazioni, una servlet di autenticazione nei confronti dei reami di sicurezza.Questo metodo è usato quando si vuole presentare all'utente una finestra curata graficamente realizzata in HTML piuttosto della asettica finestra di popup prevista dalla modalità BASIC

- CLIENT-CERT: In questo caso il server richiede al Browser client l'invio di un certificato digitale che sarà sottoposto ad opportuno riconoscimento. Il certificato digitale è sostanzialmente un documento contente la chiave pubblica del client che ne permette l'identificazione che è stato crittato con un meccanismo "one-way" con la chiave privata di una Certification Authority. La certification Authority è un ente riconosciuto da entrambe le parti come fonte autorevole di validazione della chiave pubblica; una sorta di "notaio" che certifica (da cui il nome certificato) la validità della chiave pubblica. Tale Certification Authority può essere sia una azienda riconosciuta come fornitrice di servizi del genere, sia la Consob stessa, posto che gli utilizzatori dei servizi le "facciano fiducia" ovvero la riconoscano come tale nell'ambito dei servizi stessi.

- DIGEST: Questa modalità prevede l'invio della coppia username/password successivamente ad un procedimento di "Hashing". Questo procedimento consiste nell'applicare un algoritmo crittografico "one-way" che renda virtualmente impossibile la decodifica.In questa modalità, simile alla BASIC, si ha il vantaggio di proteggere l'invio della password.

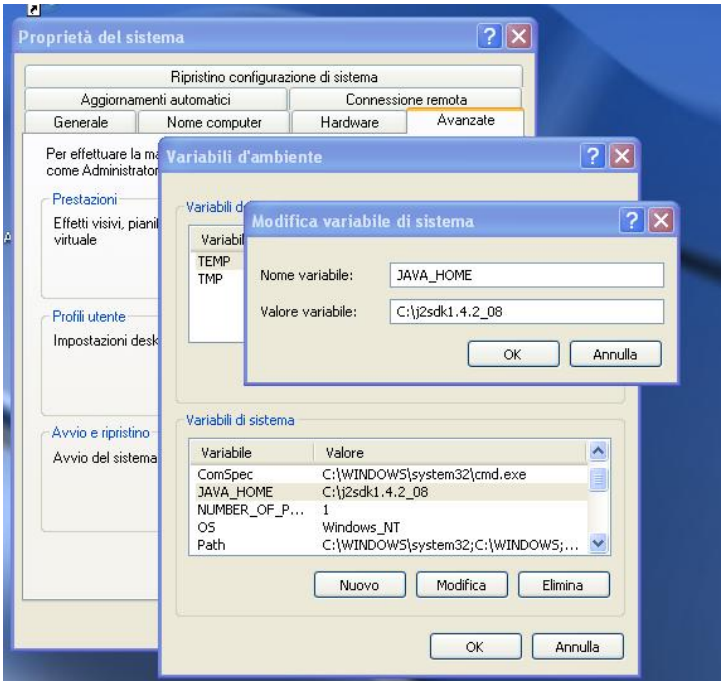
5.4 Preparazione dell'ambiente di sviluppo

Per lavorare con il framework Struts, bisogna preparare la nostra macchina con una serie di applicativi, alcuni fondamentali, altri molto utili per rendere il lavoro quanto più semplice possibile. Difatti, essendo Struts il framework più diffuso nell'ambito dello sviluppo delle web applications adottato dagli sviluppatori java, sono anche nati una vasta serie di applicativi, alcuni freeware ed open source altri commerciali, che facilitano enormemente il lavoro del programmatore. Nel seguito, prepareremo un ambiente di sviluppo, su piattaforma Windows Xp.

5.4.1 Java, Struts e Tomcat

Come prima cosa, dobbiamo scarica la Java 2 Standard Edition (J2SE - SDK) versione 1.2 o superiore dal sito della Sun http://java.sun.com/j2se/ ed istallarla. Difatti, abbiamo bisogno del compilatore javac.exe (non presente nella Java Runtime Environment), per compilare le pagine jsp.

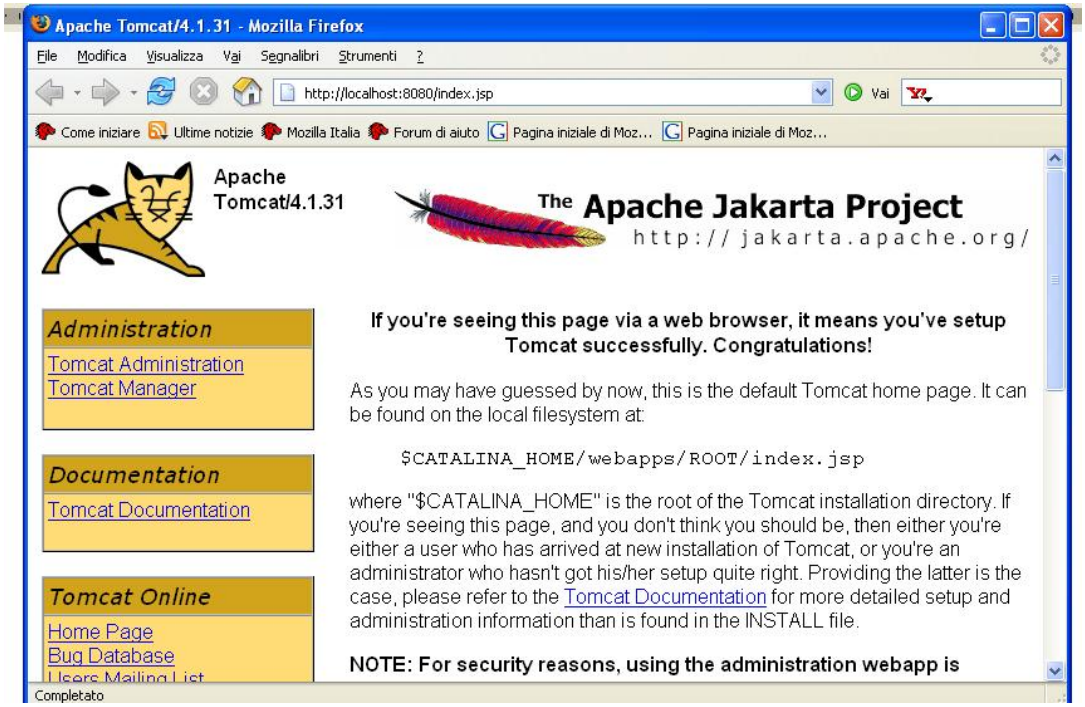
Istallato il pacchetto seguendo le istruzioni, andremo a settare nel nostro sistema, le variabili di ambiente: JAVA_HOME con valore il percorso in cui abbiamo istallato la J2SE e PATH con il percorso del compilatore. Come mostrato in figura:



Poiché, come abbiamo capito, Struts non è altro che un insieme di servlet di controllo, abbiamo bisogno di un Servlet Container.

Una delle possibili scelte è Tomcat a partire dalla versione 3.1 (ormai arrivato alla versione 5). Esistono in realtà anche altre scelte, commerciali e non come: WebSphere, Weblogic Application Server, Resin, Jetty, Jrun, etc.

Tomcat è un Application Server conforme alle specifiche J2EE ed è Open Suorce. Per essere precisi Tomcat è un Servlet Container ed un JSP Engine. Un motore quindi in grado di eseguire lato server applicazioni web basate su architetture J2EE e costituite da componenti Servlet e da pagine JSP. Fa parte anche egli del progetto Jakarta (dunque pienamente compatibile con il framework Struts) ed è scaricabile gratuitamente all'indirizzo http://jakarta.apache.org/ . Si tratta di un servlet container di tipo autonomo, ma che può funzionare anche da tipo aggiuntivo, fornendo i suoi servizi ad un altro server che di solito è Apache. A partire dalla versione 4 l'istallazione è molto semplificata e non richiede il settaggio manuale di particolari parametri. Una volta istallato, per verificare il suo funzionamento, sarà sufficiente lanciarlo tramite l'icona 'Start' e digitare nel nostro browser l'indirizzo locale: http://localhost:8080 e ci apparirà la schermate introduttiva di Tomcat con link alla documentazione ed ad una serie di esempi contenuti nel pacchetto.



Tomcat 4 è un Servlet Container compatibile con la versione 2.3 delle servlet e la 1.2 delle jsp. Il pacchetto di istallazione, infatti, oltre ad una serie di file jar utilizzati da Tomcat, contiene il file servlet.jar necessario per compilare le applicazioni che contengono le servlet e dunque le applicazioni che usano Struts. Inoltre un altro componente fondamentale, già presente in Tomcat, è il parser XML compatibile con le Java API Parsing specification 1.1 e superiori, necessario, per poter interpretare i file web.xml e struts-config.xml dell'applicaizone che andremo a scrivere.

Per poter lanciare la nostra web-application tramite Tomcat, il modo più semplice è di copiarla ed incollarla nella cartella chiamata 'webapps', situata nel directory principale di Tomcat e poi richiamarla dal browser tramite l'url: http://localhost:8080/nomeWebApplication.

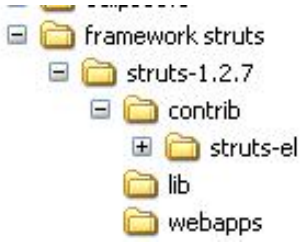
In alternativa, si può editare il file server.xml del nostro web container ed impostare la nostra directory di lavoro come directory di Tomcat. In questo modo:

```
<Context path="/nome"
       docBase="pathDirectory">
   </Context>
```

dove nome dovrà venire sostituito dal nome della directory virtuale che verrà richiamata nel browser e pathDirectory dal path della directory fisica dove saranno

inserite le pagine JSP da visualizzare. Possono naturalmente venire inserite un numero illimitato di directory, basta ripetere questo punto cambiando i nomi della directory virtuale e fisica.

