# Frameworks per la implementació àgil de la capa de presentació

**Josep Jiménez Bautista**
**Enginyeria Informàtica**

**Josep Mª Camps Riba**

**18 de juny de 2012**

# Índex General

# 1 Agraïments

Aquest projecte com a etapa final dels estudis universitaris superiors, és la culminació de l'esforç per assolir els coneixements i les capacitats que el fan possible. Per arribar fins aquí han passat tres anys, i durant aquest temps he rebut el suport i els ànims de la meva família, no només per aquest projecte si no durant tots els estudis, i els hi he d'agrair de forma infinita. A la meva germana Pilar sempre pendent de les meus avenços, al meu germà Manel pel seu interès i les seves converses sobre els aspectes més tècnics de la carrera, a la meva germana Toni sempre preocupant-se pels meus ànims, als meus nebots que m'han donat l'empenta i la frescor de la seva joventut. Al meu pare i la meva mare que espero es sentin orgullosos dels objectius acomplerts, com han de ser. I especialment amb molta estimació dono les gracies a la meva dona, companya i amiga Inma, sempre recolzant-me en els moments difícils i gaudint junts dels triomfs aconseguits.

Al meu amic i director general de PYSSA Anselmo Sells, que m'ha animat i m'ha permès disposar del temps necessari tant pels meus estudis com per confeccionar aquest llarg projecte. Als meus companys que han aguantat tot el procés, amb els seus alts i baixos. I als meus amics que no tenien cap dubte que aconseguiria concloure amb èxit estudis, treball i projecte.

A tots ells moltes gracies, aquest petit èxit en forma de projecte, és tant meu com seu.

# 2 Introducció

Durant el transcurs dels estudis d'Enginyeria Informàtica, s'ha treballat en diverses assignatures els aspectes dels components i sistemes distribuïts. J2EE és sens dubte un dels exemples més clars a l'hora d'aplicar aquest tipus de tecnologia. La complexitat que rau sota, només es pot copsar a través de l'estudi del sistema des de un punt de vista del desenvolupador que ha de dissenyar una aplicació basada en aquest paradigma.

En aquest projecte inicialment es donarà una visió global del J2EE aprofundint especialment en els components que s'acostumen a utilitzar a la capa de presentació, com son els servlets. A més es descriuran tecnologies i eines de JAVA que d'un primer cop d'ull pugui semblar no estar relacionades amb aquesta capa, però que es poden aprofitar pel facilitar la feina dels programadors.

No es pretén que aquest projecte sigui un manual de funcionament de J2EE, ni tampoc es pretén aconseguir el desenvolupament complert d'un framework, això seria impossible per la necessitat de temps i de coneixements profunds en diferents disciplines que necessita una eina complexa com aquesta, que a més es solapa entre diferents capes d'una aplicació.

D'aquest projecte es vol extreure un coneixement profund d'aquest tipus de marc de treball i la tecnologia que l'envolta i el suporta. Finalment com a resultat, oferir un framework que incorpori funcionalitats que siguin d'utilitat en el procés de construcció d'una aplicació amb interfície Web. Intentant alleugerir determinades tasques, moltes vegades repetitives, que poden esmerçar temps i recursos, i també facilitar d'altres per tal que el programador no hagi de ser un expert en totes les àrees d'una aplicació, si no només en la que s'ha de centrat.

Per tot això, aquest estudi sobre els frameworks per la implementació àgil de la capa de presentació, oferirà un anàlisis de J2EE centrat en les eines de la capa de presentació. Un examen de les característiques d'alguns dels frameworks de mercat més estesos actualment. La construcció d'un nou framework basat en aquest coneixements, per finalment aplicar-lo a un exemple funcional.

Esperem que la inversió de temps que s'ha fet per confeccionar aquest projecte, aporti d'una forma clara el coneixement sobre les tecnologies actuals en l'àmbit dels frameworks per la capa de presentació. Desitgem sigui útil, oferint a la comunitat de professionals de la programació un nou punt de vista i una nova eina oberta per poder ser explotada i ampliada. Per tot, això aquest document i programari associat s'até al tipus de llicència Creative Commons[1].

[1] Creative Commonsespaña: http://es.creativecommons.org/

# 3 Justificació

Podem dir que a l'hora de fer el disseny i implementació de qualsevol aplicació, ens trobarem que J2EE te mancances, que s'han d'anar suplint amb solucions de collita pròpia. És aquí, on la reutilització del codi ens proporciona a través de l'experiència i de la feina feta amb anterioritat, una base d'eines que en altres projectes podrem utilitzar per donar solucions als entrebancs que hem hagut de resoldre abans.

Els "frameworks" son el resultat de donar solució alguna necessitat concreta, ja per que no s'implementa en un sistema o per facilitar quelcom. L'encapsulament d'aquestes d'eines, que al seu dia algú ha valorat eren útils, i ficar-les a disposició de la resta de programadors, ens proporciona el reaprofitament de l'experiència, i l'estalvi de temps a l'hora de la creació de noves aplicacions.

Des del punt de vista del reaprofitament del codi, i la programació d'eines per programadors, és des d'on es planteja aquest Projecte Final de Carrera. Aprofundint en el connexament del J2EE des de l'estudi de diferents "framework's" dedicats a facilitar d'implementació de la capa de presentació, desenllaçant-se finament en la creació d'un "framework" que aporti algunes eines que facilitin desenvolupament en questa capa.

# 4 Objectius

Com hem dit anteriorment, l'estudi previ del J2EE segurament es trobaran mancances o necessitats del sistema que poden ser corregides o assolides a través d'algun "framework". Els següents objectius son els que a priori es defineixen com a necessaris per dur a terme el tipus de projecte que es proposa.

- Aprofundir en el coneixement de J2EE
- Aprofundir en el coneixement de la capa de presentació, els seus dispositius i eines relacionades.
- Aprofundir en el coneixement del mode de funcionament "frameworks".
- Estudi de diferents "frameworks" aplicats a la capa de presentació.
- Realització d'un "framework" útil per la capa de presentació
- Realització d'un exemple d'ús del "framework" desenvolupat.

## *4.1 Aprofundir en el coneixement de J2EE*

Difícilment es podrà fer un estudi dels diferents "frameworks" destinats a facilitar la implementació de la capa de presentació sense tenir uns coneixements mínims de l'entorn J2EE. L'objectiu d'aquest punt és anar una mica més enllà de simple utilització d'aquest entorn, estudiant les possibilitats que ens proporciona a l'hora de poder utilitzar el seu potencial per poder integrar un "framework", o finalment poder desenvolupar-ne un.

### *4.2 Aprofundir en el coneixement de la capa de presentació, els seus components i eines relacionades al J2EE*

El primer objectiu versa sobre les capacitats del J2EE de una forma més genèrica, a diferència de l'anterior, aquest punt te com a finalitat l'estudi de forma precisa del components que es proposen a J2EE per la gestió de la capa de presentació, i les alternatives existents per tal d'accedir des de la vista cap al model . Amb el primer objectiu juntament amb aquest segon, es pressuposa que s'hauran adquirit els coneixements necessaris per poder abordar els següents.

### *4.3 Aprofundir en el coneixement del mode de funcionament "framework's" a la capa de presentació.*

Existeixen diverses alternatives a l'hora d'escollir un "framework" per se integrat en J2EE. En aquest punt avaluarem les diferents alternatives de mercat que es proposen i el per què de les seves diferències. La finalitat es aconseguir el coneixement necessari per poder escollir-ne tres alternatives existents pel seu estudi, intentant buscar al menys tres que utilitzin tecnologies diferents establint així un criteri de selecció.

### *4.4 Estudi del diferents "framework's" aplicats a la capa de presentació*

Una vegada consolidats els coneixements del J2EE i escollits tres "framework's", la següent fase correspon a l'anàlisi de les tres eines escollides. L'idea principal d'aquest objectiu és poder estudiar el funcionament i les tècniques utilitzades en el seu desenvolupament i utilització, d'aquesta manera podrem anar seleccionant idees per poder finalment aplicar-les durant la fase de realització d'un "framework".

### *4.5 Realització d'un "framework" útil per la capa de presentació*

Aquest objectiu és el més complex de tot el projecte. Partint des de cero i prenent com a base les idees obtingudes de l'anàlisi dels diferents "framework's", es farà una aproximació del desenvolupament a nivell bàsic d'un "framework", que sigui útil per la integració de la capa de presentació en qualsevol aplicació J2EE. Les tecnologies aplicades i funcionalitats que requereix aquest tipus d'implementació s'obtindran de l'estudi dels frameworks existents com a guia de les necessitats que solucionen.

### *4.6 Aplicació del "framework" desenvolupat*

Com a fase final, es procedirà a demostrar el funcionament del nou framework, construint una aplicació Web que implementi aquesta eina. Aquesta aplicació es desenvoluparà des del punt de vista d'un exemple més que el d'una aplicació real, utilitzant totes les funcionalitats creades i amb els comentaris al codi i un "javadoc" extens a mode de guia de la utilització del nou framework.

# 5 Metodologia

Les primeres tasques a realitzar es compondran en gran mesura en l'estudi del J2EE i els seus elements interns utilitzats per poder accedir des de la vista de l'usuari o capa de presentació, cap a la capa de negoci. En aquesta etapa s'ha d'aconseguir els coneixements necessaris, per poder entendre els "framework's" que després s'estudiaran.

Seguidament, i amb els coneixements previs, es tindran les eines per poder analitzar un per un els "framework's" seleccionats pel seu estudi. Intentant copsar el seu funcionament i avaluant les seves principals característiques i el mode en que s'ha d'aplicar. D'aquesta manera, podrem anar descrivint com s'integren al J2EE, la seva tecnologia i les seves qualitats.

Finalment, utilitzant tot el coneixement adquirit i exposat en la memòria d'aquest projecte, es confeccionarà un "framework" que disposi de les eines necessàries per poder facilitar la integració de la capa de presentació. I com a etapa final, aquest "framework" s'utilitzarà a mode d'exemple en una mini aplicació que incorpori totes les eines creades en l'etapa anterior.

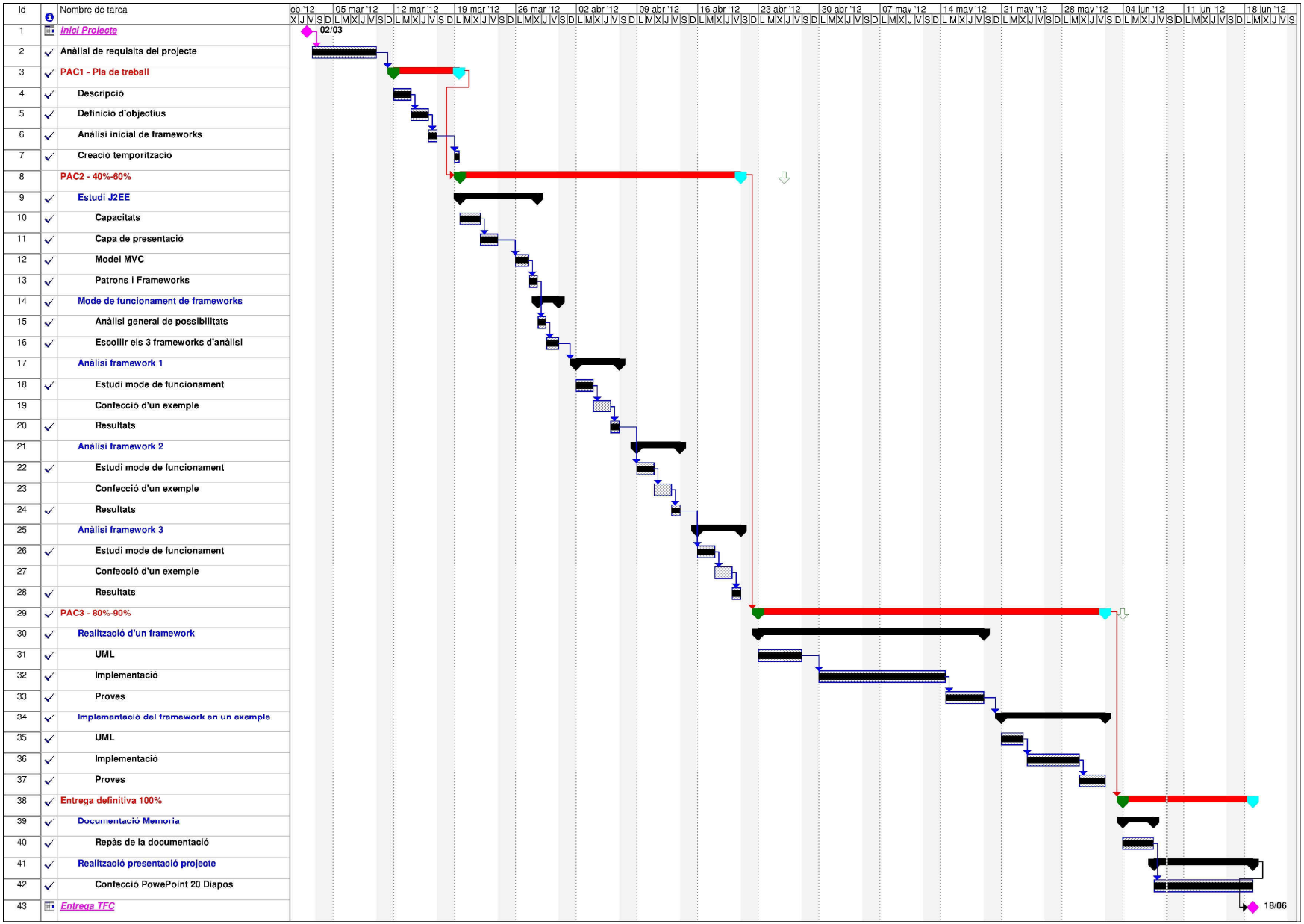
## *5.1 Descriptiu de la memòria*

Aquest document distribuït en diferents apartat, defineix un recorregut lògic per adquirir els coneixements de forma progressiva, per finalment poder copsar la tecnologia que envolta els frameworks de la capa de presentació. Podrem trobar en cadascun dels apartats una consecució lògica de subapartats que aniran concretant les idees exposades, per finalment poder oferir una conclusió dels temes tractats.

Podrem trobar codi, gràfics, imatges i diagrames, a mode d'explicació dels conceptes associats. S'ha intentat mantenir homogeneïtat en l'estructura i presentació, encara que en l'apartat de descripció de les anotacions, s'ha hagut de fer ús del ressaltat en colors per tal d'oferir una diferenciació clara en els exemples de codi. Esperem que resultat es pugui entendre com un document clar i net.

Finalment s'ha de fer especial menció a la bibliografia. Per la confecció del projecte s'han hagut de consultar moltes i molt variades fonts d'informació. És important que el lector d'aquest document pugui accedir també a aquest coneixement per tal d'ampliar les explicacions que s'aporten, per aquest motiu, s'ha confeccionat una bibliogràfica detallada on apareixen les principals consultes realitzades durant la realització d'aquest estudi, les qual és recomanable consultar per completar els coneixements.

## 5.2 Planificació



# 6 Estudi J2EE

## *6.1 Capacitats*

La plataforma J2EE proposa un entorn de desenvolupament i d'integració d'aplicacions unificat multi nivell, tenint la capacitat d'oferir solucions d'arquitectura distribuïda amb les característiques que això proporciona, com son, l'alta disponibilitat, escalabilitat, portabilitat, fàcilment integrables a d'altres sistemes conformant un conjunt heterogeni. A més, les eines de que es disposa també fan més fàcil el desenvolupament i el desplegat, ja que el programador es pot d'abstreure dels elements que no competeixen al seu apartat, com ara, comunicacions, seguretat o accés a les dades.

Anant més enllà, de la introducció del "White Paper[2]" de J2EE, podem deduir que les característiques citades son els objectius que envolten la plataforma, concretament per la versió 6, estendre les seves funcionalitats amb llibreries i "frameworks" de codi obert, abraçant la innovació amb les aportacions externes, amb la finalitat de facilitar les tasques de desenvolupament i desplegament als programadors que decideixen la seva utilització com a mitjà de treball. Son els "frameworks" el nostre cas d'estudi i allà on aprofundeix aquest projecte.