Una volta preparato tutto l'occorrente, potremmo passare a scaricare il codice binario del nostro framework, dal seguente indirizzo: http://struts.apache.org.
Al momento, l'ultima versione di Struts è la 1.2.7.
Una volta spacchettato il file .zip al suo interno troveremo una serie di file jar necessari al funzionamento di Struts e file che dovremmo inserire nella nostra web-application.



in particolare:
lib/jdbc2_0-stdext.jar -
Le classi JDBC 2.0 Optional Package API. Questo file, se stiamo utilizzando come servlet container Tomcat, lo troveremo già contenuto nella directory %TOMCAT_HOME%\common\lib.
lib/struts.jar -
Questo JAR file contiene tutte le classi facenti parte del progetto Struts.
lib/struts-*.tld –
Questi sono i files "tag library descriptor" che descrivono i custom tags nelle Struts tag libraries.
lib/*.dtd –
I files descrittori per il web.xml (secondo le specifiche 2.2 e 2.3 delle Servlet API) e per lo struts_config.xml (struts-config_1_0.dtd nel caso della versione 1.02 di Struts, anche lo struts-config_1_1.dtd nel caso della 1.1).

webapps/*.war –

Questi files WAR sono delle applicazioni di esempio oppure delle mini-guide sui taglibs etc.

webapps/struts-blank.war –

si tratta di una web-application Struts base, sulla quale si può cominciare a lavorare per far pratica con il framework, che personalmente ho trovat omolto utile.

lib/commons-*.jar –

Questi files JAR contengono tutti i pacchetti del Jakarta Commons project che sono utilizzati all'interno del framework Tiles ora integrato con Struts; , se stiamo utilizzando come servlet container Tomcat, li troveremo già contenuti nella directory %TOMCAT_HOME%\common\lib.

lib/tiles.jar –

Contiene tutte le classi del framework Tiles (che possiede le seguenti caratteristiche : utilizzo dei templates, supporto per il multi-linguaggio, caricamento e costruzione dinamica delle pagine, impostazione del layout etc.).

lib/validator-rules.xml –

Questo file contiene delle regole di validazione di default da integrare nella propria applicazione.

Per poter utilizzare Struts, tutti i file jar vanno copiati all'interno della directory WEB-INF/lib della nostra applicazione e tutti i file tld nella directory WEB-INF oppure in un'altra sottodirectory creata da noi.

A questo punto siamo pronti per lavorare con questo framework, partendo ovviamento dal progetto della nostra web-application e dalla costruzione della sua spina dorse, ovvero il file struts-config.xml.

Come accennavamo, però, essendo Struts un framework molto diffuso e valido, sono stati sviluppati molti software che agevolano l'operatore nella programmazione di Struts. Questi, sia open source che commerciali, vanno dai semplici programmi per la scrittura assistita del file struts-config.xml (sicuramente la cosa più utile, essendo il punto più importante del framework), come ad esempio Struts-Console a complessi ambienti di sviluppo che assistono il programmatore dal progetto al testing della

web-application., oppure è anche possibile utilizzare più tools diversi ognuno per una fase dello sviluppo della web application.
Tra le proposte più accattivanti, possiamo segnalare:

- Borland Jbuilder: che offre versioni di Jbuilder Enterprise, Professional e Personal (quest ultima free) vd. www.borland.com/jbuilder

- IBM WebSphere Studio: costruito su tecnologia Eclipse (vd oltre), WebSphere Studio espande le funzionalità combinando varie risorse esistenti.

- IntelliJ IDEA

- NetBeans: rilasciato come open-source dalla Sun Microsystem, NetBeans è scaricabile dall'URL www.netbeans.org.

- Eclipse

5.4.2 Eclipse 3.0

Tra i tanti ambienti di sviluppo per Java, ho scelto per il mio progetto e consiglio, Eclipse 3.0. Si tratta di un IDE Java (Integrated Development Environment) con tanti aspetti innovativi, realizzato da IBM, ma con una licenza open source. L'ambiente si può scaricare gratuitamente dal sito ufficiale http://www.eclipse.org

Anche se arrivati alla versione 3.1, consiglio di utilizzare la 3.0 perché più robusta al nostro scopo.

Eclipse è un ambiente di sviluppo molto diffuso, perché oltre ad essere gratuito è leggero e raccoglie le migliori caratteristiche di tutti gli ambienti di sviluppo professionali, diventando un supporto molto valido che velocizza molto il lavoro del programmatore. La caratteristica principale di eclipse è quella di avere una architettura a plugin (motivo della sua leggerezza), il che lo rende un ambiente molto versatile e adatto alle varie esigenze. Difatti, eclipse non nasce per lo sviluppo di web application e per l'uso di struts è un ambiente del tutto generico, ma consente la possibilità di istallare plugin che permettono di personalizzarlo ed estenderlo notevolmente. Ad esempio, si possono istallare plugin per la progettazione UML, per la connessione a Tomcat, per l'uso di Struts, fino a plugin per la programmazione in C++. Un elenco dei plugin più interessanti viene riportato a questo indirizzo: http://www.crionics.com/products/opensource/eclipse/eclipse.html

Dunque, una volta istallato eclipse 3.0 sulla nostra macchina, dovremo ancora andare a scegliere i plug-in più adatti per poter implementare una web-application Struts.

Data la fama di Struts, si capisce subito, che la scelta non è per nulla limitata, anzi negli ultimi tempi sono stati prodotti tantissimi plug-in, da aziende diverse, open source o commerciali e più o meno validi.

Prima di effettuare una scelta, ho fatto varie prove, partendo da plug-in "minimalisti", ovvero che svolgono una sola funzione come: Struts-Console per la

scrittura del struts-config.xml, oppure il plug-in prodotto da Sysdeo per interfacciarsi con Tomcat dall'interno di Eclipse (www.sysdeo.com/eclipse/tomcatPlugin.html). Infine sono approdato a plug-in molto più complessi e completi, realizzati per assistere lo sviluppatore di web application in quasi tutte le fasi della realizzazioni e che si interfacciano perfettamente con Struts.
Tra questi, i più performanti sono sicuramente stati due:

- Exadel Studio scaricabile dall'indirizzo http://www.exadel.com/ e di cui la versione 2.5 è freeware;

- Myeclipse 4.0 di cui è scaricabile una versione demo di 30gg del seguente url http://www.myeclipseide.com.

Questi due ultimi plugin, come accennato coprono tutti gli aspetti della realizzazione della web-application (tranne la progettazione UML) e scaricando questi, non sarà più necessario scaricare il container Tomcat, ne il framewor Struts, in quanto al loro interno già contengono l'ultima versione le ultime versioni.
Personalmente ho trovato, per il nostro scopo, i due plug-in equivalente, ma utilizzando fino ad ora solo applicativi open-source o freeware, mi è sembrato logico optare per Exadel

5.4.3 Exadel Studio 2.5

Exdal Studio è un potente tool free per sviluppo di web-application che estende le funzionalità di Eclipse, permettendo agli sviluppatori di utilizzare a pieno le tecniche RAD (Rapid Application Development).
Questa soluzione redditizia e semplificata include tra le caratteristiche fondamentali importanti:

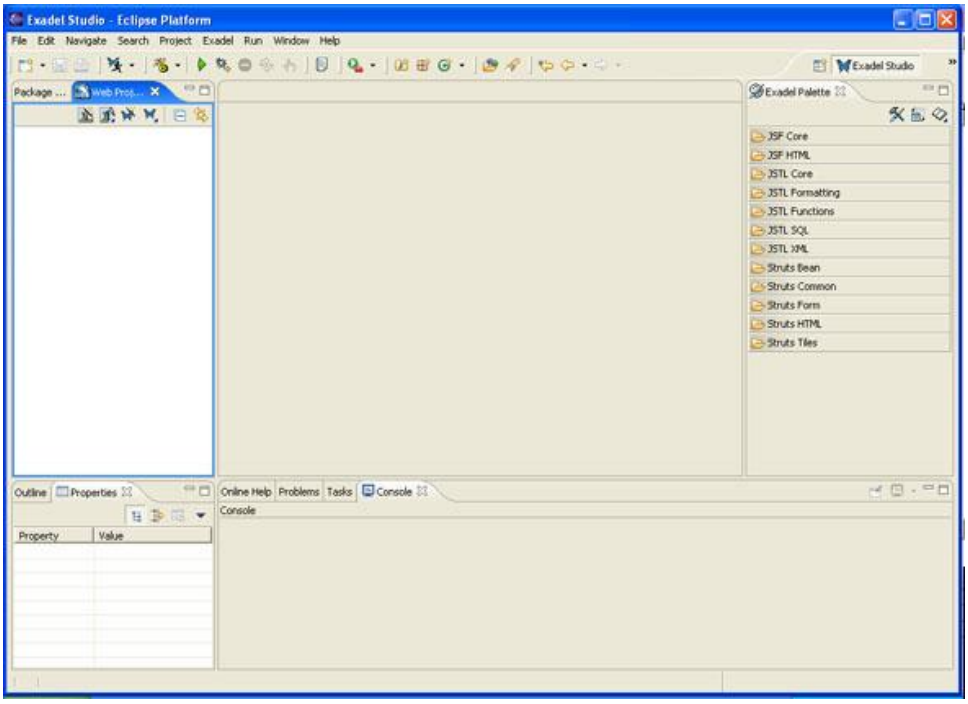
- supporto per lo sviluppo di progetti con tecnologia JSF, Struts, Spring e Hibernate;
- scrittura del codice assistita per pagine XML e JSP;

- supporto per il design e l'anteprima di pagine JSP
- flessibilità e personalizzabilità dei progetti.

Prima di andare avanti, per testare il corretto funzionamento del nostro ambiente di lavoro, possiamo utilizzare la web-application "bankwar" presente come esempio nel pacchetto di Struts che abbiamo scaricato.

Per fare questo, eseguiamo i seguenti passi:

1. Estraiamo il file struts-blank.war presente nella cartella webapps. Si tratta di una piccola web-application di esempio, che in realtà dovrebbe essere posta nella cartella di lavoro del nostro servlet container il quale dovrebbe effettuare il deploy automaticamente (ma per questa volta facciamo un eccezione).
2. Lanciamo Eclipse con Exadel istallato. Una volta aperto il programma, Eclipse ci permette di avere varie prospettive, noi scegliamo quella chiamata Exadel Studio dal menu Window-->Open Prospective--> Other...Dovremmo avere la seguente vista:



3. Dal menu file scegliamo import Struts Project. Ci verrà chiesto dove si trova il file web.xml dell'applicazione che vogliamo importare. Nel nostro caso si trova nella cartella WEB-INF al percorso dove abbiamo estratto il file .war.
4. Importata la web-application facciamo partire Tomcat, basta cliccare sul pulsante  nella barra superiore.
5. Infine lanciamo la nostra web-application cliccando su 

Come risultato dovremmo visualizzare la pagina di benvenuto dell'applicazione.

5.4.4 EcipseUML

Se tutto funziona correttamente, abbiamo un ambiente pronto per implementare e testare web-application Struts. Quello che ancora possiamo aggiungere per facilitare il lavoro di sviluppatore è un buon strumento di progettazione evoluto. Anche qui, come è facile immaginare abbiamo una vasta scelta di plug-in dal classico open-source Argo2UML a quello che sta diventando il plug-in per la progettazione UML principale di Eclipse: EclipseUML Free Edition o Studio, scaricabile dal sito www.omondo.com

Per il progetto della mia web-application dimostrativa ho avuto possibilità di provarlo ed è risultato uno strumento molto potente. Prodotto da sviluppatori Java per sviluppatori Java è un tool di progettazione visuale che con molta flessibilità copre gli aspetti della progettazione software J2EE e non.

Lo sviluppo del software è un processo iterativo ed i membri della squadra devono lavorare con le stesse informazioni correnti. Di conseguenza EclipseUML permette che gli sviluppatori abbiano un metodo guidato di modello di sviluppo usando la sincronizzazione bidirezionale del modello e di codice.

In pratica anche se non necessario per piccole applicazioni ho constatato che man mano che avanza la progettazione uml, in parallelo viene generato del codice sorgente "buono" ovvero, che non richiede di essere eccessivamente ritoccato.

Altre caratteristiche importanti di questo plug-in sono le possibilità che offre per il reverse engineering e per la generazione automatica della documentazione, funzionalità che però non ho testato in questa sede.

5.4.5 MySQL

Ultimo componente software di cui abbiamo bisogno è certamente un DBMS (database management system) per garantire la persistenza dei dati della nostra web-application. Con il termine persistenza ci si riferisce al fatto che i dati inseriti in un'applicazione, sia da un utente che da altri mezzi, avranno una esistenza che va oltre il ciclo di vita dell'applicazione stessa. L'informazione sopravviverà agli arresti dell'applicazione o allo spegnimento della macchina. È chiaro come questo sia fondamentale per una applicazione web, che nasce proprio per gestire dei dati, che devono essere persistenti, indipendentemente dalla complessità dell'applicazione. Senza persistenza dei dati non esisterebbe alcuna registrazione di quello che, ad esempio, potrebbe essere stato ordinato, oppure di chi deve pagare.
Anche se esistono tipi diversi di immagazzinamento dei dati, i database relazionali sono quelli utilizzati più di frequente per conservare i dati di un'applicazione e diventano quasi un obbligo in applicazioni web di e-commerce come quella che vogliamo realizzare.
Al solito, esistono molti produttori che mettono a disposizione DBMS open-source o meno. Tra i più popolari troviamo:

- Microsoft SQL Server
- MySQL
- Oracle
- Sybase

Per il mio progetto ho scelto di utilizzare MySQL, magari non è il più performante, ma è un progetto open-source che implementa un DBMS free per uso non commerciale sufficiente ai nostri scopi.

Per garantire le capacità di persistenza alla nostra applicazione Struts abbiamo dunque bisogno di:

- DBMS MySql scaricabile dal sito www.mysql.com
- Un DB Driver: ovvero un driver per consentire all'applicazione di connettersi al database. In pratica si tratta di una classe Java che implementa a basso livello strumenti per stabilire e mantenere una connessione ad uno specifico DBMS. MySQL prevede un driver JDBC (Java Database Connectivity) chiamato MySQL Connector/J anche questo scaricabile dal sito ufficiale di MySQL.

Scaricato il materiale occorrente, dovremo istallare il DBMS sulla nostra macchina, come una normale applicazione ed importare il driver JDBC che non è altro che un file .jar in eclipse all'interno del nostro progetto (vedremo più avanti come).

5.4.6 Pool di connessioni

Un ultimo componente per garantire la persistenza, anche se non fondamentale, potrebbe essere un driver che implementa un pool di connessioni.
In pratica, nelle web-application, specie se di grosse dimensioni, uno dei principali problemi è che ci sono molte richieste di accesso ad una base di dati e dunque aprire una connessione verso il DB per ogni richiesta non è di certo una soluzione performante, in quanto l'operazione di connessione al DB è molto onerosa e in una normale web-application potrebbero essere nell'ordine delle centinaia.
Una soluzione è quella di creare un pool di connessioni verso uno specifico DB, ovvero un insieme di connessione attive il cui numero massimo viene impostato in fase di configurazione. Ogni volta che sarà richiesta una operazione sul DB non sarà più necessario aprire una nuova connessione e poi richiuderla, ma sarà sufficiente chiedere che venga allocata una di quelle già disponibili e poi, cosa importante, rilasciarla quando l'operazione sul DB è terminata. Ovviamente, se al momento della richiesta non ci sono risorse disponibili, ci sarà un'attesa o comunque una situazione che potrà essere gestita dal sistema.

Nel JDBC 2.0 è presente solo la definizione dell'interfaccia per gestire una connection pooling, un implementazione, tra le più usate è sicuramente il progetto Jakarta Common DBCP, di cui possiamo trovare tutta la documentazione all'indirizzo: jakarta.apache.org/commons/dbcp

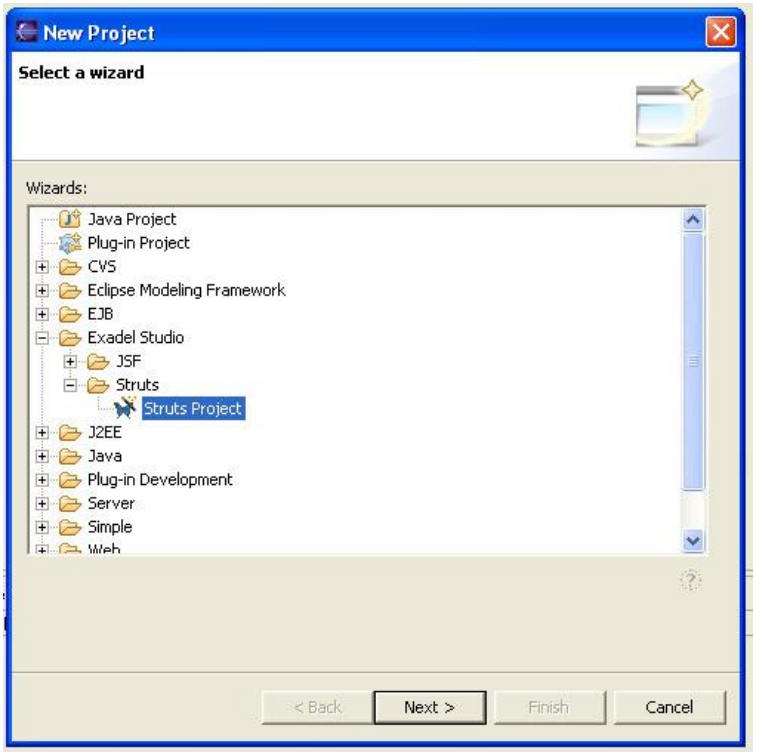
Al solito, per utilizzare nella nostra applicazione, questo progetto di Apache, bisogna scaricare i file .jar: *commons-dbcp-1.1.jar* e *commons-pool-1.1.jar* copiarli nella directory /lib della nostra applicazione e caricarli all'interno di Eclipse nel nostro progetto, come librerie aggiuntive.

5.5 VIDEOSTORE

5.5.1 Descrizione dell'applicazione

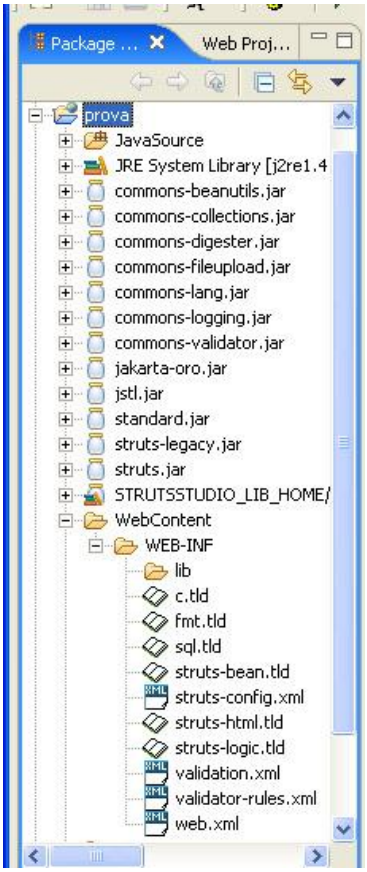
Si vuole realizzare un negozio on-line di un ipotetico video store, implementando le funzionalità classiche di questo tipo di applicazione.
In particolar modo si vuole che attraverso il proprio browser possano accedere al sistema utenti con diritti di amministratori, i quali possano svolgere le classiche funzioni di: inserimento film, eliminazione film, modifica, ricerca, gestione ordini etc e utenti privati che on-line possano consultare il data base contenente film e se registrati acquistare gli stessi e consultare il proprio carrello.

Per cominciare a lavorare, lanciamo Eclipse con vista Exadel e creiamo un nuovo progetto Struts.
Dal menù: File --> New --> Project scegliamo Struts Project



nelle finestre successive, assegniamo un nome al progetto, scegliamo la versione di Struts con cui lavorare (possiamo scegliere tra 1.1 e 1.2), la versione per le servlet ed infine le librerie di custom tag e le jstl da importare nel nostro progetto. Ovviamente, sono tutte scelte che possono essere modificate nel corso dello sviluppo dell'applicazione.

Generando il progetto, nella cartella di lavoro di eclipse, verrà creata la struttura base di una web-application e quello che visualizzeremo in eclipse sarà questo:



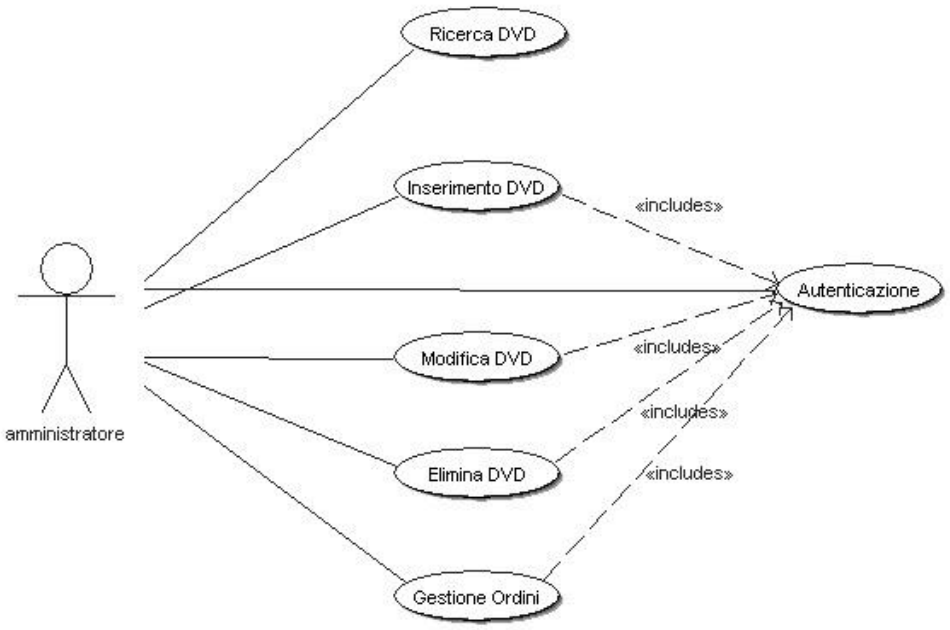
come si vede la struttura è simile a quella di un'applicazione web come descritta prima. Nella struttura ci vengono visualizzate tutte le librerie già caricate ed inoltre, come si vede, già vengono generati i file xml (es: lo struts-config.xml) che dovremo solo modificare