### 6.1.1 Cóm és la plataforma J2EE

S'ha d'aclarir que, aquest projecte no pretén ser una manual de funcionament de J2EE ni tampoc un "tutorial". Però trobem necessari una explicació general del elements de que es composa la plataforma, per després poder copsar i entendre correctament les explicacions que s'aportaran. J2EE, basat en la plataforma J2, obté la seva potencia de la utilització d'aquesta plataforma J2, afegint a més, la seva especial arquitectura basada en contenidors, que ofereixen les capacitats d'abstracció necessàries per fer que el desenvolupador  només s'hagi de centrar en la part assignada, sigui de presentació, negoci, serveis...

#### 6.1.1.1 Servidors d'aplicacions

J2EE es basa en els conceptes de contenidor i component, aquestos dos elements dels quals parlarem més endavant, son implementats per unes plataformes de programari anomenades servidors d'aplicacions. Aquest servidors estan basats en una especificació estàndard per tal de que contenidors i components puguin interactuar i funcionar correctament. Al ser un model obert i estàndard, qualsevol fabricant o equip de programari que segueixi les recomanacions en el seu disseny, pot desenvolupar un sistema capaç de suportar l'arquitectura J2EE.

[2] http://www.oracle.com/us/products/middleware/application-server/050871.pdf

Existeixen molts productes al mercat que ofereixen les funcionalitats i serveis necessaris per poder implementar l'arquitectura J2EE. Alguns d'ells son de codi obert com per exemple JBoss(JBoss Inc), Apache Geronimo(Apache Foundation) o Jonas(ObjectWeb), i uns altres de comercials com son WebSphere(IBM), WebLogic(BEA Systems) o Oracle AS(Oracle Corp). Cadascun d'ells ofereix unes funcionalitats determinades, el què sí que ha de quedar clar és que com a mínim per determinar que un servidor compleix amb els requeriments han d'implementar els serveis de contenidor Web per suportar "servlets" i "JSP" (Java Server Pages), el contenidor per suportar "EJB" (Entreprise Java Bean) i per últim superar el test de compatibilitat J2EE.

#### 6.1.1.2 Contenidors

Tal com ho defineix Oracle, aquest elements son la interfície entre els components i el nivells més baixos de la plataforma, és la funcionalitat per suportar el component. Aquesta funcionalitat serà diferent depenent del tipus de component que hagi de gestionar, i permet que els diferents tipus de components de que es composa una aplicació puguin treballar plegats.

Cada contenidor dona servei a un tipus de component determinat, i depenent de les funcionalitats que proporcioni un servidor, és possible que disposi d'uns o altres. El concepte de contenidor facilita que el programador es pugui abstreure dels nivells més baixos de la implementació, a més de no haver de repetir codi per implementar les mateixes característiques per diferents components. Existeixen dos d'imprescindibles, un per la capa web i un altre per la capa de negoci:

- **Contenidor Web:** Ofereix l'entorn d'execució per als components de tipus Web, com a elements tenim els "Servlets" i els "JSP". Com es pot deduir aquest contenidor pertany a la capa de presentació, on es desplegaran el components per gestionar la interfície d'usuari, on es generaran les peticions i on es mostraran els resultats.

- **Contenidor d'EJB:** Ofereix un entorn d'execució per als "EJB" que son els components del domini del negoci. Aquest contenidor pertany a la capa de negoci, on es desplegaran el components per gestionar la part de l'aplicació que implementaran el que podem denominar com la part funcional.

S'ha d'esmentar que a l'entorn J2EE també es poden definir dos contenidors més, el **contenidor d'aplicacions**, que son aplicacions JAVA en la màquina del client executades en aquesta. I el **contenidor d'applets** que es descarreguen a la màquina del client executant-se al navegador web. Com es pot apreciar aquest contenidors no es troben al servidor, però poden aprofitar els serveis que proporciona per utilitzar la funcionalitat d'una aplicació J2EE. O sigui, és una altra tecnòloga d'interfície, en contes de la utilització del contenidor Web, o inclús es poden implementar altres funcionalitats desenvolupades als propis contenidors.

### 6.1.1.3 Components

Aquest elements de programari, desenvolupats pels programadors, son els que aprofiten els serveis que ofereixen el contenidors. Cada tipus de component ha d'estar desplegat al contenidor que l'hi pertoca. En línies generals, un component Web que doni servei a la capa de presentació per oferir al usuari una interfície, haurà d'estar desplegat al contenidor web d'un servidor J2EE, de la mateixa manera que les peces de programari destinades a implementar l'aspecte funcional de l'aplicació, hauran de desplegar-se al contenidor EJB.

De l'explicació anterior es dedueix que han d'existir com a mínim dos tipus de components, cadascun d'ells amb les seves particularitats i necessariament desplegats al seu contenidor.

- **Component Web:** Aquest tipus de components s'apliquen particularment a la interfície d'usuari, a una petició de servei a través d'un protocol Web com HTTP o SSL, generen una resposta per satisfer aquesta petició. Existeixen dos tipus de components:

    - **Servlets:** Aquest components reben peticions HTTP i genera contingut dinàmic per donar resposta a les peticions. La programació d'aquest component es fa complexa ja que des de la classe s'ha de modelar l'aspecte la pàgina web resultant sense poder utilitzar eines de disseny. Per tant en rebre una petició determina es crearà l'objecte relatiu a aquesta petició, i la resposta serà el modelat de la pàgina web amb el contingut s'hagi definit.
    - **JSP:** (Java Server Pages) A diferència dels servlets, les JSP no son una classe, si no un document XHTML, el qual pot tenir tant contingut estàtic com dinàmic i codi java. Realment la tecnologia del JSP està basada en els servlets, degut a que al desplegar aquest component al contenidor aquest es transforma en un servlet. Així doncs, en rebre una petició dirigida a un JSP s'avalua si el servlet associat ha estat generat i si està actualitzat, provocant que es generi un de nou o s'aprofiti l'actual.

Aquest dos tipus de components, son la base per la comprensió de la capa de presentació per la plataforma J2EE. Encara que a través d'eines i frameworks es pugui facilitar i agilitzar la seva implementació, en una capa subjacent romanen aquest components per suportar el intercanvi d'informació entre el client i la plataforma.

Ens trobarem en molts casos que romanen amagats a l'interior d'alguna eina, però la interfície per antonomàsia es basa en el servlets i les JSP. Més endavant estudiarem alguns casos de frameworks els quals utilitzant aquestes peces de programari implementaran les seves funcions d'una manera més amigable, que en definitiva es la funció d'un bastiment.

- **Component EJB:** Aquest tipus de components s'apliquen particularment a la implementació de la funcionalitat en el domini de negoci. Existeixen tres tipus:

    - **EJB de sessió:** Estan dissenyats per implementar la lògica de negoci. Son els que donen resposta a les necessitats de la funcionalitat que requereix l'aplicació. Podem ser amb estat, o sigui que memoritzen d'alguna manera el seu estat per poder aprofitar-lo en noves invocacions. Els que no tenen estat es destrueixen en acabar la invocació.
    - **EJB d'entitat:** Estan dissenyats per modelar les dades del negoci. Representen les dades que l'aplicació ha de manipular, d'alguna manera podem dir que donen forma a les dades que poden estar contingudes per exemple en una base de dades. Per definició son persistents i duren tan com duren les dades. El contenidor que dona servei aquest component pot gestionar la persistència (CMP), però també es pot optar per gestionar-la al propi component (BMP).
    - **EJB de missatge:** De la mateixa manera que els EJB de sessió, acostumen a implementar la lògica de negoci, però a diferència d'aquestos ho fa d'una manera asíncrona, no amb l'acostumada petició-resposta. Aquest tipus de component el que fa és escoltar i al detectar un missatge concret, i executarà la funció que s'hagi definit al detectar-lo. No tornen cap resposta al client.

Tots els element que s'han descrit fins al moment els podem trobar representats la següent imatge d'una forma conceptual. No s'han descrit interfícies ni peculiaritats de la plataforma, aquest detalls i sobretot els que tenen a veure amb la capa de presentació s'aniran proporcionant durant la descripció d'aquest projecte.

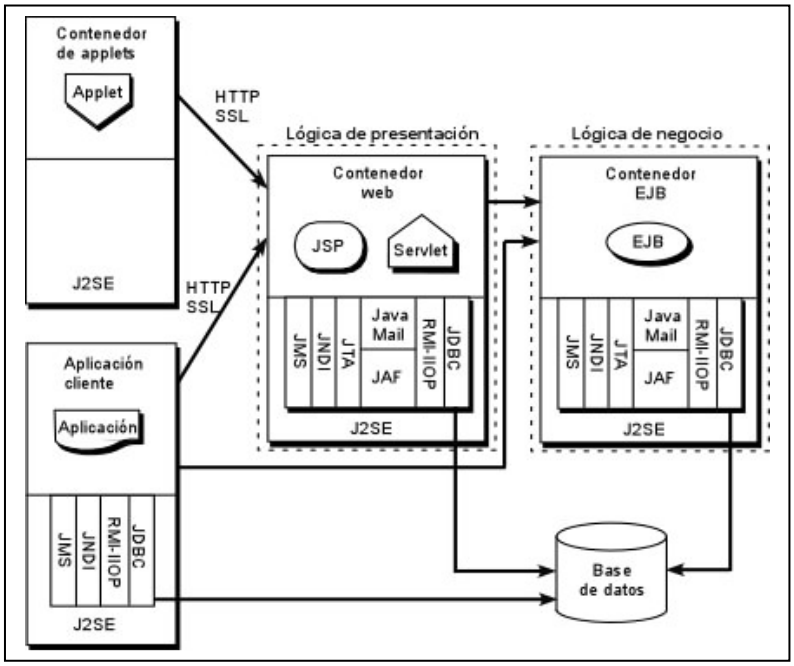



**Fig. Arquitectura J2EE**

## *6.2 Model MVC*

Fins ara s'ha fet una aproximació a la plataforma J2EE, el multi nivell com a tret característic essencial, fa pensar que el model MVC (Model, Vista, Controlador) és el que encaixa a l'hora de realitzar la implementació d'una aplicació desenvolupada sobre aquesta plataforma. Des de aquest punt de vista, intentarem copsar què significa l'aplicació d'aquest patró dins el mon J2EE.

No ens hem de confondre entre patró i framework (bastiment). Metre que un patró és una guia de cóm s'ha d'implementar, construir o crear quelcom, un framework o bastiment, és una peça de programari que implementa unes determinades funcionalitats per poder se utilitzades en una aplicació.

El patró MVC te com a objectiu desacoblar la interfície d'usuari de la resta de l'aplicació. Això és possible dividint el sistema en tres seccions diferenciades. El model correspon a la capa de negoci, és el punt on s'implementaran les funcionalitats que tenen a veure amb el domini del negoci. La vista correspon a la capa de presentació, allà on s'implementarà tot el que te a veure amb la interfície d'usuari. Finalment el controlador és la part de l'aplicació que en molts casos modelarà o ens permetrà l'accés a les dades.

### 6.2.1 MVC amb J2EE

Des de el punt de vista de la plataforma J2EE, s'intueix ràpidament que els contenidors i els components tenen molt a veure amb aquest patró. La vista està representada pel contenidor Web, que correspon a la capa de presentació. El model està representat pel contenidor EJB i els EJB de sessió que corresponent aquest amb la capa de negoci. I el controlador, encara que una mica menys intuïtiu, el representa també el contenidor WEB aplicant per exemple el patró "façana" a través d'un servlet per capturar les accions, retornant la informació a través de la vista que pot ser un JSP. A la següent figura es pot apreciar el seguiment d'aquest patró.

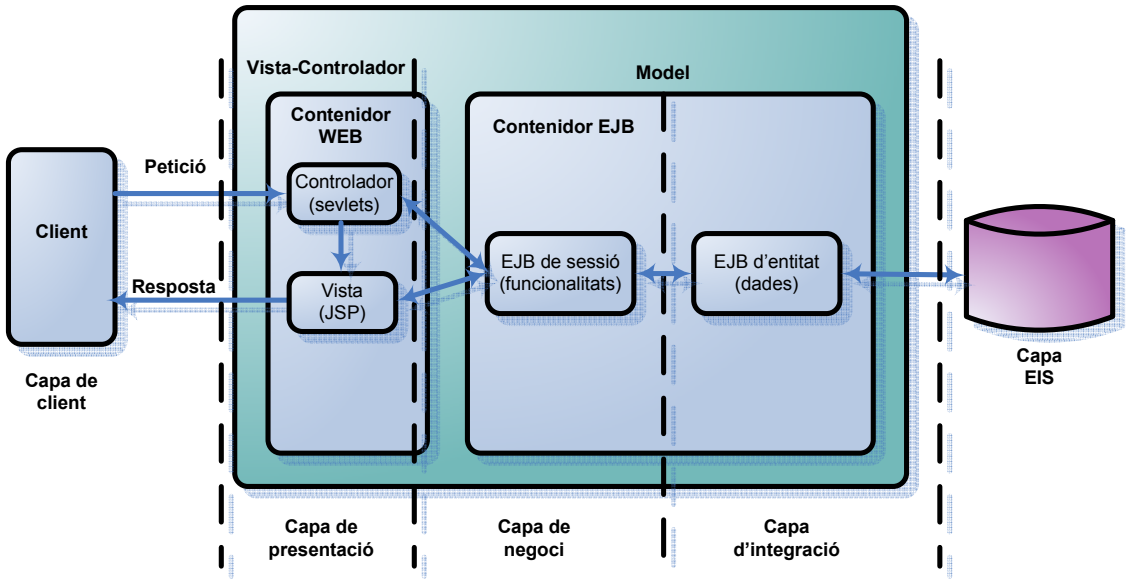



Fig. Arquitectura MVC des la perspectiva J2EE

D'un cop d'ull a la imatge anterior queda clar que aquest projecte està enfocat a la vista i al controlador dins del patró MVC. La separació de la capa de presentació representada al J2EE pel contenidor WEB proporciona certs avantatges. Es pot utilitzar una tecnologia que no sigui la original del J2EE a l'hora d'implementar aquesta capa. Els desenvolupadors tenen la possibilitat d'especialitzar-se en el domini d'aquesta tecnologia. Una actualització de la interfície no te per que afectar a la resta de l'aplicació, els rentats de cara son molt freqüents en el cicle de vida del programari.

Aquesta divisió tant clara del J2EE, no ha estat així des del seu inici. És el resultat de l'evolució. Inicialment no existia aquest concepte i les aplicacions barrejaven als JSP la lògica de negoci amb la interfície d'usuari, complicant el manteniment i la reutilització del codi. Arrel d'això va a aparèixer el Model-1 basat en el JSP, el client interactuava directament amb aquest i aquest amb la capa de negoci, el sistema es tornà rígid, ja que cada JSP era determinista en quan al resultat. Posteriorment es va evolucionar cap al Model-2, el que es manté avui en dia explicat al punt anterior, i al qual es basen els framewoks actuals destinats a facilitar la capa de presentació.

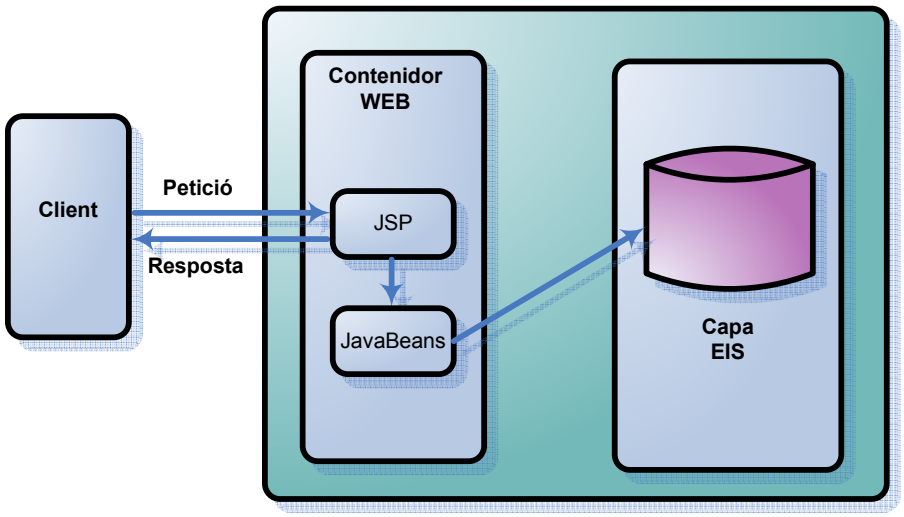



Fig. Model 1

Amb les explicacions aportades fins ara, ja es pot distingir quins elements de la plataforma J2EE encaixen amb el patró MVC. Per la capa de presentació, la que a nosaltres ens interessa, podem veure que els servlets i el JSP son els element principals d'aquesta estructura, els primers fan d'enllaç amb el client gestionant peticions i respostes, i el segon utilitzat per, d'una forma còmoda, poder modelar la informació que s'ha de presentar per tal que resulti coherent, profitosa i per què no també atractiva.