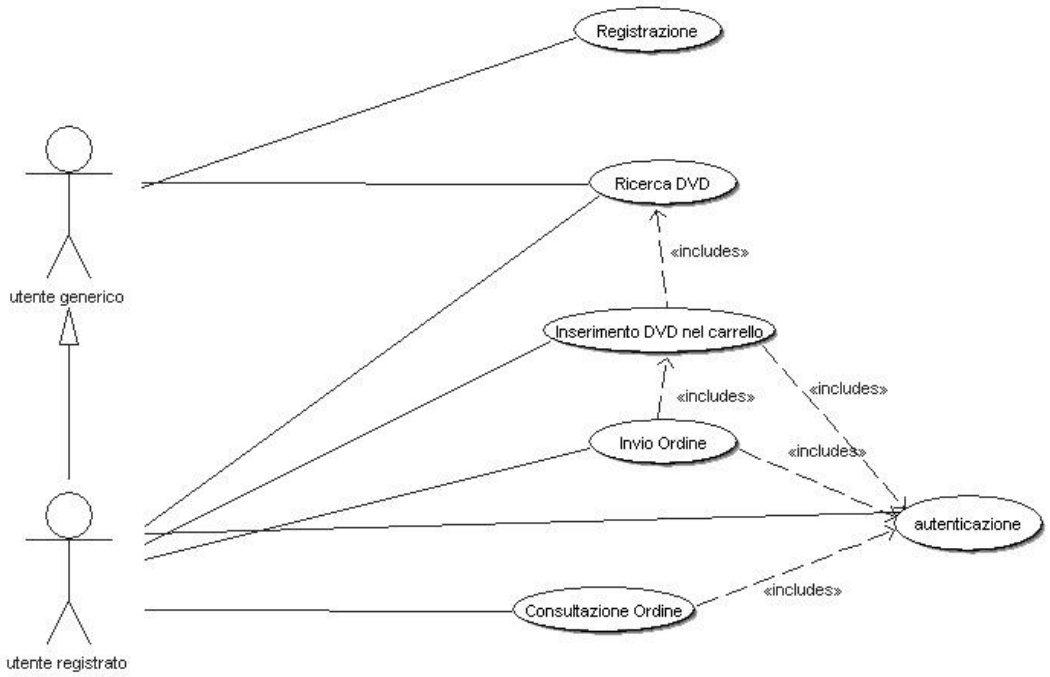
Nella cartella JavaSource andremo a salvare i file sorgenti che scriveremo, eventualmente creando dei package.

Altra operazione preliminare, prima di cominciare ad implementare è l'introduzione delle librerie non di base, come abbiamo accennato prima.

In particolare poiché per il nostro progetto abbiamo deciso di usare un DBMS MySQL unitamente ad un pool di connessioni, dobbiamo scaricare i file .jar: *commons-dbcp-1.1.jar* e *commons-pool-1.1.jar* dal sito ufficiale del progetto Apache DBCP e importarli nel nostro progetto. Allo stesso modo si procede per qualsiasi altro componente che non faccia parte di un progetto base.

Banalmente, dalle specifiche di progetto, secondo le regole della progettazione Uml, individuiamo tre attori: l'utente generico, l'utente registrato, l'amministratore ed una serie di funzionalità, come risulta dai diagrammi dei casi d'uso realizzati con EclipseUML:





La prima cosa da fare è passare alla progettazione del model della nostra applicazione.

5.5.2 Business Object

Ricordiamo che il model è un insieme di *data business* e di metodi a loro collegati, che effettuano le elaborazioni e possono essere invocati da altri livelli come il controller o anche il view. Dunque il primo scopo e quello di progettare questo model indipendentemente dagli altri livelli, in teoria dovrebbe essere un progetto a se, magari realizzato da qualcun altro e di cui il controller e il view conoscono solo le firme dei metodi senza ovviamente sapere come sono implementati. Inoltre come abbiamo detto, Struts è un framework che implementa essenzialmente il controller del pattern MVC lasciando molta libertà nell'implementazione del model dove si può far uso di varie tecnologie Java.

Un procedimento corretto può essere quello di andare ad individuare i *business objects* della nostra applicazione.

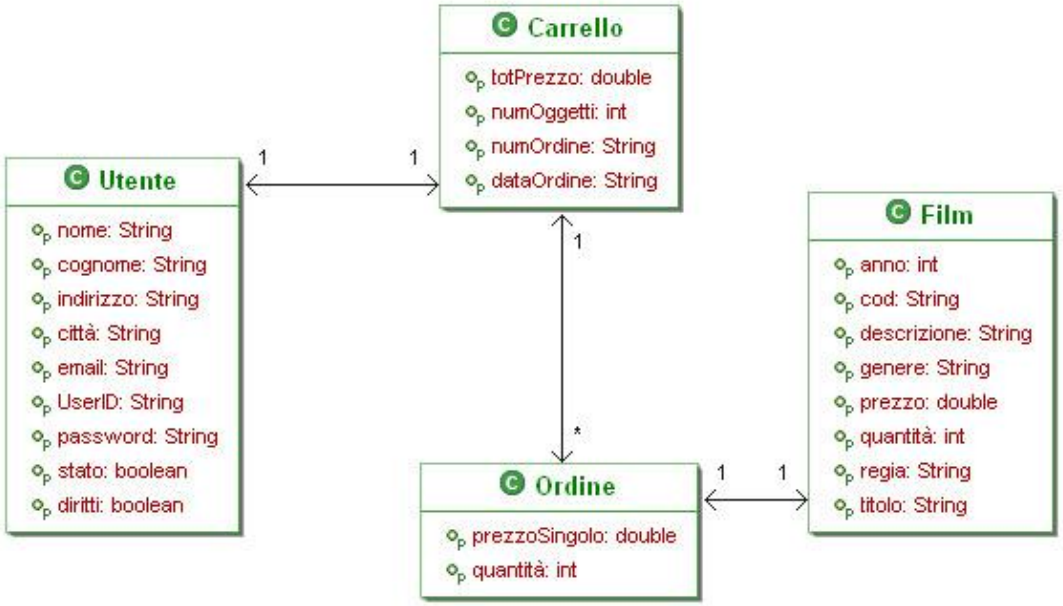
Un *business objects* è semplicemente un'astrazione software di entità appartenente alla realtà (una sorta di classi). Rappresenta una persona, una cosa o un concetto appartenente al dominio dell'elaborazione. In riferimento alle specifiche del nostro progetto, subito ci vengono in mente concetti come: film, utente, carrello, acquisto che potrebbero essere i BO della nostra applicazione.
I business objects vengono ovviamente implementati attraverso delle classi, ma una classe per essere considerata un business object, deve sottostare a diverse condizioni:

- possedere stato e comportamento;
- rappresentare una persona, un luogo, un oggetto o un concetto appartenente al dominio della nostra applicazione;
- essere riutilizzabile.

Le tecniche di progettazione che portano all'individuazioni dei BO, possono essere diverse e complesse e non è quello di cui ci vogliamo occupare. Anche perché per la nostra semplice applicazione dimostrativa è facile individuare questi oggetti in:

- film:
- utente: caratterizzato dai propri dati anagrafici e dati relativi all'account;
- carrello: associato ad un singolo userID e caratterizzato da una lista di acquisti;
- ordine: inteso come l'acquisto del singolo oggetto la collezione del quale va a formare il carrello.

Un diagramma delle classi per i business object dell'applicazione videoStore potrebbe essere la seguente:



I business object possono essere implementati attraverso dei normali oggetti JavaBeans.

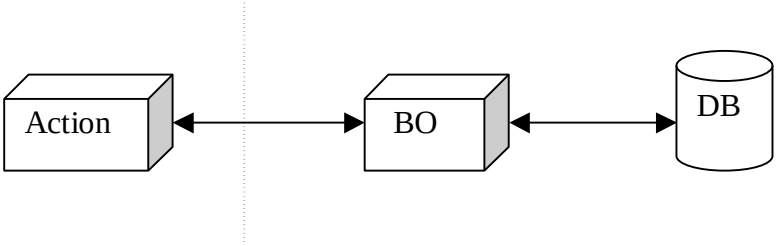
Ad esempio l'oggetto Utente potrebbe essere implementato in questo modo:

```
package com.videoStore.BOobject;

import java.util.Collection;
import java.util.Iterator;

public class Utente {

	private String nome;
	private String cognome;
	private String indirizzo;
	private String città;
	private String email;
	private String userID;
	private String password;
	private boolean admin;


	public String getNome() {
		return nome;
	}
	public void setNome(String nome) {
		this.nome = nome;
	}
```

```
public String getCognome() {
	return cognome;
}
public void setCognome(String cognome) {
	this.cognome = cognome;
}

public String getIndirizzo() {
	return indirizzo;
}
public void setIndirizzo(String indirizzo) {
	this.indirizzo = indirizzo;
}

public String getCittà() {
	return città;
}
public void setCittà(String città) {
	this.città = città;
}

public String getEmail() {
	return email;
}
public void setEmail(String email) {
	this.email = email;
}

public String getUserID() {
	return userID;
}
public void setUserID(String userID) {
	this.userID = userID;
}

public String getPassword() {
	return password;
}
public void setPassword(String password) {
	this.password = password;
}

public boolean isAdmin() {
	return admin;
}
public void setAdmin(boolean stato) {
	this.admin = stato;
}


public carrello carrello;
```

```
public carrello getCarrello() {
	return carrello;
}

public void setCarrello(carrello carrello) {
	this.carrello = carrello;
}
```

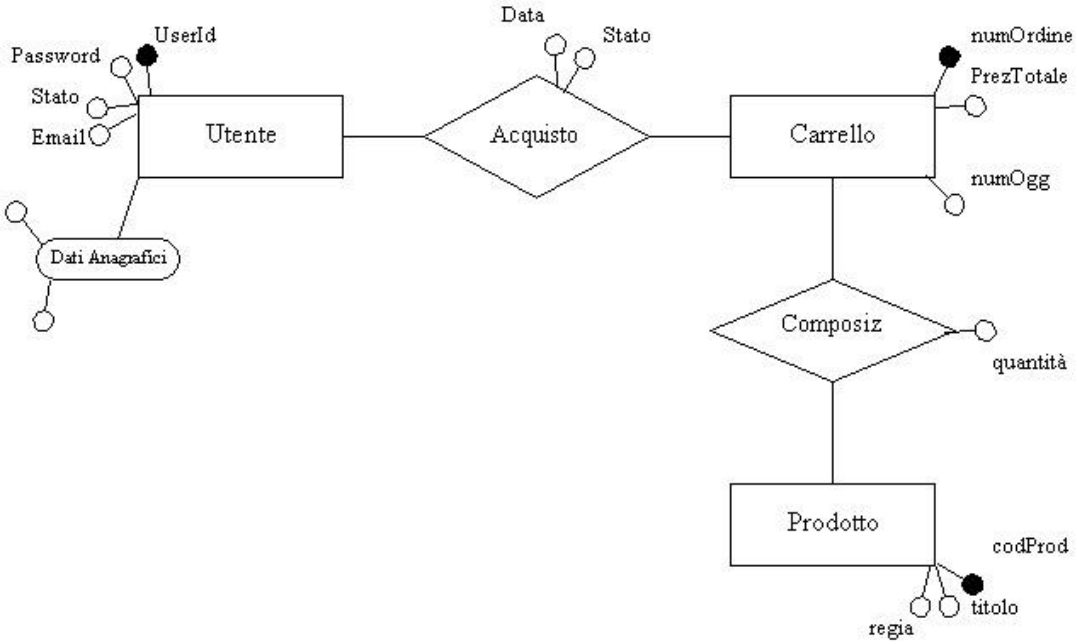
In realtà, come abbiamo già detto, se utilizziamo EclipseUML di Omondo per realizzare il diagramma delle classi dei business object, questi verranno implementati automaticamente dal plug-in, attraverso dei JavaBeans come quello visto, conservando le associazioni impostate e riducendo il compito del programmatore.
Questi business object che abbiamo creati, potrebbero essere visti come dei semplici contenitori di dati, detti a volte DTO (Data Transfer Object) perché hanno il compito di trasferire dati tra uno strato e l'altro della nostra applicazione. Quindi verso l'alto, cioè verso il controller o il view, oppure verso il basso, ovvero verso il DB.



Creati tutti i business object per l'applicazione videoStore bisogna creare un modello di database e uno schema. Infatti quando un'applicazione è in esecuzione, ogni istanza dei bo, possono contenere dati che in quel momento risiedono in memoria, ma come abbiamo già detto non posso restare lì per sempre e a seconda dei casi, devono essere cancellati o conservati per mezzo di una memorizzazione in una base di dati.
Inoltre in applicazione complesse, potrebbe non esserci una corrispondenza netta tra la struttura dei business object e lo schema del DB e dunque il passaggio diviene più delicato.

5.5.3 Struttura del DB

Il modello entità relazione per la nostra applicazione, potrebbe essere il seguente:

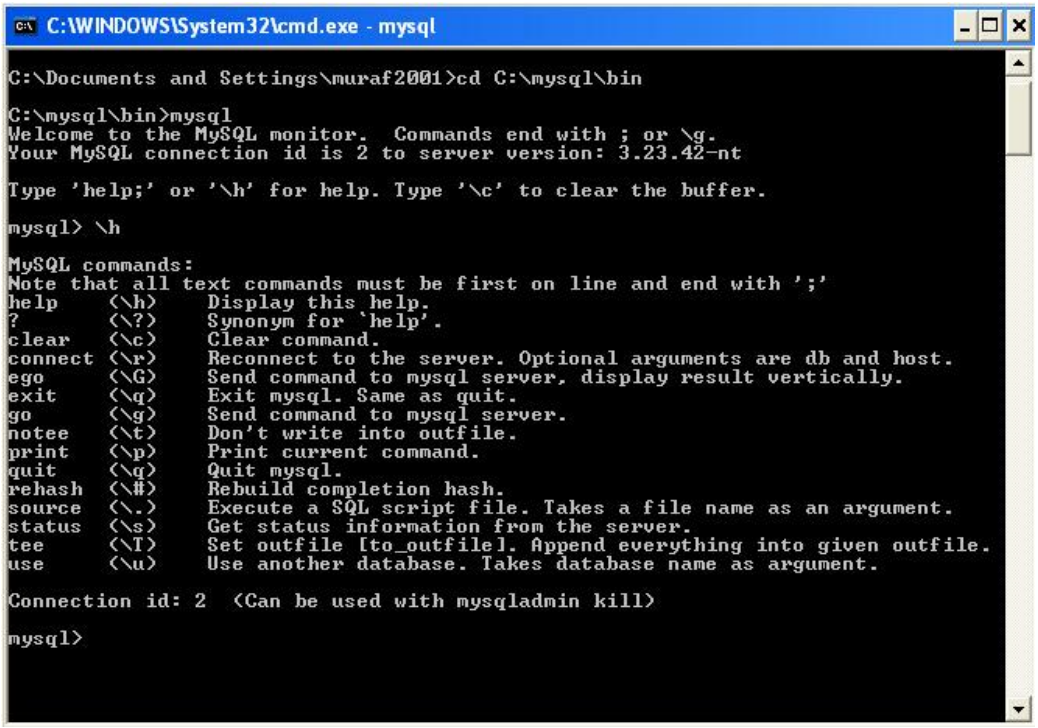


(per chiarezza non sono stati indicati tutti gli attributi)

da questo modello, ricaviamo il seguente schema:

Utente (UserId, Password, email, nome, cognome,città, indirizzo, cap, admin)
Carrello (userid, numOrdine, data, prezTotale, numOgg);
Composizione (numOrdine, codProd , quantità);
Prodotto (codProd, titolo,genere,regia,descrizione,prezzo,quantità).

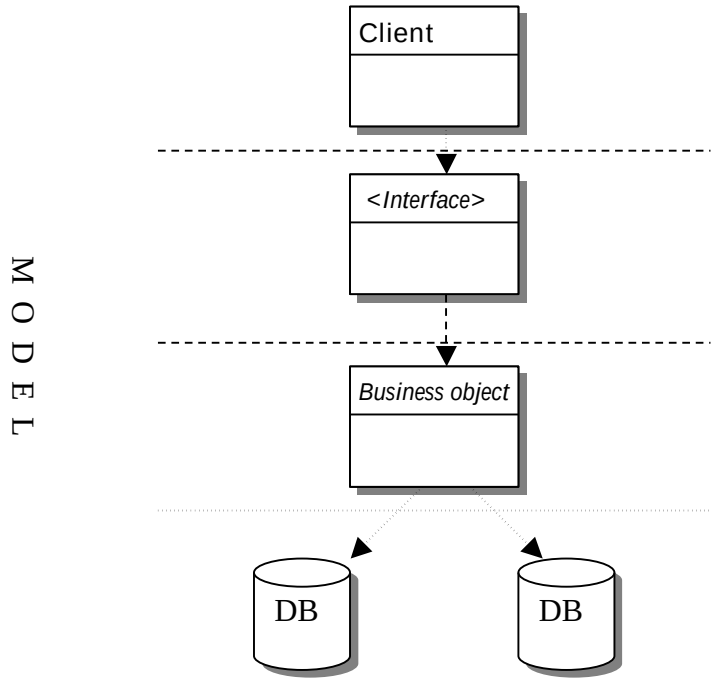
Per creare lo schema appena visto, dal prompt di Dos spostiamoci nella cartella /bin di mysql e lanciamo l'applicazione. Digitando '\h' verranno visualizzati una serie di comandi disponibili.



A questo punto, basta costruire il nostro schema direttamente a riga di comando, utilizzando la normale sintassi SQL oppure preparando un file script da lanciare come questo:

```
Nome file: videostore.sql

DROP DATABASE VIDEOSTORE;

CREATE TABLE UTENTI (
       USERID VARCHAR(30) PRIMARY KAY,
       PASSWORD VARCHAR(30) NOT NULL,
       NOME VARCHAR(30) NOT NULL,
       COGNOME VARCHAR(50) NOT NULL,
       INDIRIZZO VARCHAR(50) NOT NULL,
       CITTA VARCHAR(50) NOT NULL,
       EMAIL VARCHAR(30) NOT NULL,
       ADMIN BOOL
)

CREATE TABLE FILM...
....
..
```

L’ultima fase della realizzazione del model è costruire delle interfacce di servizio che forniscano dei metodi che agiscono sulla struttura dati appena realizza. In tal modo, queste interfacce, utilizzabili dai clients, che nel nostro caso saranno le action di Struts, permetto un maggior disaccoppiamento tra i vari livelli. Difatti le action avranno a che fare solo con le interfacce di cui richiameranno i metodi, eventualmente senza neanche conoscere cosa c’è sotto. Tutto ciò, va proprio in direzione della riusabilità e separazione dei compiti di cui si è parlato nella parte teorica. All’interno di un team di lavoro reale, non sarebbe strano individuare un programmatore Struts, che avrà il compito di configurare la parte del controller e scrivere le action conoscendo solo dei metodi che gli sono messi a disposizione da un livello model realizzato da qualcun altro.



Per la nostra applicazione, ho diviso le diverse funzionalità in interfacce distinti, in base al loro campo di azione sui business object.
Ad esempio l’intefaccia Idvd sarà fatta in questo modo:

```
package com.videoStore.servizi;

import java.util.Collection;

public interface IDvd {
```

```
	public Collection ricFilm(String campo, String
valore)throws Exception;

	public void inserFilm(String titolo,String regia,int
anno,String descrizione,String genere,int quantità, double
prezzo)throws Exception;

	public void elimFilm(String cod)throws Exception;

	public void editFilm(String campo, String valore)throws
Exception;

	public boolean isDisp(String cod)throws Exception;

}
```

Ovviamente, tutti questi metodi vanno poi implementati ed è qui che si inserisce la logica di business che rappresenta il valore aggiunto della nostra applicazione Struts.

Non essendo un manuale di programmazione Java sarebbe superfluo riportare le implementazioni che ho effettuato, non è questo quello che ci interessa, quanto piuttosto capire come va progettato il livello model e passare alle altre fasi della programmazione, ovvero al controller e alla view.