### 6.2.2 Detalls sobre J2EE 6

L'arquitectura que s'ha exposat fins ara, es pot trobar a qualsevol aplicació que s'hagi desenvolupat amb tecnologia J2EE 5. Però amb aquesta última versió 6, s'han modificat determinats conceptes, els que més ens afecten son els relatius a la versió d'EJB utilitzada, la nova API EJB 3.1. Com a punt a destacar, la utilització de la tecnologia Annotations s'ha intensificat i s'aprofita per modelar els comportament dels javaBeans de les aplicacions.

S'ha de fer notar que en aquesta nova arquitectura els Beans d'entitat canvien, amb aquesta nova versió, es representen a través de classes POJO, i l'encarregat de gestionar-los és l'EntityManager, un servei subministrat pel contenidor d'EJB's inclós a l'API de persistència de Java (JPA). Per tant ens podem trobar representacions diferents d'aquest tipus d'EJB en les aplicacions basades en l'última versió de J2EE.

S'han introduït nous conceptes con les vistes sense interfície que permeten l'accés directe als beans des de la capa de presentació, la instanciació única "Singleton", invocació de bean de forma asíncrona obtenint processos paral·lels, inclusió de sistemes de pre i post processat en l'execució dels javaBeans, inclús una versió simplificada dels EJB's "Lite". Hi han moltes altres diferències per tenir obtenir més informació es pot accedir a la pàgina Java Community Process[3].

L'ús del EJB 3.1 efectuat a través de les anotacions de java, fan que un component de terminat definit a través d'una classe i la seva l'anotació, no es converteixi en aquest tipus realment fins que no s'ha desplegat al contenidor, o sigui fins que no ha estat interpretat i modelat com a tal. Així ens trobarem amb els conceptes d'EJB reflectits i reals. El primers son components que encara no has estan desplegats, per tant l'únic que els defineix és la seva anotació, en canvi els components desplegats incorporen tots els elements d'un EJB.

A nivell conceptual tota l'arquitectura que s'ha exposat fins és valida per la finalitat d'aquest projecte, encara que amb un estudi més profund es podrien cercar alternatives a l'hora d'implementar programari adaptat a aquesta versió. Si fos així, el punt negatiu seria que no es podria reutilitzar el codi en versions anteriors, amb les quals hi ha força aplicacions que l'utilitzen.

En el següent punt parlarem de la capa de presentació en J2EE, a arrel d'això s'ha de comentar que en la versió 6 ha aparegut una funcionalitat per poder realitzar vistes sense la necessitat d'incorporar la interfície requerida. Com aquest projecte tracta sobre els frameworks de la capa de presentació, s'ha de puntualitzar que no es creu que s'hagi de tractar aquesta funcionalitat ja que es un camí intermedi als nostre propòsits i no és propicia una separació clara entre capes.

[3] Java Community Process http://jcp.org/aboutJava/communityprocess/final/jsr318/index.html

## *6.3 La capa de presentació a J2EE*

El contenidor Web, els servlets i els JSP, son els elements bàsics de la capa de presentació en aquesta arquitectura. Als punts següents explicarem més detalladament quina es la funció i les característiques de cadascun d'aquests elements. Pararem especial atenció als servlets i els JSP, ja que son els elements que tenen més a veure amb la implementació de les aplicacions.

### 6.3.1 Contenidor Web

S'ha de fer notar que el contenidor Web depèn d'un servidor Web que l'hi proporciona una interfície, aquest servidor és el responsable d'encaminar les peticions cap al contenidor Web al que correspongui una determinada petició. Aquestes peticions s'associen a un element del contenidor, fent aquesta associació en temps de desplegament. Les peticions poden ser per a un servlet, una JSP, contingut estàtic amb HTML, imatges o arxius. Podem dir que el servidor Web actua com a interfície amb el contenidor Web.

El contenidor web proporciona l'entorn d'execució i els serveis necessaris per servlets, JSP's..., invocant els serveis que correspongui per exemple a través dels servlets partint de les peticions del client, i retornant les respostes aquestes peticions utilitzant per exemple JSP. També es la seva responsabilitat controlar el cicle de vida dels diferents elements que es generen a través de les peticions. Per tant una vegada el client emet una petició i el servidor Web la passa al contenidor que correspongui, és aquest el que l'executa. El client mai podrà executar un servei directament.

### 6.3.2 Servlets

D'una forma molt resumida, podem dir que els servlets son els encarregats de rebre les peticions del client i enviar les respostes. Als processos interns, per poder generar la informació que reclama el client o realitzar l'acció que es demana, intervindran altres elements com EJB's o classes POJO (Plain Old Java Objects). Per tant, els servlets proporcionen una interfície cap a la capa de negoci, i des de aquesta, cap al usuari.

Des de el punt de vista del programari, son peces que capturen les peticions, comuniquen amb els serveis de la capa de negoci i esperen una resposta d'aquestos per poder oferir un resultat al client. D'una manera molt purista, cada petició d'usuari correspon a un servlet, així doncs tindríem tants servlets com tipus diferents de peticions tingui la nostra aplicació.

Si implementéssim la capa de presentació d'aquesta manera, la capa de presentació d'una aplicació s'allargaria molt, obtenint un codi molt repetitiu. Per pal·liar aquest inconvenient existeixen patrons d'implementació, com el patró "façana" fent que totes les peticions passin pel mateix servlet i aquest distribuir-les, podent utilitzar el patró "factoria" el qual s'encarregarà de crear l'objecte que tingui a veure amb la petició passada pel servlet.

La tecnologia dels servlets està basada en l'API Java Servlet, actualment en la versió 3.0 sota la especificació JSR 315. Aquesta API es composa de dos paquets el "*javax.servlet*" i el "*javax.servlet.http*" que contenen diverses interfícies, classes, classes abstractes i excepcions. Per poder utilitzar aquesta tecnologia en un servidor Web, ha d'implementar Java Servlet, alguns exemples son Apache Tomcat, Allaire JRun, Sun's Java Web Server. El contenidor Web i el servidor Web son els que porten el control dels objectes que tenen a veure amb el Java Servlet.

Si deleguem la responsabilitat del control del servlets al servidor Web i al contenidor Web, què ens queda per implementar un servlet?. Be doncs, entrant al detall de codi, per poder implementat un servlet bàsic, només s'ha d'estendre la classe HttpServlet sobre la classe que serà el nostre servlet i sobrecarregar els mètodes doGet i doPost, depenent del tipus de comunicació que tinguem amb el client, i tot seguit implementar la resposta. El següent codi és un senzill exemple:

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Public class HolaMon extendsHttpServlet{
      protected void doGet(HttpServletRequest request,  HttpServletResponse response) throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        out.println("Hola Mon del J2EE Març 2012"); }
}
```

Fig. Detall codi Servlet sortida simple

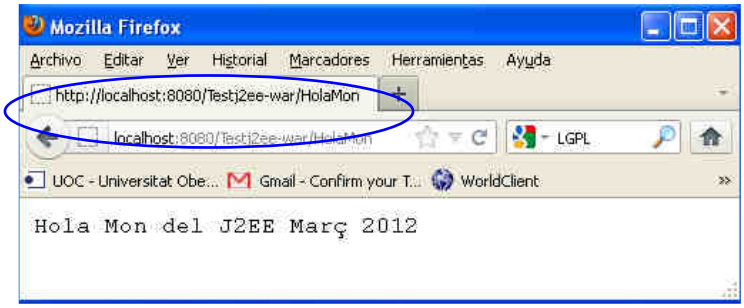



Fig. Resultat codi Servlet

Com es pot apreciar la implementació és molt senzilla, la crida des del navegador "http://localhost:8080/Testj2ee-war/HolaMon" provoca la petició al servlet amb nom "HolaMon" fent que al navegador en aparegui la frase "Hola Mon del J2EE Març 2012". Aquest exemple s'ha desplegat sobre un servidor d'aplicacions Grassfish 3.1, al qual només s'ha hagut d'indicar el nom del servlet i on es troba per poder ser executat, a través del arxiu de desplegament web.xml.

```
<servlet>
   <servlet-name>HolaMon</servlet-name>
   <servlet-class>Hlm.HolaMon</servlet-class>
</servlet>
<servlet-mapping>
   <servlet-name>HolaMon</servlet-name>
   <url-pattern>/HolaMon</url-pattern>
</servlet-mapping>
```

Fig. Detall configuració web.xml

Tal i com s'ha explicat amb anterioritat, si es volgués formatar la pàgina, s'hauria d'implementar tot el codi HTML al servlet, com es pot veure al següent exemple:

```
Import java.io.*;
Import javax.servlet.*;
Import javax.servlet.http.*;
Publicclass HolaMon extendsHttpServlet{
  protected void doGet(HttpServletRequest request, HttpServletResponse response)  throws ServletException, IOException {
        PrintWriter out = response.getWriter();
        String titol = "Servlet formatat";
        String contingut= "Hola Mon del J2EE Març 2012";
        response.setContentType("text/html");
        out = response.getWriter();
        out.println("<HTML><HEAD><TITLE>");
        out.println(titol);
        out.println("</TITLE></HEAD><BODY>");
        out.println("<H1>" + titol + "</H1>");
        out.println("<P>"+ contingut);
        out.println("</BODY></HTML>");
        out.close();}}
```

Fig. Detall codi servlet amb sortida complexa

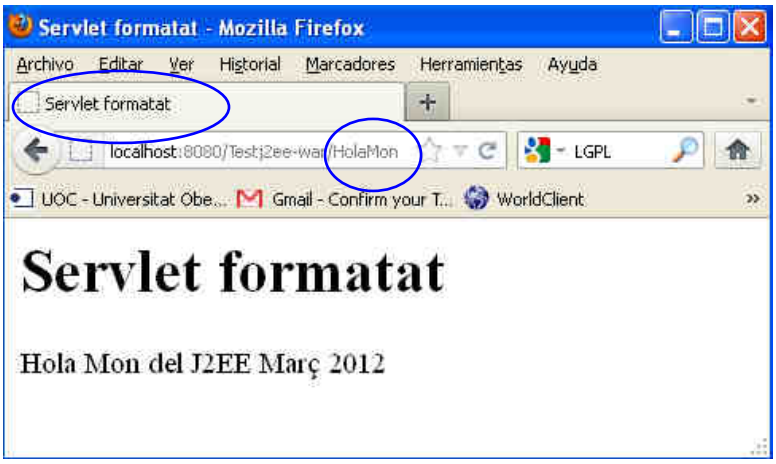



Fig. Resultat codi Servlet complex

S'ha donat format al títol de la pàgina, i al navegador es mostra la informació que s'ha hagut de codificar completament al servlet. Es pot veure que el codi s'allarga molt, i és poc intuïtiu a l'hora de donar forma a un document en HMLT.

Degut a tots el inconvenients que hem explicat, i als que s'intueixen per la complexitat a l'hora de confeccionar un document amb contingut dinàmic i formatat d'una manera més o menys complexa, J2EE aporta una solució amb les JSP (Java Server Pages).

### 6.3.3 JSP

Els JSP son documents XHTML els quals poden oferir, contingut dinàmic a través de codi JAVA implementant directament al codi de la pàgina. També suporten contingut estàtic, i el formatat es fa directament al document. Aquesta manera de procedir converteix la implementació en quelcom més intuïtiva i molt més senzilla a l'hora de ser interpretada.

La tecnologia dels JSP es basa en els servlets, degut a que la manera de procedir del contenidor Web davant d'una Java Server Page deriva en convertir aquesta pàgina en un servlet, codificat aquest amb les sentències HTML necessàries per que el servidor Web pugui mostrar el contingut. El codi de d'una pàgina JSP amb contingut estàtic format, de la mateixa manera que el servlet seria el següent:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Pagina JSP</title>
  </head>
  <body>
    <h1>JSP formatat</h1>
    <P>Hola Mon del J2EE Març 2012</p>
  </body>
</html>
```

Fig. Detall codi pàgina Web index.jsp


Fig. Sortida index.jsp

D'un cop d'ull es pot apreciar que el codi en HTML del JSP, és el que es mostra a la plana Web. No s'han de fer codificacions especials, és intuïtiu i ràpid d'entendre. Aquesta característica permet incorporar eines de desenvolupament gràfic per disseny de planes Web, sense que importi quina tecnologia s'aplica a la resta de nivells. D'això es desprèn que la separació de la interfície gràfica o capa de presentació, permet que el personal dedicat al disseny no te per que conèixer la lògica del negoci, separant rols de treball, afegint efectivitat al desenvolupament.

### 6.3.4 Servlets + JSP

S'han analitzat separadament els servlets i les JSP. Hem vist que un servlet pot generar contingut dinàmic, i que una JSP te facilitat per formatar i generar contingut estàtic en una plana web. L'efectivitat d'aquestes dues tecnologies s'assoleix quan son utilitzades conjuntament. Un servlet no deixa de ser una gestió petició/resposta en HTML per un usuari, i una JSP és una plana Web que pot fer peticions tipus Get i Post. Així doncs, no hi ha res que impedeixi que des d'una pàgina JSP es cridi a un Servlet per demanar un determinat contingut, i aquesta JSP el formati i el mostri convenientment.

D'es d'aquest punt de vista, cóm seria una codificació que aprofiti les dues tecnologies?. Al següent codi es pot apreciar la crida al servlet "HolaMon" del segon exemple cridat des d'una modificació del JSP del tercer exemple:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Pagina JSP</title>
  </head>
  <body>
    <h1>JSP amb crida a un servlet</h1>
    <P>Peticions</p>
  <li><font face="geneva,verdana,arial" size="4"><a href="/Testj2ee-war/HolaMon">Petició 1</a></font></li>
  </body>
</html>
```

Fig. Detall codi pàgina Web amb crida al servlet

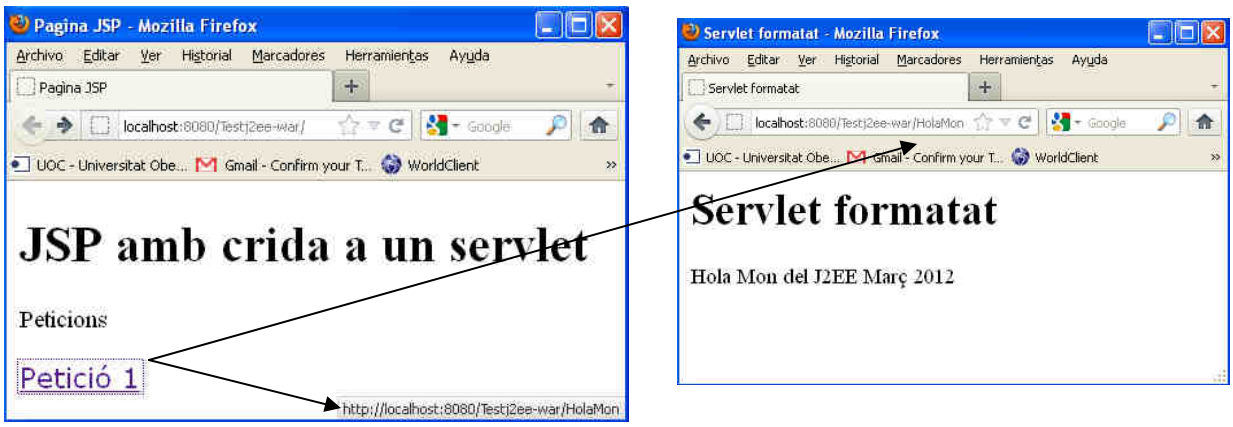



Fig. Detall sortida i resultat de l'execució de la crida

Aquest exemple mostra la capacitat de les JSP per poder accedir al codi d'un servlet. Encara que a través del servlet es mostra un contingut estàtic, aquest podria accedir a la capa de negoci per recuperar unes determinades dades i mostrar-les a la pàgina JSP. Tot aquest entrellat necessita una arquitectura de carpetes i arxius de desplegament, a la següent imatge es pot veure de quina manera s'ha estructurat l'exemple per aconseguir aquest desplegament:

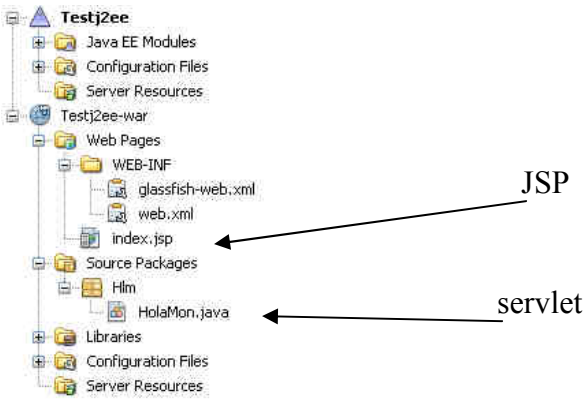



Fig. Estructura de projecte

## *6.4 Patrons i frameworks*

Després d'analitzar la capa de presentació del J2EE, s'obre tot un ventall de possibilitats a l'hora de realitzar la implementació d'aquesta capa en una aplicació. Podem optar per treballar directament amb les eines que ens proporciona, i sens dubte, obtindríem un producte correcte que proporcioni les funcionalitats necessàries. Però existeixen ajudes per estalviar feina, tant en la estructuració com en la seva construcció, estem parlant de patrons i frameworks.

### 6.4.1 Patrons

Els patrons consisteixen en un conjunt de recomanacions que proposen solucions a problemes que son habituals durant el disseny de programari. Es pot dir que son una solució estàndard a un problema comú de disseny. La particularitat principal rau en l'experiència aportada en aquestes solucions, o sigui que s'ha demostrat la seva efectivitat en casos similars, oferint flexibilitat per ser aplicats en diferents escenaris.

J2EE és susceptible de la utilització de patrons en qualsevol punt, depenent del problema que es vulgui afrontar podrem aplicar diferents tipus. En el nostre cas, i parlant sempre de la capa de presentació, trobem de tant importants com l'estructuració per capes aplicant el patró MVC, el qual ja l'hem comentat abans, i és més que recomanable la seva utilització en aplicacions amb base Web.

Com a habituals en aquest nivell trobem el patró "Factory" utilitzat per poder instanciar objectes de classes que no es poden definir en temps de compilació, i que s'ha de fer en funció del les dades que es passen com a paràmetres. Aquest patró resulta d'especial utilitat en el moment que un servlet rep una petició determinada, aquesta petició es passa a la factoria per que l'analitzi i triï la classe a instanciar, fent accessibles els seus mètodes i els tipus per poder donar resposta a la petició.

Acompanyant al patró "Factory" acostuma a anar el patró o interfície "Command" al qual s'ha d'implementar el mètode execute(). La utilització habitual a la capa de presentació es fa passant com a paràmetres del mètode "execute" el HttpServletRequest i el HttpServletResponse proporcionats pel servlet. I com a resposta acostuma a subministrar un String amb la pàgina JSP que toca mostrar per la petició rebuda amb les dades que s'han de mostrar.
Per la comunicació entre les capes de presentació i negoci i oferir un desacoblament efectiu, s'acostuma a utilitzar el patró "facade" façana. Aquest patró refina la interfície entre capes, forçant que totes les peticions s'efectuïn a través de la façana. Qualsevol funcionalitat del negoci o accés a les dades s'ha d'implementar en aquesta part, sense donar lloc a fer salts entre capes.

Tots aquest patrons i la funcionalitat explicada no deixen de ser exemples. Existeixen molts d'altres de recomanables depenent de la necessitat durant el disseny de l'aplicació. El què sí és segur, es que els frameworks s'aprofiten d'aquests patrons de disseny, amb les característiques que proporcionen, fent més entenedor el seu funcionament ja que les característiques generals dels patrons universals.

### 6.4.2 Frameworks

Aquestes peces de programari tenen com a objectiu proporcionar un marc de treball que faciliti determinades tasques en la construcció d'aplicacions. Acostumen a distribuir-se empaquetades en mòduls, on cada paquet ofereix determinades eines. Al nostre cas, de frameworks per la capa de presentació, n'hi han de diferents, cadascun d'ells amb funcionalitats per donar servei als programadors, i com no, amb diferents procediments per aplicar la seva tecnologia.

Pe què s'haurien d'utilitzar aquests paquets?. Durant la confecció de la capa de presentació apareixen diverses necessitats que es repeteixen per totes les aplicacions. Començant per la manera que les peticions arriben des de l'usuari cap al nucli de l'aplicació, continuant com s'ha de formatar la resposta per que el servidor Web l'entengui, el control de les dades d'entrada i sortida, etc...

La repetibilitat de determinat codi, i el fet de que s'ha de donar una solució segura i fiable, ens dona la resposta a la utilització de frameworks. Son elements de software que ja estan provats, que funcionen correctament i ens donen la seguretat i la fiabilitat sense necessitat d'implementar tot aquest codi, que d'una altra manera, implementat per un desenvolupador, caldria sotmetre'l a un banc de proves per garantir el seu funcionament, estabilitat i seguretat.

Afegint avantatges, podem dir que el desenvolupador, una vegada coneix l'eina, l'hi permet guanyar temps en les seves tasques, ja que es pot abstreure de certs aspectes de la implementació que no tenen a veure directament amb la part de presentació. Hi ha frameworks al mercat que ofereixen funcionalitats per la generació d'accions de forma gairebé directa, pel control i validació de dades i tipus, generar llistes de forma automàtica, i moltes d'altres.

Finalment s'ha de dir que existeixen inconvenients, el primer i principal és la necessitat d'un aprenentatge previ de l'eina, amb el temps que això suposa, sens dubte, aquest temps invertit després es recuperarà ràpidament. I com no, les eines no son perfectes i tenen mancances limitant la seva utilització en determinats aspectes, que s'hauran de suplir introduint codi que s'adapti al framework i a l'aplicació. Inclús amb aquest inconvenients es valora positivament la utilització d'aquestes eines.

# 7 Mode de funcionament dels frameworks

## *7.1 Característiques comuns de funcionament*

Després d'una anàlisi inicial de diferents frameworks de la capa de presentació que existeixen al mercat, trobem característiques comunes en la seva estructuració i forma de funcionament. Part d'aquesta similitud es deguda a que aquests frameworks s'han d'adaptar a l'arquitectura J2EE, i en part també es deu a que la utilització de patrons al seu desenvolupament marquen una pauta en la seva construcció.

Si parlem dels elements de configuració, acostumen a estar basats en arxius XML, els quals contenen la definició dels elements que entren en joc durant l'execució, tals com per exemple, la definició de la ruta d'una acció determinada, amb l'objectiu que el controlador de les peticions pugui trobar aquesta acció i es pugui executar. Poden ser configuracions tant senzilles com aquesta, o també la seva complexitat pot acabar definint la naturalesa d'una resposta davant d'una acció, com el control del resultat, si es correcte o no.

Sense perdre de vista que els arxius de configuració que sempre hi son presents en major o menor mesura, també s'apliquen altres tecnologies a l'hora definir una configuració, com per exemple la configuració per convenció de noms, delegant al framework quina és la URI de cada acció en funció del seu nom. En molt casos també s'utilitza la configuració utilitzant "Annotations" de JAVA, en aquest cas analitzat determinades indicacions al codi amb una "@" coma capçalera, en temps de compilació i de execució podem modificar el comportament dels elements que les incorporin, com ara mètodes i tipus.

Una de les dificultats que es troba habitualment, és la gestió de les peticions i respostes dels clients. La majoria de frameworks donen solució a aquesta dificultat donant un punt comú d'entrada, una façana o interfície, per la qual passin totes les peticions i s'encarregui de dirigir-les al seu destí basant-se en diferents criteris, com per exemple els definits als elements de configuració. Així d'aquesta manera s'aconsegueix alleujar les tasques gestió de les peticions.

La presentació dels resultats d'una petició determinada, també és una tasca que requereix un esforç important durant el procés d'implementació d'una aplicació, per això la majoria de frameworks proposen eines, com les llibreries de "tags" variables i els "templates" plantilles, per facilitar les tasques de manipulació de dades i com presentar aquestes a l'usuari.

En conclusió, la majoria de frameworks plantegen una arquitectura molt similar, oferint les eines comunes per solucionar els problemes habituals en la confecció de la capa de presentació. Depenent de la qualitat del framework proporcionarà més o menys funcionalitats, però com a mínim han de ser capaços d'alleugerir les tasques de gestió de peticions i la confecció d'una manera àgil de les respostes.

## *7.2 Selecció dels frameworks de l'estudi*

En una recerca realitzada per la Web, apareixen un gran nombre de frameworks destinats a la capa de presentació per aplicacions sota l'arquitectura JAVA. Degut a que tots tenen en comú la mateixa base on s'han d'implementar, tenen molts trets que a primer cop de vista els fan semblar similars. Alguns dels que s'han trobat son els següents:

| |
|---|
| Spring Web MVC |
| JOSSO |
| WebWork |
| Maverick |
| SOFIA |
| Verge |
| Java Server Faces |
| Struts |
| Struts 2 |
| Click |
| Aurora |
| Grails |
| JAT |

En una anàlisis de més a prop, cadascun d'ells incorpora diferents característiques amb els seus avantatges i inconvenients. La pauta per escollir-ne tres ha estat que puguin aportar un coneixement global i actual del estat d'aquesta tecnologia, i que fossin suficientment estesos per no tenir problemes amb la documentació necessària per realitzar l'estudi.

Sota aquesta premissa en primer lloc s'ha escollit Struts 2, un framework que ha estat actualitzat des de la seva primera versió i que incorpora la intenció de configuració cero, és un framework força estès, del qual es troba molta informació i amb llibreries molt interessants.

En segon lloc s'ha escollit Spring Web MVC Framework, basat en el patró MVC i gual que l'anterior, aquest framework integrat a dins del conegut framework Spring, incorpora característiques semblants a l'anterior, però amb una arquitectura diferent en el tractament de la captura d'accions. També es troba força informació sobre la seva arquitectura i implementació.

En tercer i últim lloc, s'ha escollit Java Server Faces degut a que aquest framework parteix de la Java Community i es pot dir que és la base dels frameworks per la capa de presentació en JAVA, degut a la seva procedència ha semblat convenient fer un estudi sobre aquesta eina.

# 8 Struts 2

## *8.1 Introducció*

Aquest framework va ser originalment desenvolupat per Craig McClanahan a l'any 2000 amb el nom STRUTS. A l'any 2002 va passar a pertànyer a l'Apache Software Foundation. Posteriorment i degut a les limitacions de la primera versió, i a l'aparició de nous frameworks que aporten nous conceptes, es decideix millorar l'eina aportant més flexibilitat, fiabilitat i facilitat d'ús, apareixen el nou **STRUTS$^2$**.

En aquesta nova versió no s'ha fet un rentat de cara i s'han afegit funcionalitats, si no que s'ha canviat la direcció i s'ha basat en WebWork un altre framework de desenvolupament Web. Canviat per complet el codi base i la seva arquitectura, mantenint el patró MVC, per tal de reduir l'acoblament entre els tres nivells diferenciats.

El Model, que fa referència al les regles de negoci l'encarnen les accions. La vista, encarregada de genera la interfície amb el client equival als resultats. I el controlador que comunica la vista i el model està implementat mitjançant el filtre "FilterDispatcher". Per poder donar aquestes funcionalitats STRUTS 2 requereix les versions mínimes de Servlet API 2.4, JSP API 2.0 y JAVA 5, encara que després de proves s'ha determinat que és compatible amb la versió JAVA 6.

## *8.2 Arquitectura*

Durant les explicacions anteriors s'ha fet referència al concepte de servlet com a base per establir un punt d'accés amb el client. STRUTS 2, s'utilitza el patró "FrontController" a l'hora de proporcionar aquest punt d'accés, però a diferencia de la primera versió, la tecnologia aplicada per anàlisi, gestió i distribució de les peticions son els "Filters" de JAVA.

Com a elements importants de l'arquitectura temin:

- FrontController: Com apunt d'entrada
- ActionMapper: que determina quina és l'acció que s'ha d'executar.
- ActionContextCleanUp: filtre per l'anàlisi de peticions d'una altra tecnologia.
- FilterDespatcher: Determina si la petició ha de ser una acció o un altre element.
- ActionProxy: Crea el "ActionInvocation" per executar l'acció.
- Interceptor: Processament PRE o POST d'una acció.
- ActionInvocations: Executa el "Action" tenint en conte la configuració
- Action: Conté la lògica de negoci.
- Result: És la sortida del "Action" que determina la vista a presentar al client

No hem de perdre de vista que aquests element sempre tenen per sobre la configuració que s'ha realitzat per cadascun d'ells, que pot ser a través dels diferents arxius XML, els "Annotations" o per convenció de noms, executant el comportament que s'ha definit prèviament.