Per chiarezza, riportiamo solo l'implementazione del metodo verificaUtente( ) fornito dall'interfacca IUtente:

```
ConnMngImpl cn = new ConnMngImpl();
public Utente verifUtente(String userid,
                    String password) throws Exception {
try{
	stmt = con.createStatement();
	sQuery = "SELECT * FROM utenti " + "WHERE userid = '" +
     userid + "' " + "AND password = '" + password + "'";
	rs = stmt.executeQuery(sQuery);
	if (rs.next())
	{
		// Crea e popola un nuovo oggetto utente
		// da restituire
		user = new Utente();
		user.setUserID(rs.getString("userid"));
		user.setPassword(rs.getString("password"));
		user.setAdmin(rs.getBoolean("admin"));
		user.setCognome(rs.getString("cognome"));
		user.setNome(rs.getString("nome"));
		user.setIndirizzo(rs.getString("indirizzo"));
```

```
            user.setEmail(rs.getString("email"));
            user.setCarrello(new carrello());
      }
      catch (SQLException se){
            se.printStackTrace();
            ModuleException me =
                  new
ModuleException("verif.select.error");
            throw me;
      }
      finally
      {
            cn.close();
      }
            return user;
      }
```

Come si vede si tratta di un semplice funzione che riceve in ingresso due stringhe ed effettua una ricerca in un DB, restituendo un oggetto utente, perfettamente neutrale, nel senso che non è legata a nessun framework specifico e che “non sa” cosa c’è al di sopra o chi la invoca.

Unica osservazione da fare, riguarda la connessione al db. Come accennato ho utilizzato un pool di connessioni. Le funzioni di richiesta e rilascio della risorse, poiché comuni a quasi tutte i metodi implementati, sono state realizzate in una classe esterna ConnMngImpl.java che fornisce i metodi

```
public Connection open(DataSource dataSource) throws
Exception;
public void close() throws Exception;
```

5.5.4 Realizzazione della Web-Application Struts: View e Controller

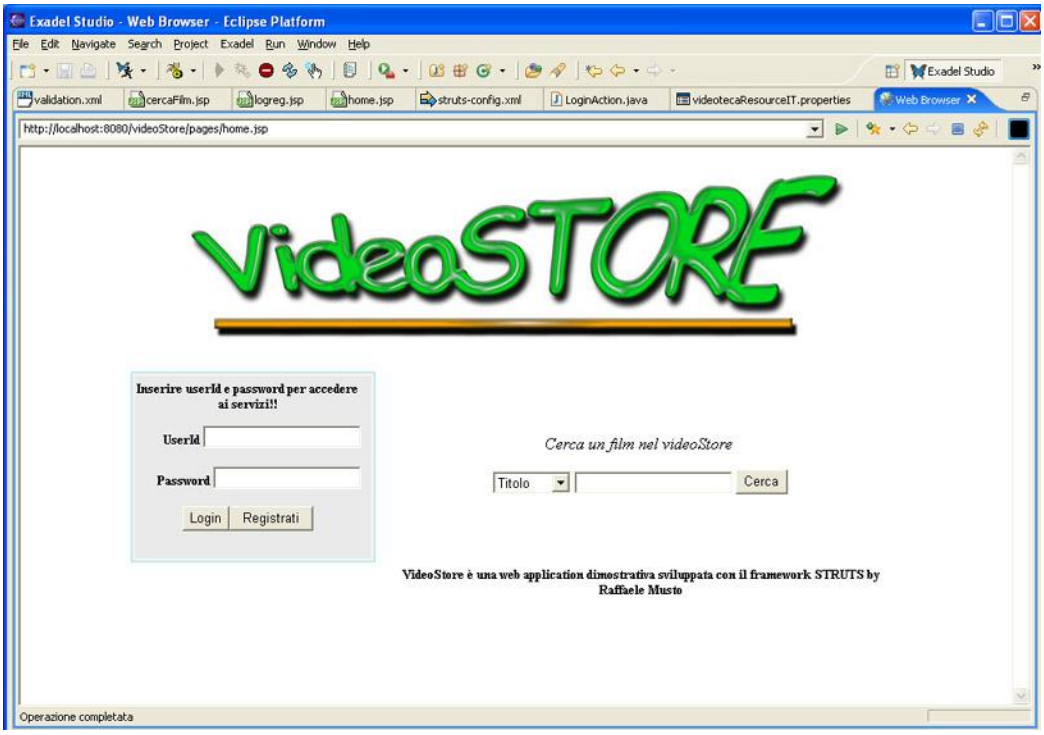
Implementata la struttura di supporto alla nostre applicazione web, ovvero il model con tutta la logica di business, il passo successivo consiste nella progettazione e realizzazione del view e del controller, dunque la vera e propria programmazione di Struts. In un progetto di grosse dimensioni e professionale, il procedimento dovrebbe essere quello di realizzare il controller, ovvero implementare tutte le action, dunque i servizi offerti agli utenti del sito e successivamente realizzare il view (al solito potrebbe essere un altro team a farlo) e configurare l'applicazione attraverso lo struts-config.xml

Poiché la mia è una semplice applicazione dimostrativa, risulta più semplice progettare il sito web, definendo una sorta di diagramma di sequenza tra le pagine JSP ed in parallelo programmare le action.

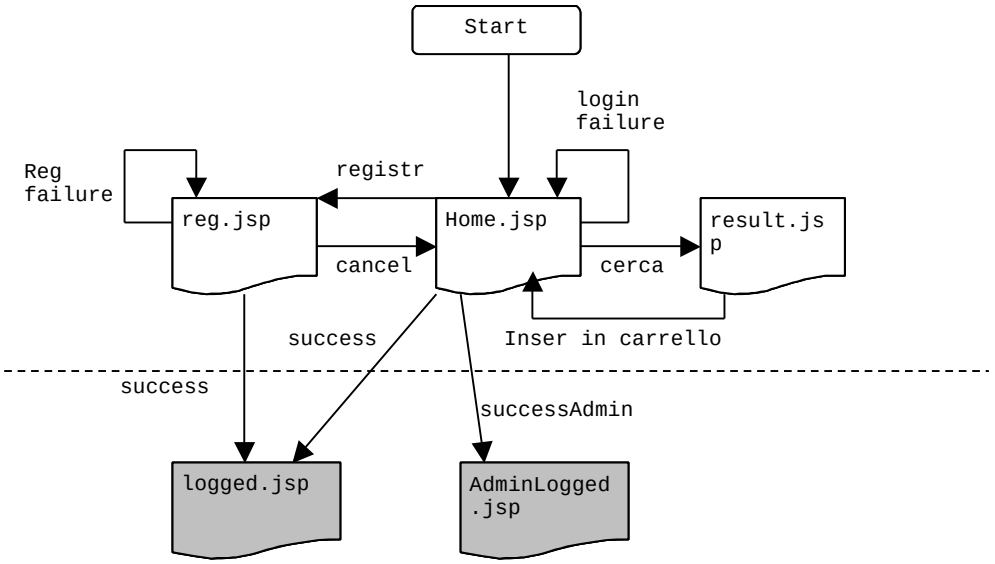
Mi sembrerebbe inutile riportare lo sviluppo di tutta l'applicazione passo passo, dato la ripetitività delle operazioni. Dunque di seguito mi limiterò a mostrare in dettaglio solo la realizzazione di un action e limitarmi a descrivere, dove necessario, le soluzioni implementative adottate.

Dalla lettura delle specifiche e dall'analisi dei casi d'uso, supponiamo che l'utente che si colleghi alla home page del sito web del VideoStore, abbi da subito, la possibilità di effettuare la ricerca di un film all'interno del database anche se non autenticato, prevedendo che l'autenticazione sarà necessaria all'atto del inserimento del film nel carrello. In alternativa, sempre dal home page, viene data la possibilità di effettuare il login e di registrarsi. In caso di insuccesso di queste due operazioni, verrà segnalato un errore all'utente e le azioni verranno riproposte, in caso di successo, l'utente verrà inviato alla pagina personale.

Quando l'utente accede al videoStore collegandosi ad uno specifico url, gli si presenterà un home page di questo tipo:



Una sorta di diagramma di sequenza di questa prima parte dell'applicazione, potrebbe essere la seguente:



Come primo passo, andiamo a realizzare un form da associare all'operazione di login, che chiameremo *loginForm*. Solitamente, un form del genere sarà costituito da due campi di input in cui l'utente dovrà inserire userID e password. Come visto dalla teoria, possiamo scegliere tra l'implementazione di una classe LoginForm che

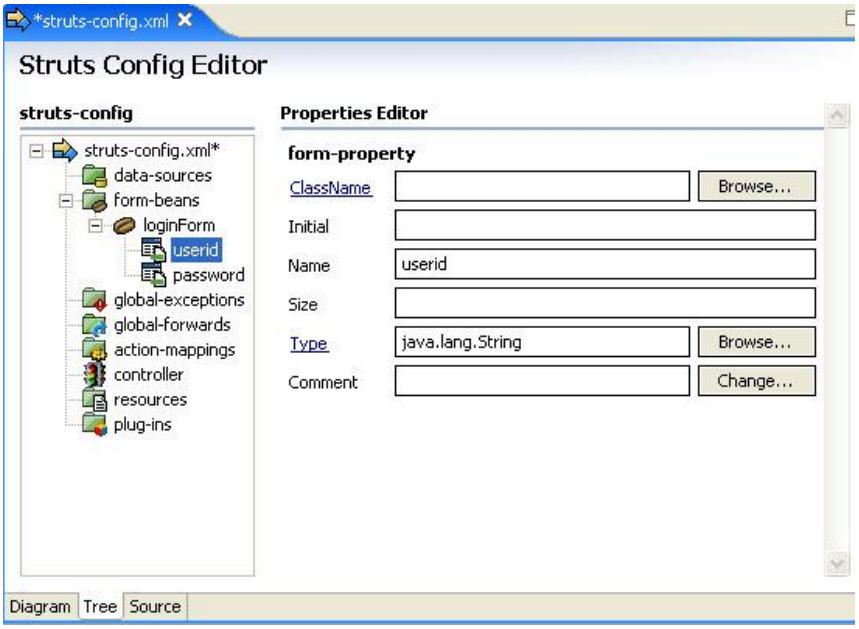
estenda la ActionForm di Struts implementando i metodi reset() e validate(), oppure realizzare un form dinamico con validazione dei dati attraverso il framework validator. Per i vantaggi, già citati, legati alla seconda tecnica, ho deciso di adottarla in tutta l'applicazione.

Sempre dalla prospettiva Exadel di Eclipse, apriamo il file struts-config.xml.

Exadel offre tre viste differenti per questo file:

- *source*: in cui visualizza il codice in metadati;
- *tree*: in cui visualizza il file attraverso una struttura ad albero. Sicuramente delle tre è quella che ho trovato più utile, in quanto facilita enormemente la costruzione del file, fornendo percorsi guidati per l'inserimento di tag e attributi;
- *diagram*: in cui visualizza un diagramma di sequenza per le action, utile per avere un visione di insieme.