A la següent imatge es pot apreciar l'arquitectura de STRUTS 2 i el camí que pot seguir una petició:

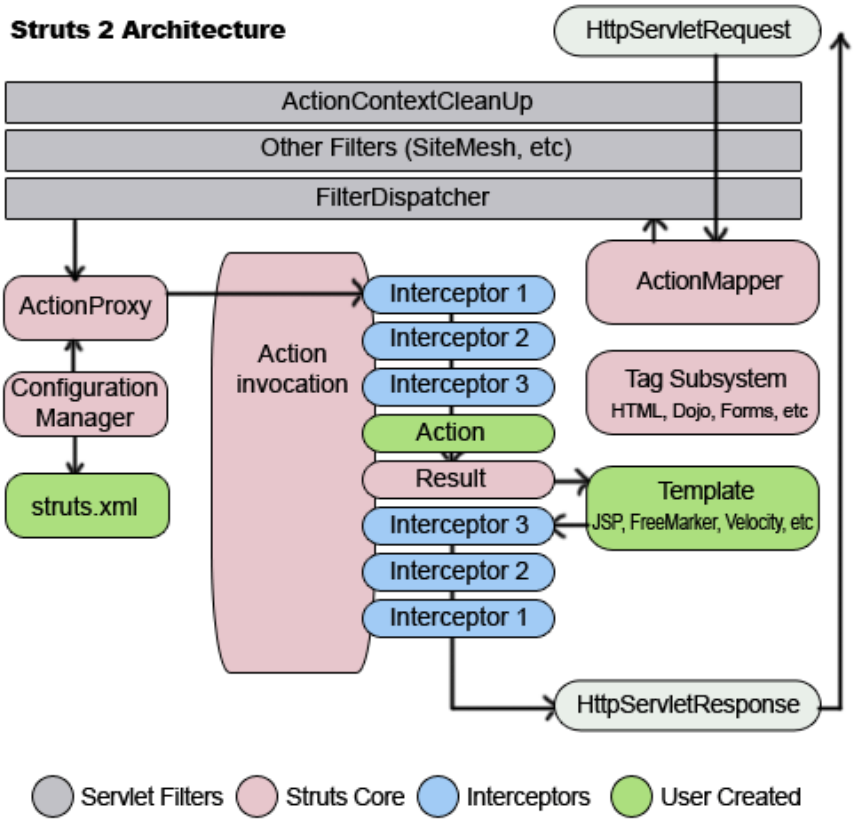



Fig. Arquitectura STRUTS 2

El "FrontController" que implementa la Servlet API, a través del HttpServletRequest rep la petició delegant-la al "ActionMapper", que determina quina és l'acció que s'ha d'executar. Seguidament s'executa la cadena de filtres, arribant la petició al "ActionContextCleanUp" si es una petició d'un tipus especial integrat per una altra tecnologia, o al "FilterDispatcher" per determinar si la petició ha de ser una acció o un altre element.

Si el "FilterDispatcher" determina que la petició correspon a un "Action", delega al "ActionProxy" l'execució, aquest component a través del gestor de configuracions analitza el "Action" i crea un "ActionInvocation" que implementa el patró "command" amb el mètode "execute()" en aquest moment ja sap quina és la classe de l'acció que el mètode "execute()" portarà a terme i quins son els interceptors configurats per aquesta acció.

La classe corresponent al "Action" una vegada executada, donarà com a sortida un "Result" el qual s'utilitza per determinar la vista que s'ha de mostrar, aquesta vista pot ser una plantilla, un JSP, un element d'un altre framework, etc... El resultat pot tornar a passar per altres interceptors i finalment és passa el resultat al "FrontController" que a través del HttpServletResponse retornarà el resultat de la petició.

## *8.3 Elements principals*

Al punt anterior s'ha descrit l'arquitectura i el funcionament del framework on apareixen alguns del components principals que l'integren. Encara que s'intueix el seu rol, aportem una explicació més detallada d'alguns d'ells i exposarem d'altres que no hem especificat.

### 8.3.1 FilterDispatcher

En aquest component es deleguen moltes de les tasques de manipulació de les peticions. És el que proporciona l'accés a la infraestructura necessària pel procés de les peticions, tenint en conte les configuracions que s'han fet dels elements que ho permeten. Segons el plantejament d'algunes documentacions, tan el ActionMapper com el ActionProxy els engloba dins la tecnologia del FilterDispatcher, encara que cadascun d'ells te una tasca definida.

El FilterDispatcher serà l'encarregat d'invocar el ActionProxy que conté les configuracions i la informació per processar les peticions i traduir-les en Actions a través del ActionInvocation, aquest a la vegada implementant el patró command a través del seu mètode execute() serà capaç de crear l'objecte que correspongui a l'acció que s'ha determinat. En la creació i execució del Action s'afegeixen dos components més, els interceptors i els results.

### 8.3.2 Action

Aquest component incorpora la lògica de l'aplicació. En alguns casos el Model, s'incorpora completament en aquest component sense accedir a capes més baixes degut a les característiques de les aplicacions, en definitiva pel seu baix nivell de complexitat, en contes d'implementar EJB de sessió o d'entitat, a l'Action es pot implementar la lògica de negoci. En aplicacions complexes es recomanable seguir el patró MVC d'una manera més rígida.

Els objectes que fan referència als Action es creen i s'executen a partir d'una petició del client, la implementació habitual és a través de classes de tipus POJO (Plain Old Java Objects). La seva característica principal és que no necessiten incorporar cap tipus de d'interfície o herència. Per facilitar la seva implementació s'utilitza la tecnologia Reflections i Introspection de JAVA afegint la configuració associada pel seu comportament com ara el arxius XML o Annotations.

### 8.3.3 Interceptor

El framework STRUTS 2 incorpora un component per alleugerir certes tasques en el preprocessat i post processat de la informació. Els interceptors son components que es poden executar abans o després de la execució d'un Action. Poden resoldre diferents tasques com ara la validació de dades d'entrada o conversió d'aquestes en un determinat format per poder tractar-les, de la mateixa manera es poden tractar les dades de sortida del Result. Existeix la possibilitat de crear-los a mida afegint versatilitat i potencia al sistema.

### 8.3.4 Result

Després de l'execució d'una acció s'espera un resultat, que poden ser unes dades, la confirmació d'una acció, etc... Els objectes Result son els que modelen aquesta informació, necessariament a l'hora d'exposar-la al client s'ha de preparar, per oferir les dades d'una manera entenedora, les JSP (Java Server Pages) son la eina utilitzada per defecte, encara que STRUTS 2 ofereix suport per altres tecnologies con FreeMarker, Velocity, texte pla, XSLT...

### 8.3.5 TagLib JSP

Per definició un Framework ha de donar eines per poder implementar de forma ràpida certes parts del programari. Les TagLib de STRUTS 2, son una llibreria destinada a la manipulació de les dades de la capa de presentació a través de la utilització d'etiquetes. Utilitzant la tecnologia de referenciat a Tags, STRUTS implementa una eina que és capaç de manipular les dades rebudes o enviades al client. Aquesta eina pretén que el programador es despreocupi de certs aspectes de la presentació.

En molts casos, la confecció d'una llista o un formulari en HTML pot arribar a ser molt complex, amb el sistema d'etiquetes es proporciona una eina capaç de formatar dades, proporcionant etiquetes amb diferents funcionalitats associades que poden inclús arribar a iterar una llista filtrant per determinats paràmetres, oferir dades de sortida endreçades en un formulari, o proporcionar dades d'entrada ja convertides directament al model per no haver de fer cap manipulació.

## *8.4 Sistemes de configuració*

Als punts anteriors s'ha fet menció als diferents sistemes de configuració que proporciona STRUT 2. L'idea de la configuració per aquesta versió del framework era configuració cero, o sigui no haver complimentar configuració extra a arxius XML. Cal remarcar que aquesta idea és inassequible degut a la complexitat que rau en les aplicacions basades en J2EE i les moltes necessitats que requereix la capa de presentació durant la seva implementació, a la disparitat de dades, conversions, diferents presentacions, sempre s'acaba per necessitar algun tipus de configuració.

### 8.4.1 Configuració per arxius XML

Està compost bàsicament per dos arxius el web.xml i el struts.xml. El primer correspon al descriptor i s'indica quin serà el punt d'accés que tal i com podem deduir serà el FilterDispatcher tractat amb anterioritat, amb aquesta configuració inicial n'hi haurà prou per que el contenidor Web s'assabenti d'on ha de dirigir les peticions.

En segon terme trobem l'arxiu struts.xml que serà el que mantingui la configuració dels elements del framework, encara que aquest pot indicar que la configuració es descompongui en altre arxius. Tindrà en conte aspectes com la ruta per trobar les Action, quins han de ser els Result d'un Action determinat, etc... Les configuracions a implementar es basaran en els diferents Tags que es defineixen al DTD del struts.xml.

### 8.4.2 Configuració per convenció de noms

Aquesta llibreria es basa en definir el comportament del diferents elements que integren el sistema basant-se en els noms que els defineixen. De la mateixa manera que a la configuració per XML es defineix quina és la ruta d'un Action determinat, aplicant aquest tipus de configuració, el simple fet de que una classe implementi la interfície Action atendrà a les peticions amb el nom de la classe, per exemple, una classe anomenada "Test" atendrà a les peticions amb la URI "/Test". D'aquesta mateixa manera, donant els mateixos noms a plantilles i Result el sistema podrà interpretar el seu comportament.

### 8.4.3 Configuració per Annotations

Aquest tipus de configuració utilitza la tecnologia Annotation de JAVA, que correspon a la utilització d'etiquetes amb el distintiu "@" col·locat al davant. Aquesta pràctica aporta informació del programa sense que formi part d'aquest i sense tenir efecte directe. La llibreria "java.lang.annotation.*" és la que implementa l'eina per la seva utilització.

L'aplicació a STRUTS 2 aporta versatilitat al sistema, ja que les configuracions no cal fer-les en un arxiu XML apart, encara que s'hauran de seguir una determinada convenció de noms en les Action per que el sistema funcioni. En el cas de les Action, caldrà la annotation @Action per indicar que una classe correspon a aquest tipus, amb la annotation @Result la sortida del Action, o indicar quins interceptors s'ha d'activar amb el annotation @InterceptorRef.

Aquestes etiquetes també poden tenir paràmetres per refinar el seu comportament, com en el cas de que desitgem que una determinada Action atengui a una URI que no és la que s'assigna per convenció de noms @Action("/uri") o la definició del nom del interceptor que s'ha d'activar @InterceptorRefs("inter1"), també es pot determinar el comportament d'un Result @Result(name="failure", location=fail.jsp).

# 9 Spring Web MVC Framework

## *9.1 Introducció*

Spring Web MVC Framework, és par integrant del framework Spring, hores d'ara per la versió 3.1, un dels més utilitzats per aplicacions J2EE. Spring MVC ha estat creat per donar suport a la creació de la capa Web, oferint una eina als desenvolupadors que faciliti la seva confecció.

Tal com expressa la introducció de la documentació de referència d'Spring, els patrons i bones pràctiques no deixen de esser una referència pel programador, deixant-lo sol davant del desenvolupament d'una aplicació. La creació del framework respon a la necessitat del reaprofitament de l'experiència, oferint eines reals i aplicables al problemes quotidians durant la implementació d'una aplicació.

De la mateixa manera que d'altres paquets inclosos a Spring, el framework Spring MVC, ofereix les classes, objectes i serveis que composen una aplicació Web, permetent que el programador pugui abstreure's de certes parts del programari i facilitant la seva tasca de desenvolupament.

## *9.2 Arquitectura*

Seguint en el patró MVC, es basa en un servlet central que gestiona les peticions i les envia a capes més baixes, a un controlador, per la seva execució. Aquest servlet central és el DispatcherServlet. A la següent diagrama es pot veure el flux que genera una petició:

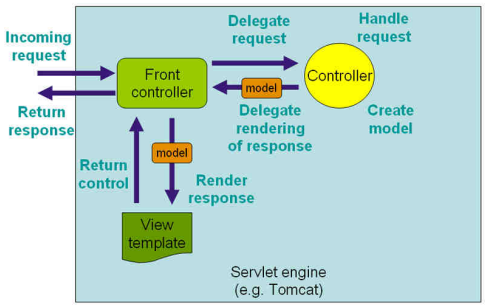



Fig. Flux d'una petició

Com es pot deduir el FrontController correspon al DispatcherServlet que hereta de la classe base HttpServlet, i es declara a l'arxiu web.xml que defineix el punt d'entrada de les peticions. En rebre un petició analitza la URI per poder dirigir la petició cap al controlador que la gestiona. A cada DispatcherServlet l'hi correspon un contenidor de tipus WebApplicationContext que tindrà tot el relacionat amb la nostra aplicació Web.

En rebre una petició d'un client el DispatcherServlet a través del WebApplicationContext analitza la URI per encaminar la petició cap al Controller o el element que pertoqui. El Controller gestiona la petició i retornant la resposta amb el resultat, al següent pas es procedeix a esbrinar la ruta de la vista proposa com a sortida pel Controller a través del ViewResolver, una vegada s'obté la vista a través del View es renderitza i es produeix una resposta cap al client.

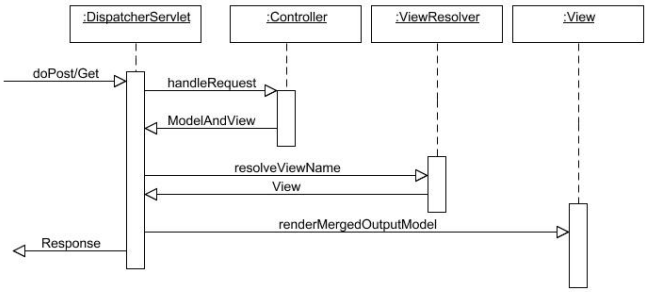



Fig Diagrama de seqüència d'una petició

## *9.3 Elements principals*

Per entendre els elements del sistema Spring MVC, s'ha de tenir en conte que per la configuració del sistema es pot utilitzar la tecnologia Annotations de JAVA per definir el comportament dels elements que el composen. Encara que també es poden utilitzar arxius XML per realitzar la configuració, al document de referència d'aquesta tecnologia, s'aplica la tecnologia de Annotations com a eix central.

Aquesta tecnologia de configuració permet no haver d'estendre classes base específiques ni tampoc implementar cap tipus d'interfície. Tot això provoca usualment que no es creïn dependències sobre altres paquets o API's, oferint una manera fàcil i àgil a l'hora confeccionar una aplicació. Per tant a l'hora de parlar dels elements del sistema estarem parlant del diferents tipus d'Annotations que incorpora Spring MVC.

La utilització d'aquesta tecnologia rau sota el concepte de configuració cero. Però com sempre, difícilment es pot arribar a assolir aquest objectiu. Més endavant, podrem comprovar que sempre es requereix d'una petita aportació als arxius de configuració de Spring MVC com el spring-context per poder acabar de definir el comportament dels elements que hi intervenen.

### 9.3.1 @Controller

Aquesta Annotation és la peça bàsic dins Spring MVC. Ofereixen el punt d'entrada a l'aplicació, interpretant les peticions de l'usuari i la transformant-la en un model que serà presentat a l'usuari a través d'una vista. Spring MVC implementa aquest element d'una manera abstracta que possibilita la creació d'una gama molt amplia de solucions.

Amb aquesta Annotation a la capçalera d'una classe s'indica el rol de controlador, el dispatcher escaneja les classes cercant les Annotations per mapar els mètodes que la composen i detectar altres Annotations relacionades com ara el @RequestMapping. Per realitzar la detecció automàtica s'haurà d'habilitar aquesta característica al arxiu de configuració spring-context.

Com s'ha dit, en contes de la utilització de la tecnologia de Annotations per definir les classes que tenen el rol de Controller, també es pot utilitzar la forma estàndard a través dels arxius de configuració definint de forma explicita aquestes classes. Amb la resta de elements del sistema també succeeix el mateix i no caldrà repetir-ho en les successives explicacions.