Dalla vista tree, sarà sufficiente cliccare su form-beans e aggiungere un nuovo form con nome "loginForm" e type "org.apache.struts.validator.DynaValidatorForm" (ricordiamo che type indica la classe che deve essere estesa). Creato il formbean aggiungiamo a questo dei form-property uno per ogni campo del nostro form, specificando il nome, il tipo ed eventualmente un valore iniziale. Nel caso del loginForm, dovremo specificare due campi di tipo stringa senza valore iniziale. Il risultato dovrà essere quello in figura:



passando alla vista source, il codice generato dovrebbe essere il seguente:

```
<form-bean name="loginForm"
       type="org.apache.struts.validator.DynaValidatorForm">
   <form-property name="userid"
              type="java.lang.String"/>
   <form-property name="password"
              type="java.lang.String"/>
</form-bean>
```

come vedremo, assieme a questui due campi di input, nel form avremo anche due pulsanti, uno di submit e uno cancel.

Realizzato il form, dovremo inserire le dichiarazioni di validazione per i due campi.

Apriamo il file validation.xml, già presente nel nostro progetto e contenente del codice di esempio ed inseriamo all'interno del tag <formset> il seguente codice:

```
<form name="loginForm">
       <field
              property="userid"
              depends="required">
              <arg0 key="login.userid"/>
       </field>
       <field
              property="password"
              depends="required">
              <arg0 key="login.password"/>
       </field>
</form>
```

Il significato è alquanto semplice: ai due campi dalla form loginForm andiamo ad assegnare una regola standard "required" che fa parte di quelle base offerte dal framework validator e specificate nel validation-rules.xml. Inoltre arg0 è un paramtro che contiene una chiave il cui valore associato verrà visualizzato in caso di errore, ovvero quando la regola non verrà rispettata. Il mapping tra le chiavi e i rispettivi valori, avviene nel file *videotecaResourceIT.properties* .

Questo file, deve essere creato da noi, e solitamente viene posizionato nella directory Javasource. Salvando il file come .properties, verrà riconosciuto da Exadel, che come per tutti i files contenenti metadati, ci supporterà nella sua costruzione.

Per ora, andremo solamente ad inserire le due chiavi appena viste:

```
login.userid = userid
login.password = passwrod
```

come specificato quando abbiamo parlato dello struts-config.xml, il file properties va specificato nel tag message-resource

```
<message-resources parameter="videotecaResourceIT"/>
```

Ricordiamo che per utilizzare il framework validator, questo deve essere caricato come plugin dall'applicazione, dunque appositamente dichiarato nello strtus-config come specificato quando si è parlato di questo framework.

Fatto questo, passiamo alla realizzazione dell'action che chiameremo LoginAction.
Il compito di questa Action, sarà semplicemente quello di verificare se è stato premuto il tasto "cancella" e in caso contrario, prelevare i dati immessi nel form loginForm e richiamare la funzione del model verificaUtente( ), nella quale c'è la logica di business, per accertarsi che si tratti di un utente registrato. Ulteriore compito dell'action è stabilire quale debba essere la successiva pagina in cui dirottare l'utente. Le direzioni possibili per il flusso applicativo, dopo l'esecuzione dell'Action, sono tre. L'utente verrà dirottato nella sua pagina personale in caso di successo del login, nell' home page in caso di insuccesso e nella pagina di registrazione nel caso sia stato premuto il pulsante "cancella".
Precisiamo che quest'ultima decisione, quella di camuffare il tasto cancella come richiesta di registrazione nuovo utente, risponde all'esigenza di evitare un proliferare inutile di Action. L'alternativa era creare un'altra action appositamente per gestire la richiesta di registrazione.

Il metodo execute implementato è il seguente:

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             HttpServletRequest request,
                             HttpServletResponse response)
                                    throws Exception {
       Boolean bCancel =
(Boolean)request.getAttribute(org.apache.struts.action.CANCEL)
       if(bCancel != null)
              if(bCancel.booleanValue())
```

```
                    return (mapping.findForward("join"));

        IAutenticazione lg =
        new ServUtenteImpl(getDataSource(request, "videoDB"));

        //viene chiamata la funzione del model
        Utente user =lg.verifUtente(
                (String)((DynaValidatorForm)form).get("userid"),

(String)((DynaValidatorForm)form).get("password"));

        //se l'utente è stato autenticato
        if(user !=null){
                request.getSession().setAttribute("user",user);
                if(user.isAdmin())
                        return
mapping.findForward("successAdmin");
                return mapping.findForward("success");
        }else {
                ActionErrors errors = new ActionErrors();
                ActionError error = new
                        ActionError("error.login.invalid");
                errors.add("login",error);
                saveErrors(request,errors);
                return mapping.findForward("failure");
        }
```

Ricordiamo che la classe LoginAction deve estendere la classe org.apache.struts.action.Action.

Come si può vedere, in rispetto di quanto detto fin ora, la action non contiene alcuna logica di business, la quale è inglobata tutta nel metodo verifUtente( ) che fa parte del livello model, ma svolge solo funzioni che fanno parte del livello controller.

Ultimo passo è mappare l'action nello struts-config.

Sempre nella vista ad albero, aggiungiamo un elemento *action* al tag action-mappings e mappiamo l'action, settando gli attributi visti quando abbiamo parlato dettagliatamente dello strtus-config. Il risultato dovrebbe essere il seguente:

```
<action input="/pages/home.jsp"
        name="loginForm"
        path="/login"
        scope="request"
        type="com.videoStore.struts.LoginAction"
```

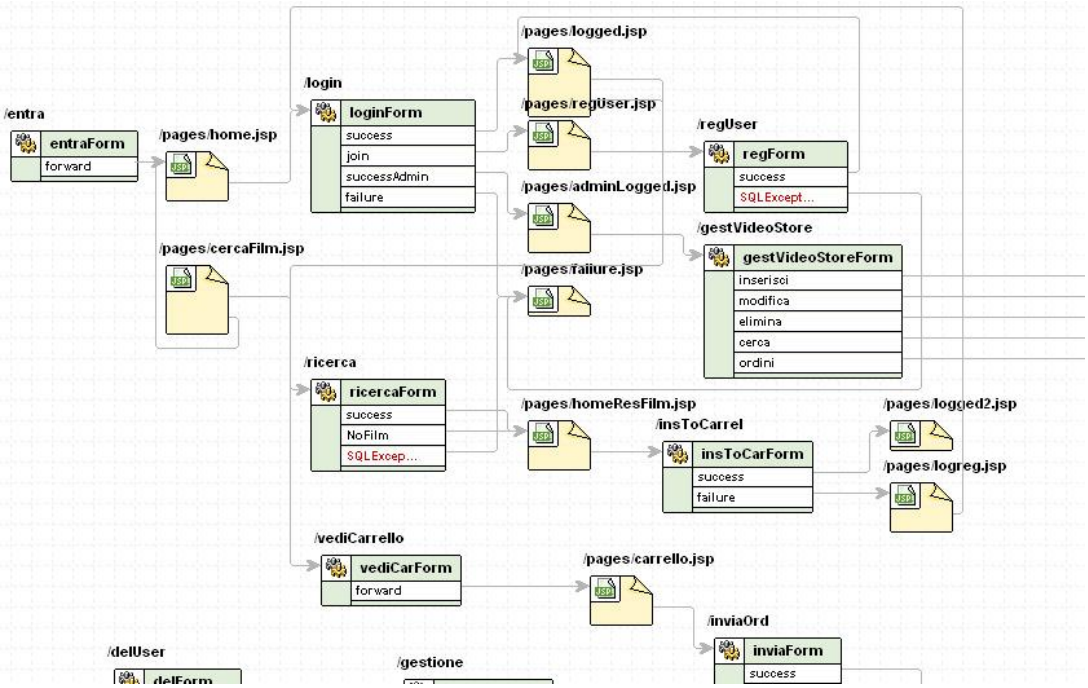
```
        validate="true">
   <forward name="success" path="/pages/logged.jsp"/>
   <forward name="successAdmin"
path="/pages/adminLogged.jsp"/>
   <forward name="failure" path="/pages/home.jsp"/>
   <forward name="join" path="/pages/regUser.jsp"/>
</action>
```

come si vede, abbiamo quattro elementi forward a seconda dell'esito dell'action.
Si noti che ne caso l'utente risulti registrato, l'action realizza un ulteriore controllo, andando a verificare se l'utente ha diritti di amministratore o meno redirigendo il flusso di conseguenza.
Per quanto riguarda le caratteristiche di sicurezza della nostra applicazione, essendo di piccole dimensioni abbiamo deciso di agire sull'applicazione stessa e non sul web container come proposto dalle specifiche J2EE. In particolar modo, per l'autenticazione è stato fatto l'override del metodo processPreproces() della RequestProcessor come illustrato nel paragrafo dedicato alla sicurezza.

Fatto questo non resta che andare a costruire la pagina JSP in cui inserire il form per il login. Come abbiamo visto, anche qui abbiamo una scelta da fare, ovvero la possibilità di utilizzare le librerie custom di Struts oppure le JSTL. Poiché le seconde come detto, ancora non coprono interamente le prime, ho deciso di adottare solo le librerie personalizzate, anche se la differenza è minima. Inoltre, ho trovato molto utile utilizzare la tecnica di composizione delle pagine, che mira alla riusabilità delle risorse.
In altre parole, se ad esempio osserviamo l' home page, è naturale individuare almeno tre aree: l'intestazione (con il logo), l'area per il login e un'area per la ricerca di un film nell'archivio. Questa suddivisone logica della pagina, si traduce in una vera e propria suddivisione fisica, in quanto ogni area è stata realizzata in una file jsp distinto e poi incluso attraverso il tag <jsp:include> nel file home.jsp che dunque no rappresenta altro che l'unione di più file. Il vantaggio di questa tecnica, ovviamente si evidenzia nelle applicazioni di grosse dimensioni, in quanto porta ad una riduzione e soprattutto ad una maggiore manutenibilità delle pagine.
La prima funzionalità della nostra applicazione web, è a questo punto completata. Per testarla, dopo aver salvato il tutto, sempre da eclipse, avviamo tomcat e poi la

nostra applicazione, come visto precedentemente. Ovviamente prima di fare questo, sarà necessario avviare MySQL e popolare almeno la tabella “utenti” del db.

La procedura illustrata per la realizzazione di l’action di login è valida e può essere seguita per tutte le altre funzionalità della nostra applicazione web. Anche un’applicazione semplice e di piccole dimensioni, può contenere molte Action, risulta quindi superfluo dettagliare tutte le fasi della progettazione, per le quali si rimanda direttamente al codice sorgente allegato a questo lavoro di tesi. Di seguito è riportato solamente un frammento dello struts-config.xml finale nella prospettiva “diagram”:



Tuttavia, ritengo sia interessante trattare alcuni aspetti realizzati della applicazione. Il primo punto è la gestione delle eccezioni, di cui si è parlato a sufficienza nella parte teorica. All’interno della mia applicazione, tra quelle possibili, ho adottato una soluzione mista. In altre parole le eccezioni sono state per lo più gestite in maniera programmatica, usando gli strumenti classici di Java, ma in alcuni punti ho preferito, anche per testarla, la gestione dichiarativa messa a disposizione da Struts. Ad esempio, come si vede dal diagramma dello struts-config.xml, nell’action /regUser (action per la registrazione di un nuovo utente) è stato inserito un tag <exception> per la gestione dichiarativa delle eccezioni di tipo SQLexception. Questa action, ha il comito di acquisire i dati per la registrazione e di richiamare la funzione messa a

disposizione dal livello business registraUtente(), la quale provvederà ad inserire i dati del nuovo utente all'interno del db. Nel fare questo possono sollevarsi eccezioni di tipo SQLException per motivi diversi. Con la gestione dichiarativa di queste eccezioni, per questa particolare action, noi andiamo a specificare nello struts-config.xml che il flusso applicativo deve essere dirottato sulla pagina failure.jsp, dove c'è la visualizzazione di un messaggio di errore ricavato, tramite il solito meccanismo di chiave/valore dal resource bundel. Il codice da inserire nel tag <action > del action con path regUser è il seguente:

```
<exception key="userReg.sql.exception"
           path="/pages/failure.jsp"
           type="java.sql.SQLException"/>
```

In realtà, era possibile gestire questa eccezione, anche a livello globale una volta per tutte e non più della singola action, inserendo lo stesso codice nel tag <global-exception>. Purtroppo, in questo modo diventa poi difficile capire da quale action è stata sollevata l'eccezione e quindi dare all'utente i dettagli sull'errore verificatosi.
Un altro grosso problema che si affronta, nella realizzazione di una applicazione web con Struts è il proliferare delle action. Far cresce il numero delle action, non vuol dire solo spendere maggior tempo nella scrittura del codice, ma anche rendere l'applicazione meno leggibile e mantenibile. Per evitare questo problema, la soluzione principale è fare pratica, la seconda è utilizzare, laddove possibile, la DispatchAction invece della Action. La tecnica per usare le DispatchAction è stata già illustrata, nell'applicaizone da me sviluppata, è stata usata per dare la possibilità all'amministratore del sistema di scegliere l'operazione da effettuare.

Come si vede dalla schermata che viene presentata all'amministratore, questo ha a disposizione dei pulsanti, uno per ogni operazione possibile. Una soluzione implementativa, poteva essere quella di associare una action ad ogni pulsante che dirottasse l'amministratore nella pagine jsp relativa. Di certo non sarebbe stata una soluzione pratica ed elegante, viceversa è stato necessario realizzare una sola action di tipo DispatchAction che preleva dalla request il valore del parametro 'metodo', il cui valore è uguale alla label del pulsante premuto, e in base al suo valore dirotta in maniera opportuna il flusso elaborativo ad altre action o pagine jsp.

Concludendo, è importante sottolineare che quello che ho cercato di mettere in luce è come procedere nel progetto di una applicazione web con struts che come visto, incoraggia lo sviluppatore ad usare in pieno il pattern MVC. Per quel che riguarda gli aspetti pratici, legati al codice, ricordo che il web è disseminato da esempi e turorials che propongono strategie implementative diverse ed eleganti. Alcuni di questi, sono riportati nella bibliografia.

Conclusioni

Quando ho intrapreso questo lavoro di tesi, non avevo alcuna conoscenza di questo framework. Adesso, di certo, non è completa, dato la sua vastità e la sua continua evoluzione, ma penso sia sufficiente per riconoscere che il framework Struts sia uno strumento, come ho letto più volte, *'potente'* e capire perché abbia oramai conquistato il mondo degli sviluppatori di web applications. Struts mette a disposizione dello sviluppatore alle prime armi una piattaforma già pronta e robusta che gli consente di realizzare semplici applicazioni funzionali, scrivendo semplici action; ma nello stesso tempo, fornisce strumenti altamente sofisticati e complessi che permettono a programmatori esperti di realizzare applicazione altamente performanti con un esiguo impegno di energie. Gran parte del so successo, come visto è dovuto alla perfetta aderenza al pattern MVC. Difatti, il successo di una web application, credo dipenda da tre fattori: le funzionalità offerte, la qualità del servizio e l'aspetto (che è la percezione che l'utente ha della stessa), quindi rispettivamente: il model, il controller ed il view. Quindi concentrarsi separatamente su questi tre punti è la soluzione ottimale.

Il lavoro di tesi da me presentato ha cercato di fornire una panoramica del framework ed una analisi del core e sicuramente lascia aperta la strada a tanti sviluppi futuri. Prima di tutto, perché Struts è un framework in continua evoluzione e nel suo sviluppo ha inglobato e continua a farlo tanti altri framework di cui non si è parlato.

Tra questi, possiamo certamente citare il framework Velocity e Tiles per la composizione delle pagine Jsp. Il sottoprogetto Jakarta Commons Logging, che fornisce una serie di strumenti per effettuare il logging della nostra web application.

È sicuramente da suggerire il nuovo framework che sta nescendo a partire da Struts, ovvero JSF (Java Server Faces).

JSF è in sostanza un framework di sviluppo che ha lo scopo di fornire al programmatore degli elementi visuali utili a costruire interfacce utente Web. Come Swing ed AWT per le applicazioni standalone, le applet ed i JavaBean, allo stesso modo i componenti JSF implementano una separazione tra dati e rappresentazione, secondo un modello MVC.

JSF si può per certi versi considerare un "figlio" di Struts, una tecnologia che si sovrappone di molto, senza però essere completamente sostituitiva.

Un’altra direzione, in cui può proseguire questo studio, è inoltre il team working e gli strumenti messi a disposizione da Struts, argomento che ho trovato interessante, ma che non ho trattato per ragioni di tempo. All’nterno di un progetto di grosse dimensioni, una delle carte su cui giocare è la suddivisione dello stesso, non solo intesa come sottolineato più volte in questa tesi, cioè secondo il pattern MVC, ma scendendo ancora più in dettaglio, suddividendo il progetto in più sottoprogetti a se stanti, che dovranno poi cooperare tra di loro. Struts, che come visto è un framework molto valido a livello professionale, non resta indifferente a questa esigenza degli sviluppatori, mettendo a disposizione strumenti appositi, come si è accennato quando si è parlato della suddivisione dello struts-config.xml.

Bibliografia:

[1] Chuck Cavaness - *Programming Jakarta Struts* - OReilly 2002;

[2] Richard Hightower - *Jakarta Struts Live* – SourceBeat 2004;

[3] Mike Robinson, Ellen Finkelstein - *Jakarta Struts for Dummies* – Wiley Publishing 2004;

[4] James Holmes - *Struts: the complete reference* – McGraw-Hill 2004;

[5] Neal Ford – *Art of Java Web Development* – Manning 2004;

[6] Massimiliano Tarquini - *Java Enterprise Computig* – 2001;

[7] Simone Dionisi - *Struts* - http://www.javaportal.it 2002;

[8] Andrea Gini - *Manuale pratico di Java* vol. 1 e 2 – www.mokabyte.it;

[9] Luca Balzerani - *Struttura di una applicazione web* - www.latoserver.it - 18/04/2002;

Dal portale www.Mokabyte.it:

[10] Lavinio Cerquetti - *JSP 1.2: JSTL* - MokaByte vol.67 ottobre 2002;

[11] S.Rossini e L.Dozio - *Il pattern MVC* – MokaByte vol.70 gennaio 2003;

[12] Lavinio Cerquetti - *JSP e Web User Interface* - MokaByte vol.71 febbraio 2003;

[13] Alfredo Larotonda - *Sviluppare applicazioni J2EE con Jakarta Struts* – MokaByte vol.81 gennaio 2004;

[14] Dario Gallozzi - *Perchè usare i pattern* – MokaByte vol.89 ottobre 2004;

[15] Dario Gallozzi - *Le Business Rules nelle applicazioni J2EE* - MokaByte vol.91 dicembre 2004;

[16] Dario Gallozzi - *Sviluppare applicazioni J2EE di grandi dimensioni* – MokaByte vol.94 marzo 2005;

[17] Giovanni puliti - *Accesso ai database in J2EE* - MokaByte vol.94 marzo 2005;

[18] Jonathan Di Nardo - *Autenticazione password crittografata* - MokaByte vol.99 settembre 2005;

[19] Salvatore Giordano - *Applicazioni Web-based* – http://www.prometheo.it 17/07/1999;

[20] Matteo Zinato - *Guida JSP* – www.html.it;

[21] Claudio De Sio Cesari - *Model View Controller Pattern (MVC)* - http://www.claudiodesio.com;

[22] Keld H. Hansen - *The Power of Three - Eclipse, Tomcat, and Struts* – www.javaportal.it;

[23] Giuseppe Capodieci – *Java lato server: le Servlet* – www.latoserver.it 24/11/2001;

[24] Francesco Mele - *Action chaining con Struts* - http://www.javaportal.it;

[25] Paolo Arcangeli – *La sicurezza nelle applicazioni J2EE* – www.javaportal.it giugno 2005;

[26] Giovanni Bronzini – Note sulla sicurezza delle applicazioni J2EE – www.javaportal.it luglio 2005;

[27] Michele Sciabarra - *Eclipse - l'IDE Java OpenSource di IBM* – www.eprometeus.it;

[28] Luca Balzerani – *Istallare Apache su Windows* – www.latoserver.it;

[29] Matteo Zinato – *Guida JSP* – www.html.it;

[30] Autori vari – *Core J2EE Patterns* - http://corej2eepatterns.com/Patterns2ndEd;

Link per reperire il software utilizzato e la relativa documentazione:

[31] Sito ufficiale del progetto Struts http://struts.apache.org/;

[32] Sito ufficiale di Tomcat http://jakarta.apache.org/tomcat/

[33] Sito ufficiale della Sun http://java.sun.com

[34] Sito ufficiale del progetto DBCP http://jakarta.apache.org/commons/dbcp/

[35] Sito ufficiale di Eclipse http://www.eclipse.org

[36] Sito ufficiale del plug-in exadel
http://www.exadel.com;

[37] Sito ufficiale del plug-in EclipseUML
http://www.omondo.org;

[38] Sito ufficiale di MySQL
http://www.mysql.com.