### 9.3.2 @RequestMapping

Les respostes a les peticions d'usuaris, es fan a través de classes o mètodes que criden i incorporen la informació a pàgines Web, fitxers JSP, o amb altres tipus d'elements de tecnologies importades. Per poder encaminar la resposta de forma correcta a on correspongui, és necessari fer un mapatge que relaciona la petició que realitza l'usuari amb la resposta adequada.

El RequestMapping és l'anotació que ens permetrà definir aquesta relació, tan si es fa per una classe complerta o per cada mètode. A través d'una indicació amb una codificació d'URL dirigirem la petició cap a la resposta. La forma es directa i no requereix d'altres configuracions auxiliar seguint el concepte de la tecnologia Annotation. A la versió 3.1 s'han millorat en alguns aspectes, però en línees generals el objectiu continua sent el mateix.

La implementació es pot realitzar de diverses maneres, de forma directa, amb el camí complert definit en la URL, de forma relativa a la indicació de la capçalera de la classe o del mètode, aplicant variable utilitzant @PathVariable, inclús aplicant expressions regulars. Els únics paràmetres que accepta son el RequestMethod. GET i el RequestMethod. GET, indicant quin serà el tipus de petició que atendrà.

Existeixen una gran varietat de Annotations relacionades amb el @RequestMapping, com el @ModelAttrribute per afegir atributs al model, @SessionAtributes per mantenir atributs de sessió, @RequestHeader per recuperar la informació de la capçalera, @CookieValue per manipular galetes, entre d'altres.

### 9.3.3 HandlerInterceptor

Com altres frameworks, Spring MVC també incorpora interceptors per fer un preprocessat o un potsprocessat de les peticions. Amb la implementació de la interfície HandlerInterceptorAdapter en una classe, proporcionem l'assequibilitat al mètode preHandle() pel preprocessat i amb el mètode portHandle() potprocessat.

En aquestos mètodes s'implementarà la part de codi relativa a les accions a prendre abans i després de l'execució d'un controlador. Per poder utilitzar aquesta funcionalitat serà necessari realitzar una configuració al apartat del handleMapping per identificar quin son el interceptors de l'aplicació, i la seva definició en quan a variables i la URL per poder trobar-los.

### 9.3.4 ViewResolver

Spring MVC proporciona una eina per poder adreçar-se a les diferents vistes que composen una aplicació d'una forma senzilla, el que es denomina un "resolver" de vistes. Aquesta funcionalitat es defineix a partir de la implementació de la interfície ViewResolver proporcionant una forma de mapar el nom de les vistes i la vista actual, i una segona interfície View per preparar la petició i la manipulació d'aquesta.

La resposta que proporciona un controlador després de ser executat te a veure amb una vista amb el seu nom lògic, proporcionat a través d'un String o un model i una vista associada. Per resoldre l'adreça a que corresponen, s'utilitza el ViewResolver a través d'un seguit de mètodes que proporcionen diferents funcionalitats en quan al tipus de vista que implementada.

La forma més senzilla d'utilització és a través del mètode UrlBasedViewResolver, Ofereix una traducció lògica entre el nom de la petició i la vista associada sense necessitat de definir una mapatge específic ja que s'aplica la regla que es configuri al apartat del viewResolver. Existeixen altres mètodes per la utilització de tecnologies com ara Velocity o FreeMarker que acompanyen a Spring MVC.

### 9.3.5 Altres elements destacables

A part del elements que hem citat amb anterioritat que creiem imprescindibles per la creació d'una aplicació incorporant Spring MVC, aquesta eina proporciona moltes altres funcionalitats destacables, com ara el control d'excepcions, suport pel control de la descarrega d'arxius, control local per determinar diferents perfils de l'aplicació en funció de determinats trets com la ubicació del client, aplicació d'etiquetes ETag.

També cal destacar que Spring MVC utilitza internament el frameworks FreeMarker i Velocity incorporant la seva funcionalitat i ampliant les seves característiques. Però si en qualsevol cas es requereixen altres frameworks externs, existeixen mecanismes per tal d'incorporar-los sense que hagi de ser una acció traumàtica pel programador. En definitiva sempre es cerca la comoditat durant la implementació.

## *9.4 Configuració*

Com s'ha vist a les explicacions anteriors hi ha dos maneres d'afrontar la configuració d'aquest framework, a través de fitxers XML seguint esquemes predefinits on s'especifiquen els diferents components del sistema i a través de la tecnologia de Annotations de JAVA. A més sempre tindrem una configuració pel punt d'entrada del servlet a través del fitxer web.xml comú a les totes les aplicacions integrades en J2EE.

Per una aplicació senzilla, probablement no es requereixin més configuracions que el punt d'entrada del servlet i les Annotations al programa, però quan les aplicacions exigeixen més complexitat es farà indispensable complementar la configuració a través dels fitxers XML. No hi ha cap recomanació especifica que indiqui quina és la manera adequada d'actuar.

Per acabar, encara que no hi hagi una recomanació , la millor manera d'aprofitar la al màxim potencia que proporciona el framework, és aplicar un sistema híbrid en que les dues tecnologies de configuració conviuen. Spring MVC te mecanismes pels quals pot analitzar les configuracions dels fitxers XML i també escanejar el codi cercant Annotations que també incorporin configuracions als components.

# 10 Java Server Faces

## *10.1 Introducció*

L'objectiu de JSF pretén facilitar la construcció d'aplicacions que proporcionin un entorn de treball via Web, gestionant les accions produïdes per l'usuari traduint-les en esdeveniments enviats al servidor amb la finalitat d'oferir els resultats que provoquen aquestes accions. Aquesta eina es basa en esdeveniments provocats per les peticions del usuari a diferència d'altres semblants basades en petició-resposta.

La tecnologia JSF constitueix un framework d'interfícies d'usuari pel cantó del servidor per aplicacions Web. El patró MVC és present en aquest estil arquitectònic, amb una API i una implementació de referència per representar els components UI (User Interface) i gestionar el seu estat d'una forma neta oferint una clara separació entre capes.

A més, ofereix serveis de control d'esdeveniments, validadors i conversors de dades. També incorpora un sistema de configuració per la navegació entre pàgines, la gestió local y una llibreria d'etiquetes TagLib per poder implementar les vistes d'una forma més ràpida i còmoda.

En la última versió 2.0, l'equip de desenvolupament de JSF s'ha adaptat a la nova tendència en la utilització de la tecnologia Annotations de JAVA, proporcionant etiquetes per estalviar en la configuració a través de fitxers XML. Aquesta última versió també incorpora Facelets introduint avantatges en la creació de pàgines Web i el control del cicle de vida del components encara que no implementa completament les funcionalitats de J2EE 6.

Per últim cal dir que JSF també és una especificació estàndard basada principalment en el JSR127, JSR252, JSR276, JSR314, també dona suport a d'altres. Això permet que es puguin trobar implementacions de diferents fabricants, donant l'oportunitat d'escollir una distribució depenent del servei, del tipus de components que ofereix, rendiment, etc... també existeix la possibilitat de implementació de nous components per part de desenvolupador flexibilitzant encara més les capacitats d'aquesta eina.

## *10.2 Arquitectura*

La orientació al component és el tret característic d'aquesta arquitectura, l'usuari interactuarà a través dels components instal·lats pel servidor, generant esdeveniments que els components reben com peticions generant una resposta a través de la transformació de la interfície utilitzant Beans per produir aquesta transformació.

Com els altres casos exposats s'utilitza el patró FrontController com a punt d'accés a través del FacesServlet, en rebre una petició la comunica al component que correspongui, això es gracies a que el servidor manté una estructura interna de components per dirigir tant les peticions com les respostes.

Tal com es veu a la següent figura, després d'una petició s'inicia el procés de reconstrucció de l'arbre de components, s'apliquen les dades rebudes als components que correspongui, seguidament passa per un procés de validació, s'actualitza el model de dades i s'invoca l'aplicació per després renderitzar la resposta. Entre mig de cada fase poden haver esdeveniments que modifiquin el tipus de resposta esperada, com ara un error en el procés de validació de les dades.

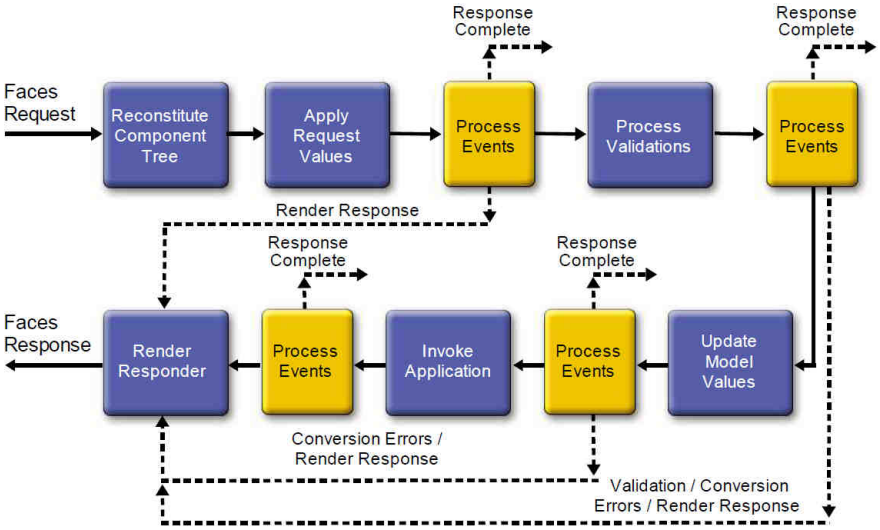



Fig. Procés d'una petició

## *10.3 Elements principals*

Vist el cicle de vida d'una petició d'usuari, es pot entendre que durant el procés de confeccionar la resposta intervenen diversos components. Començant per punt d'entrada a l'aplicació que proporciona el FacesServlet, passant per la interacció que ens permeten els UI Components basat en l'etiquetatge de les funcions en els JSP, la gestió de l'aplicació que ve donada pels Beans i finalment la navegació i renderització de la sortida.

### 10.3.1 FacesServlet

Com a punt comú de configuració en el frameworks de la capa de presentació que han d'interactuar amb J2EE, s'ha de definir quin serà el servlet d'entrada que manegarà les peticions, això sempre es fa configurant a l'arxiu web.xml el nom del servlet i la classe a que fa referència, en aquest cas FacesServlet.

### 10.3.2 UI Components

Son elements configurables i reutilitzables basats en etiquetes que composen la interfície d'usuari JSF. Un component pot ser simple com un botó per interactuar amb l'aplicació o pot tenir diversos elements integrats com una taula amb diferents components. JFS proporciona un conjunt de classes del tipus UI Components, son completament extensibles, per tant, podrem crear components personalitzats, podent combinar-los entre ells per crear elements més complexos.

El conjunt de classes existent a JSF cridades a través d'etiquetes a les JSP, proporcionen diferents funcionalitats, des de el UICommand per disparar les accions quan succeeix el esdeveniment, el UIFom per encapsular un grup de controladors i enviar dades de l'aplicació, UIInput per les dades d'entrada de l'usuari, UIOutput per mostra la sortida de dades en una pàgina, etc... Aquesta particularitat facilita la tasca d'implementació dels controls en la capa de presentació als JSP.

### 10.3.3 Managed Beans

Aquests components son els encarregats de modelar el comportament de l'aplicació. Tal com hem dit, les peticions es tornen esdeveniments que venen adreçats a través dels UI Components, per tal que els Beans portin a terme la tasca que tinguin definida. En aquest components s'implementen la funcionalitat de l'aplicació.

Seguint el patró MVC, aquest components corresponen a la lògica de negoci, concebuts per provocar la separació entre capes. Des de la capa de presentació l'accés als mètodes i als atributs es fa possible a través del IU Components que faciliten l'acoblament entre capes.

### 10.3.4 Navegació

Per tal de poder controlar la navegació entre pàgines, JSF utilitza regles de navegació, que especifiquin com anar d'una pàgina a una altra. Aquestes regles estan contingudes en l'arxiu de configuració faces-config.xml, on es determina quin és el flux de les pàgines depenent de la interacció de l'usuari amb l'aplicació.

Existeixen dos tipus diferenciats de navegació, la estàtica on les accions desencadenades per un esdeveniments, per exemple una pulsació d'un botó, han de concordar amb la definició del arxiu de configuració. I la dinàmica, on la navegació no dependrà del botó que s'hagi accionat, si no per exemple de les dades que un usuari pot haver inserit en un formulari, que faran que la següent pàgina sigui el resultat de l'anàlisi de les dades rebudes.

### 10.3.5 TagLib

Aquest element és una peça bàsica en el JSF, degut a que la utilització d'etiquetes en la confecció de les JSP és indispensable. Existeixen dos tipus d'etiquetes diferenciats, les de tipus "core" que s'utilitzen entre d'altres coses per manegar esdeveniments, atributs, conversors de dades, validació de dades etc... identificades amb el prefixe "f".

Les etiquetes HTML identificades amb el prefix "h" son les més nombroses i s'utilitzen bàsicament per la confecció de la interfície d'usuari. Es poden agrupar en diferents categories, entrada, sortida, comandaments, selecció, panells, taules, errors i missatges. La riquesa d'aquesta funcionalitat és el que aporta senzillesa a la implementació d'aplicacions amb JSF.

### 10.3.6 Converters Listeners i Validators

Aquests elements faciliten les tasques de manipulació de dades, en el moment que es detecta un esdeveniment es pot configurar el sistema per que reaccioni i interposi un conversor o un validador de dades. Es fica a disposició del desenvolupador tota una llibreria per facilitar la realització aquestes tasques, que com sempre poden ser ampliades amb d'altres de creació pròpia.

## *10.4 Configuració*

Durant les explicacions s'ha fet referència a la configuració de JSF a través de fitxers XML, encara que existeix una nova alternativa per tal de reduir aquest tipus de configuració i que el desenvolupament sigui més directe. En la última versió de JSF s'ha implementat la nova tendència en la utilització de la tecnologia Annotations de JAVA.

### 10.4.1 Configuració per arxius XML

La majoria de la literatura sobre JSF ofereix les explicacions del sistema sobre aquest tipus de configuració que és el primitiu de les primeres versions. Es basa en l'arxiu faces-config.xml amb un esquema definit a través d'un XSD, on es defineixen un seguit d'etiquetes per tal de configurar tots els elements del sistema.

Per poder facilitar la configuració s'ofereix l'alternativa de poder dividir-la en diferents arxius, per tal de poder estructurar-la configuració amb l'objectiu que les aplicacions amb més necessitats no quedin condensades en un sol lloc i es pugui tractar de forma individual seccions determinades de l'aplicatiu.

### 10.4.2 Configuració per Annotations

De la mateixa manera que a altres frameworks, a JSF es poden inserir Annotations al codi per tal de definir el comportament de l'aplicació, s'han aplicat aquest tipus d'elements a components com per exemple als Beans, en contes de identificar-los a través del arxiu de configuració es pot aplicar la directiva @ManagedBean per expressar aquest rol en una classe.

També s'aplica en elements concrets com per exemple als Bean Validation Model, on existeixen tot un seguit de Annotations per tal de controlar el mètode de validació sense cap configuració afegida que solucionen la necessitat de codificar la funcionalitat en la classe.

Existeixen d'altres Annotations per els diferents components de JSF que també redueixen la necessitat de configuració. Un dels aspectes en que s'acostuma a aplicar més configuració és a la navegació, en canvi en aquesta versió de JSF no he trobat cap referència al respecte, complicant la implementació en aquest aspecte.

# 11 Conclusions dels Frameworks de l'estudi

Els tres frameworks analitzats tenen com a objectiu alleugerir les tasques de implementació de la capa de presentació. Oferint eines tan per accedir a les dades de la Web, com per proporcionant-les, pel seu pre i pots processat, solucionant feines com la navegació, el control de l'aplicació i la programació de les pàgines Web entre d'altres. I en els tres casos, cadascun a la seva manera, proporcionen una solució. Difícilment es pot afirmar quina és la millor, això o deixem al judici dels programadors, segurament per cada projecte l'eina adequada serà una de diferent.

Moltes vegades la selecció d'aquesta eina rau en el coneixement que es te d'ella, per tant, serà més fàcil que un programador es decideixi per una en concret degut a estar més experimentat en haver-la utilitzat abans, que no pas per que aporti algun tipus de funcionalitat especial. Amb això volem dir que en major o menor mesura, aquest tipus de framework ofereix solucions comuns d'una manera o una altra, i que els programadors poden realitzar qualsevol desenvolupament amb totes elles.

En tots els casos les tecnologies aplicades son molt semblants, poden estar més evolucionades com son Struts2 i Spring, que en la seva última versió han adoptat tecnologies de nova collita per se implementades amb l'objectiu d'assolir una millora substancial. En el cas de JSF, la seva evolució no és tant evident, encara que incorpora tecnologies semblants als dos anteriors, dona la impressió de no estar tant acabat o ser tan complert. Queda patent que les solucions que ofereixen tots tres frameworks parteixen de l'experiència i les necessitats dels professionals que han contribuït en el seu disseny i la seva implementació des d'un punt de vista funcional.

Un tret comú entre Struts2 i Spring, és la seva capacitat per poder incloure tecnologies alienes en la confecció de la seva sortida, com ara FreeMarker, Velocity o d'altres. Tenint en conte que una gran part de les hores de desenvolupament s'inverteixen en la interfície gràfica de les pàgines Web, incloure aquest tipus de paquets soluciona en gran mesura aquestes tasques, facilitant la confecció i reduint el temps de desenvolupament, a demés de l'enriquiment visual que suposa. El JSF sembla ser que no incorpora directament aquest tipus de solució, per tant és un punt important en contra a l'hora d'escollir aquesta eina, ja que haurem de confeccionar la sortida només amb les eines que implementa per si mateix.

En conclusió, l'esforç més important d'aquest tipus de framework està en l'estalvi de codi en el desenvolupament de la capa de presentació, i en la intenció de configuració cero per facilitar la seva implementació. Ho duen a terme diferents maneres i amb diferents tecnologies per arribar al mateix resultat, la creació de programari de forma efectiva, tant en temps com en resultats. Així doncs, podem concloure que la seva utilització és recomanable, si no en tots els casos, en aquells que l'aplicació per les seves característiques i volum ho demanin.

# 12 El nou framework ***SappFramework***

## *12.1 Introducció*

Després de l'anàlisi d'algunes de les eines que actualment es troben al mercat, es poden comprendre les necessitats dels integradors de la capa de presentació que impulsen el desenvolupament d'una eina que ajudi en les tasques d'aquest nivell del programari. Tot i això, tornem a exposar que una eina d'aquest tipus naix de l'experiència i dels entrebancs que es troben durant el procés de disseny i implementació d'aquesta capa.

En el moment de la creació d'aquest nou framework, s'han fixat uns objectius per tal d'obtenir un producte final que aporti solucions a aquests entrebancs, aquestos objectius son el següents:

- Configuració cero.
- Simplificació d'accés a les dades entre capes WEB <-> Aplicació.
- Estalvi de codi.
- Àgil d'implementar.

Però ha estat durant la posada en marxa del sistema, aplicat a en uns requisits concretes d'un desenvolupament, quan han aparegut els entrebancs que han provocat la millora del framework per poder superar determinats obstacles de la implementació. Aquesta experiència a enriquit el producte final amb solucions reals basades en les necessitats.

Com no, per implementar el codi que composa el nou framework Sapp (Sistema Àgil per la Producció de la capa de Presentació), s'ha invertit una gran quantitat de temps en la recerca de tècniques, llibreries i codi, per ser incorporades a Sapp, per tal d'obtenir un producte sòlid, amb els requeriments necessaris per poder ser implementat amb totes les garanties de funcionament, i amb les funcionalitats que converteixin aquest sistema en una eina útil pels desenvolupadors.

La separació clara entre capes, respectant el patró MVC, ha estat present tant en el procés de disseny com en el desenvolupament. Ha estat un prioritat per tal de que l'aplicació del framework per part d'un programador, no deixi lloc a dubtes a l'hora de decidir on ubicar cada porció de codi. L'objectiu de minimitzar el codi de la capa de presentació ajuda a questa separació, aportant les eines necessàries per oblidar-nos de implementar codi per determinades tasques, com l'accés a les dades tan en la pàgina web com en l'aplicació, el control d'usuaris o les preexecucions i postexecucions d'una acció determinada.

Durant els següents apartats s'explicaran les solucions aplicades per tal d'aconseguir els objectius, així com les funcionalitats que aporta Sapp, de les quals es podran aprofitar els programadors que decideixin la utilització d'aquesta eina com a mitjà per la implementació de la cap de presentació. Amb el codi públic es deixa la porta oberta per la millora i la incorporació d'eines que ampliïn les seves funcionalitats.

## *12.2 Arquitectura de SappFramework*

Les opcions que ofereix la tecnologia JAVA i J2EE a l'hora d'implementar un framework d'aquest estil, son moltes i variades. En alguns casos les decisions s'han pres de forma arbitraria, i en altres casos com a conseqüència d'exigències de la tecnologia en que es basa el sistema. Però en tot cas, també s'ha de tenir en conte el rendiment a l'hora de la seva aplicació en un cas real. Els sistemes complexos i pensants requereixen d'un maquinari potent i per tan molt més car. Tenint en conte aquestos conceptes exposem a continuació l'arquitectura del sistema Sapp.

La descripció d'una arquitectura complexa i el seu funcionament, com és el cas del sappFramework, moltes vegades es torna complicada i pot ser motiu de confusió pels programadors que l'utilitzen. S'ha fet un esforç especial per generar documentació en "JavaDoc", per tal de que durant el procés d'implementació del sistema es tingui una guia útil que aporti informació en el moment de la utilització de les funcionalitats que incorpora. Per tant és més que recomanable fer us de la informació que aporta el "JavaDoc" en cada moment.

### 12.2.1 Tecnologia Servlet com a mitjà d'accés i control

Per garantir l'accés des de la capa d'usuari fins a la capa d'aplicació s'ha optat per la tecnologia servlet, incorporant les llibreries "javax.Servlet.*". Aquesta llibreria és prou difosa i coneguda per tal que no resulti un problema pels desenvolupadors en cas de voler ampliar les funcionalitats del framework. Les característiques que implementa aquesta llibreria proporcionen les eines necessaris per tal d'assolir una configuració senzilla al nostre framework, la manipulació de les dades rebudes i enviades, així com el control de les sessions en curs.

El descriptor de desplegament representat per l'arxiu "web.xml", sempre present en la tecnologia de servlets, facilita la configuració del sistema i ens proporciona la porta d'entrada al framework. Ofereix la possibilitat d'incorporar certes funcionalitats que té el framework Sapp. Per tal d'aproximar-nos al màxim a l'objectiu de configuració cero, no s'ha abusat d'aquest arxiu, deixant determinades funcionalitats obertes per que l'integrador les incorpori si ho troba útil pel seu desenvolupament. Per cobrir la necessitat bàsica d'accés només caldrà configurar, la referència a SappFramework i la ruta de l'aplicació on trobarem els controladors.

El SappFramework utilitza el sistema servlet com a mitjà des del qual es detecta i es configura l'aplicació en el moment de l'arrencada. Una vegada configurat i en servei, la seva tasca serà la de transportar i mantenir les dades que intervenen en l'aplicació, fent el manteniment del controladors, atenent a les peticions que es fan des de la Web i proporcionant les respostes a través dels objectes que intervenen en l'aplicació implementada pel desenvolupador.

### 12.2.2 Llibreries Google Reflections

La reutilització del codi és peça fonamental en la programació, per tal d'obtenir bons resultats tan en la durada de la implementació d'un projecte, com en les garanties de funcionament d'aquest. Aquesta llibreria s'ha incorporat per tal de facilitar determinades les tasques d'anàlisi dels controladors durant el procés d'arrencada del Sapp. Ens permet l'anàlisi en temps d'execució de les característiques assignades pel desenvolupador de les classes que intervenen en una aplicació, reflectides per les "Annotations" que incorporen al seu codi.

La peça clau pel funcionament d'aquest nou framework és la tecnologia "Annotations" de java. Aquesta informació afegida al codi, que no forma part del programa i que no te incidència directa sobre ell, aporta informació extra en temps d'execució sobre l'aplicació mateixa. Aquesta informació extra no es pot avaluar directament en el programa en execució, cal fer un anàlisi anterior per avaluar el que s'especifica en les "Annotations".

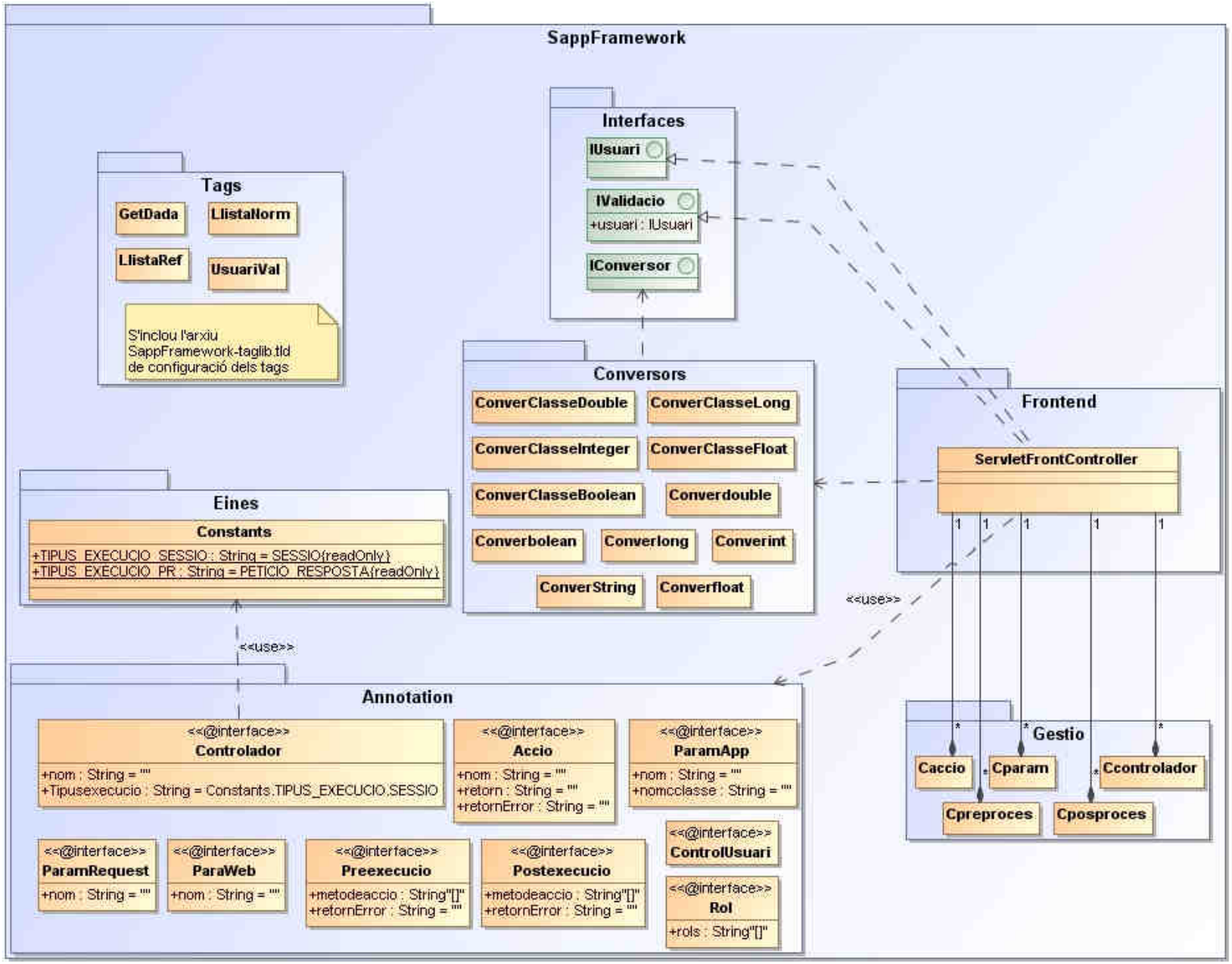
Aquestes anotacions extra, codificades amb el símbol "@" abans de cadascuna d'elles, han d'estar prèviament definides en una classe que implementi l'anotació "@Interface", on es codificarà la informació que aportarà una determinada anotació, cadascun dels seus atributs afegirà una determinada informació que podrà ser avaluada en temps d'execució. És en aquest moment on entra en joc la llibreria Google Reflections, que ens proporciona les eines per tal de poder extreure la informació de les "Annotations" del codi permetent redefinir el comportament d'una part de codi.

La tecnologia Java Annotation també ens permet construir codi per poder analitzar les anotacions, però s'ha de crear des de cero, Google Reflections ja incorpora determinades classes i mètodes per tal de facilitar les tasques d'obtenció d'anotacions d'una forma segura i efectiva, és per aquest motiu pel qual s'ha cregut convenient la seva utilització, estalviant codi i garantint el funcionament. L'únic inconvenient que s'ha trobat és que s'han hagut d'incloure altres llibreries de les quals Google Reflections depèn (guava, javassist, i slf4j).

### 12.2.3 Estructura i diagrama de classes

L'estructura de SappFramework te un component central que gestiona el comportament del framework, des de la seva inicialització fins a l'execució de l'aplicació, la classe ServletFrontController . Aquesta classe depèn de diferents paquets on s'organitzen diferents elements per la gestió del framework.

Al següent diagrama de classes es mostra d'una manera simplificada els elements que composen el framework, dels quals es descriuran més endavant en el transcurs de tota l'explicació de funcionament del SappFramework.


SappFramework
Interfaces
IUsuari
IValidacio
+usuari : IUsuari
IConversor
Tags
GetDada
LlistaNorm
LlistaRef
UsuariVal
S'inclou l'arxiu SappFramework-taglib.tld de configuració dels tags
Conversors
ConverClasseDouble
ConverClasseLong
ConverClasseInteger
ConverClasseFloat
ConverClasseBoolean
Converdouble
Converbolean
Converlong
Converint
ConverString
Converfloat
Frontend
ServletFrontController
Eines
Constants
+TIPUS_EXECUCIO_SESSIO : String = SESSIO{readOnly}
+TIPUS_EXECUCIO_PR : String = PETICIO_RESPOSTA{readOnly}
<<use>>
<<use>>
Annotation
<<@interface>> Controlador
+nom : String = ""
+Tipusexecucio : String = Constants.TIPUS_EXECUCIO.SESSIO
<<@interface>> Accio
+nom : String = ""
+retorn : String = ""
+retornError : String = ""
<<@interface>> ParamApp
+nom : String = ""
+nomcclasse : String = ""
<<@interface>> ParamRequest
+nom : String = ""
<<@interface>> ParaWeb
+nom : String = ""
<<@interface>> Preexecucio
+metodeaccio : String"[]"
+retornError : String = ""
<<@interface>> Postexecucio
+metodeaccio : String"[]"
+retornError : String = ""
<<@interface>> ControlUsuari
<<@interface>> Rol
+rols : String"[]"
Gestio
Caccio
Cparam
Ccontrolador
Cpreproces
Cposproces

#### 12.2.3.1 Paquet Annotation

S'ha destinat aquest paquet a l'agrupació de les anotacions que incorpora el framework. Cada una de les classes correspon a una possible anotació que espot utilitzar durant el procés de confecció d'una aplicació. Aquestes anotacions es descriurà amb detall en l'apartat de configuració i implementació.

#### 12.2.3.2 Paquet Eines

Inicialment es va crear aquest paquet per incorporar eines d'ús comú a tot el framework. Pel tipus de desenvolupament que s'ha dut a terme només ha calgut una classe on s'agrupen les constants que es fan servir. En cas que en un futur es necessiti afegir qualsevol altra eina, es podrà ubicar en aquest espai.

#### 12.2.3.3 Paquet Conversors

En aquest paquet poden trobar les classes utilitzades per ver la conversió de les dades que provenen de la web, o sigui d'un "String" a qualsevol altre tipus que podem trobar a les classes del paquet. S'ha de puntualitzar que el tipus d'una variable s'esbrina a través del tipus retornat del mètode "get" del atribut del mateix nom del controlador que correspongui.

#### 12.2.3.4 Paquet Interfaces

Al nostre framework s'utilitzen 3 interfícies agrupades en aquest paquet. La interfície "IConversor" és la utilitzada per dona suport les conversions de dades i la implementen totes les classes del paquet "Conversors". Les interfícies "IUsuari" i "IValidacio" son utilitzades pel framework per oferir la funcionalitat de control del rol de l'usuari, en cas de que es vulgui fer aquesta gestió a través del SappFramework.

#### 12.2.3.5 Paquet Gestió

Per poder modelar la informació dels controladors, les accions i els seus paràmetres, els mètodes de preexecució i els de postexecució i els seus paràmetres, s'han implementat les classes d'aquest paquet. A través de les anotacions que s'inclouen a una aplicació basada en el SappFramework, es guardaran les dades i configuracions de cadascun dels elements.

#### 12.2.3.6 Paquet Frontend

Aquest paquet és la peça central que uneix la resta de paquets, només conté la classe ServletFrontControler, que gestionarà la inicialització del sistema i la seva execució.

#### 12.2.3.7 Paquet Tags

S'han desenvolupat diverses eines, sobretot com a exemple, per accedir de forma fàcil a les dades que el framework. Aquest paquet conté aquests tags.

### 12.2.4 Conversió de dades

Un dels punts importants en un framework que ha de rebre dades des d'una Web, és la conversió de les dades rebudes com una cadena "String", al format utilitzat durant l'execució de l'aplicació. Per aquest motiu s'ha incorporat un automatisme per que aquesta tasca es pugui fer cap esforç. A través de l'anotació @ParamWeb inserida abans de cada paràmetre d'un mètode de tipus "@Accio" es defineix el nom dels paràmetres que es rebran des de el web, el Sappframework ha d'identificar el paràmetre que correspon i el seu tipus.

Caldrà respectar uns requisits per que el sistema funcioni correctament:

- El nom de la variable ha de coincidir amb el nom utilitzat a l'aplicació, o sigui, a d'existir un atribut al controlador amb el mateix nom.
- L'atribut del controlador ha de tenir implementats els mètodes d'accés "get" i "set".

El primer requisit és necessari per poder trobar la variable al controlador, si el nom no coincideix no el podrà identificar. El segon requisit és necessari degut a que per esbrinar el tipus no es pot fer a través dels paràmetres del mètode, ja que amb el polimorfisme de la programació OO poden haver dos mètodes amb el mateix nom i diferents paràmetres, així dons, serà a través del tipus de retorn de mètode "get" de l'atribut de la classe controlador que podrem esbrinar quin ha de ser la classe del paquet "Conversors" que s'ha d'utilitzar. Aquest paquet disposa de dues classes per cada tipus ja que tenim tipus primitius i tipus Classe i s'han de poder convertir tots dos.

### 12.2.5 Concepte de controlador i acció

Abans de continuar amb explicacions més detalles, cal definir don senzills conceptes, el de controlador i el de acció.

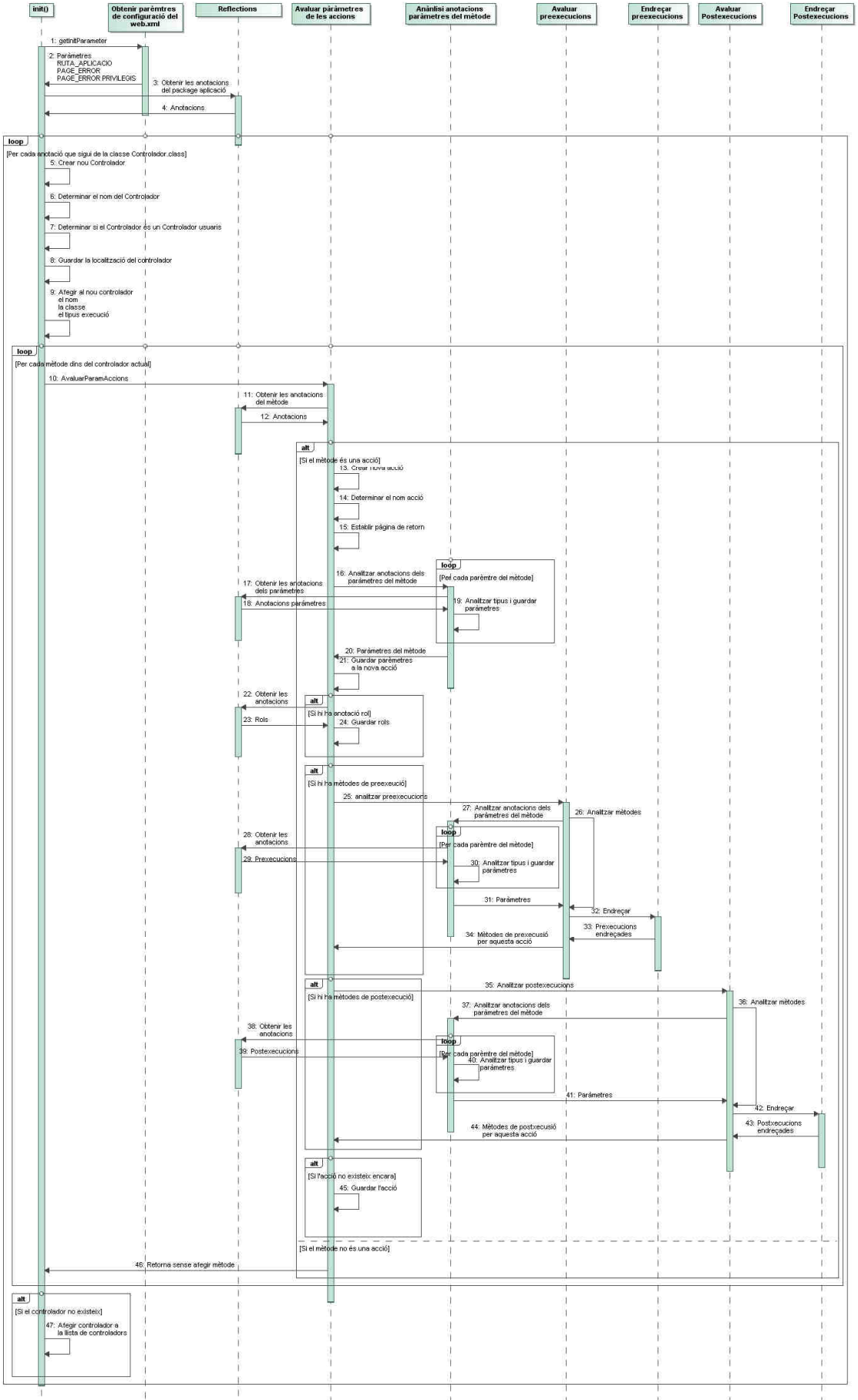
Un controlador és una classe que es defineix d'aquest tipus a través de l'anotació "@Controlador". El framework en iniciar-se primerament cercarà a la ruta base de l'aplicació classes que incorporin aquesta anotació, i les analitzarà per tal de trobar mètodes amb l'anotació "@Accio", que representen accions accessibles des de la Web a través del seu nom per executar els mètodes que representen.

Els controladors son una manera de passar dades capa la capa Web, ja que son instanciats al inici de la crida d'una acció, i després d'haver executat aquesta, es passa l'objecte Controlador amb tots els seus atributs a la capa Web, sempre a través d'un **"HashMap" amb el nom "Sapp"** que contindrà tots els objectes de tipus controladors que s'han instanciat durant la sessió a través de les crides a accions. D'aquesta manera sempre tindrem les dades actualitzades a la Web sense necessitat programar a la "request" el set dels atributs.

Per tant per accedir a les dades dels controladors només caldrà demanar al "HashMap" "Sapp" pel nom del controlador i retornarà un objecte que el representa, evitant la necessitat de manipular la "request" cada cop que hem d'enviar dades a la Web.

### 12.2.6 Inicialització del SappFramework

En aquesta seqüència s'exposa d'una manera resumida la inicialització del framework:



El diagrama anterior és un petit resum de la seqüència que segueix el SappFramework en el moment de ser inicialitzat, a continuació fem una petita explicació d'aquesta seqüència:

1. En arrencar el SappFramework s'executa el mètode Init().
2. Es llegeixen els paràmetres de la ruta base de l'aplicació, la pàgina d'error general i la pàgina d'error de privilegis.
3. Es llegeixen les anotacions de la ruta base de l'aplicació
4. Es cerquen tots els Controladors
5. Per cada Controlador
    a. Es cerquen totes les seves accions
        i. Per cada acció
            1. S'analitzen els paràmetres de la crida i es guarden
            2. Es cerquen el Rols i es guarden
            3. Es cerquen el mètodes de preexecució i es guarden
            4. Es cerquen el mètodes de postexecució i es guarden
    b. Es guarden les accions del controlador.
6. Es guarda el controlador.
7. Finalitza la inicialització del SappFramework i arrenca l'aplicació.

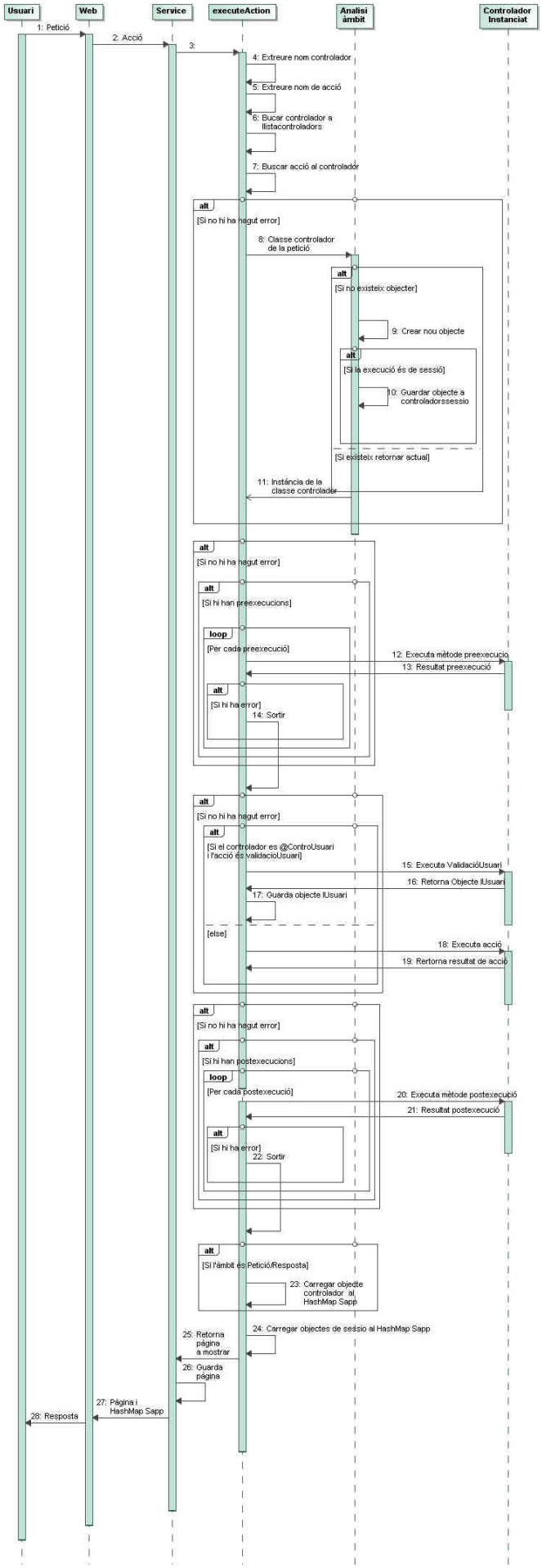
Per guardar cadascun dels elements que es citen, s'utilitzaran les classes del paquet "Gestió" que modelen els atributs necessaris per les diferents configuracions que pot arribar a assolit tant un controlador com una acció. En molt casos els atribut son multiples, com per exemple el nombre d'accions que integra un controladors, per tant es guarden en "array's" per tal de facilitar la seva utilització

Una vegada acabada la inicialització, el resultat és un "HashMap" amb el nom "llistacontroladors", que conté les classes controladores amb els seus atributs d'execució, on l'identificador per poder accedir i crear una instància de la classe controlador és el seu nom. Existeixen altres "HashMap" que recolzen el funcionament del framework que no son d'interès.

### 12.2.7 Tractament d'una petició

Després de la inicialització del framework, aquest queda en disposició de rebre peticions des de la Web. En aquest punt explicarem el procés que es desencadena en rere la petició d'execució d'una acció.

Primer mostrarem un diagrama resumit de la seqüència que segueix una petició que te com a objecte l'execució d'una acció de l'aplicació. En la següent imatge es poden veure els passos que segueix la petició des de el moment que l'usuari la genera fins que l'hi retorna la resposta a la seva petició


Usuari
Web
Service
executeAction
Analisi àmbit
Controlador Instanciat
1: Petició
2: Acció
3:
4: Extreure nom controlador
5: Extreure nom de acció
6: Bucar controlador a llistacontroladors
7: Buscar acció al controlador
alt
[Si no hi ha hagut error]
8: Classe controlador de la petició
alt
[Si no existeix objecter]
9: Crear nou objecte
alt
[Si la execució és de sessió]
10: Guardar objecte a controladorssessio
[Si existeix retornar actual]
11: Instància de la classe controlador
alt
[Si no hi ha hagut error]
alt
[Si hi han preexecucions]
loop
[Per cada preexecució]
12: Executa mètode preexecució
13: Resultat preexecució
alt
[Si hi ha error]
14: Sortir
alt
[Si no hi ha hagut error]
alt
[Si el controlador es @ControUsuari i l'acció és validacioUsuari]
15: Executa ValidacióUsuari
16: Retorna Objecte IUsuari
17: Guarda objecte IUsuari
[else]
18: Executa acció
19: Retorna resultat de acció
alt
[Si no hi ha hagut error]
alt
[Si hi han postexecucions]
loop
[Per cada postexecució]
20: Executa mètode postexecució
21: Resultat postexecució
alt
[Si hi ha error]
22: Sortir
alt
[Si l'àmbit és Petició/Resposta]
23: Carregar objecte controlador al HashMap Sapp
24: Carregar objectes de sessio al HashMap Sapp
25: Retorna pàgina a mostrar
26: Guarda pàgina
27: Pàgina i HashMap Sapp
28: Resposta

El diagrama anterior és un resum de la seqüència que segueix el SappFramework en el moment rebre una petició, a continuació fem una explicació d'aquesta seqüència:

1. L'usuari a través de la pàgina Web selecciona una acció i genera una petició.
2. La pàgina Web en rebre la petició envia l'acció i el controlador associats.
3. S'extreu el nom del controlador de la "Query".
4. S'extreu el nom de l'acció de la "Query".
5. Es busca el controlador a "llistacontroladors"
6. Es busca l'acció al controlador
7. S'analitza l'àmbit de la petició si és de sessió o petició/resposta
    a. Si l'objecte no existeix es crea un de nou.
    b. Si l'execució es de sessió guarda l'objecte a la sessió.
    c. Si l'objecte existeix es retorna el que està en memòria.
8. Es retorna la instancia de la classe controlador.
9. Si hi han mètodes de preexecució s'executen
    a. Si ha error en una preexecució surt
10. S'executa l'acció
    a. Si l'acció es de validació d'usuari rep un tractament especial.
11. Si hi ha mètodes de postexecució s'executen.
    a. Si ha error en una postexecució surt
12. Si l'àmbit del controlador és de petició/resposta carregar l'objecte controlador a "sortidadadesdessessio"
13. Carregar "sortidadadesdessessio" a "HashMap" "Saap"
14. Carregar a la "request" el "HashMap" "Saap".
15. Retorna la pàgina a mostrar
16. A la pàgina Web es rep el "HashMap" "Saap".
17. L'usuari rep la resposta.

Com es pot veure en el diagrama a cada pas s'analitza si el pas anterior ha tingut un error, si la sortida es amb error no s'ha de continuar executant cap més etapa. Al diagrama no es pot mostrar el detall de les pàgines que es carreguen en funció del resultat ok o no ok. Més endavant en l'explicació relativa les diferents anotacions, s'exposa quina és la reacció i la pàgina de sortida per cada cas.

Ja hem parlat amb anterioritat del "HashMap" "Sapp", però és em aquest apartat on es pot apreciar més clarament el seu sentit. Aquest element del framework és un dels principals, ja que a través d'ell es proporciona la informació a la Web que l'aplicació confecciona per se mostrada o manipulada.

Per accedir als atributs d'un controlador només caldrà extreure l'objecte del "HashMap" "Sapp" a través del nom que se l'hi ha assignat, i quedarà a disposició del programador. S'ha de puntualitzar que la utilització de la tecnologia TagLib de Java és molt recomanable per estalvia codi JAVA a les pàgines JSP. Queda a criteri del programador la seva utilització.

### 12.2.8 Control d'errors

El Sappframework incorpora un sistema de control d'errors per tal d'ajudar al programador durant el procés d'implementació del sistema a trobar configuracions errònies o codi mal format, i també a avaluar situacions controlades durant l'execució del sistema, com una entrada de dades no satisfactòria. Així doncs, es poden identificar dos nivells d'errors, el nivell d'error de codificació, i el nivell d'error d'aplicació.

#### 12.2.8.1 Errors de codificació

Com a errors de codificació es defineixen tots aquells que tenen a veure amb el codi desenvolupat a l'aplicació, com per exemple que les anotacions en una aplicació no han estat correctes, falta el "set" o "get" d'un atribut, el nom d'una acció repetit, una preexecució mal formada, o una a la petició d'una acció que no existeix. Aquestos errors es veuran reflectits a la sortida del log del servidor. S'ha utilitzat la llibreria "Java.util.logging.*" per tal de mostrar els errors i utilitzar la jerarquia de missatges que proporciona.

#### 12.2.8.2 Errors d'aplicació

Els errors a nivell d'aplicació son els controlats pel programador, i que poden donar una sortida d'informació a l'usuari, com a exemple tenim el control de dates ben formades, si la data introduïda no és correcta s'hauria d'informar que l'acció no s'ha dut a terme o que cal introduir la data correcta. Per aquesta gestió es dona la possibilitat al programador d'implementar un atribut amb el nom "error" dins d'una classe "Controlador", es tracta d'un String al qual rs pot descriure l'error. No és obligatòria la seva utilització, si no existeix aquest atribut el SappFramework no el gestionarà.

Com a resultat de l'execució d'una acció que no ha estat satisfactòria, o sigui que el booleà de retorn es "true", a més de redirigir la navegació cap a la pàgina d'error, el SappFramework buscarà l'atribut "error" de la classe Controlador de l'acció executada, i el copiarà sobre l'atribut "error" de la "request", d'aquesta manera podrem accedir directament al text de l'error sense haver de cercar-lo a la classe controlador. Per tant, en una pàgina definida com a pàgina d'error, a la qual sempre accedim després d'algun tipus d'error, només caldrà consultar l'atribut "error" de la "request" on trobarem la descripció que ha realitzat el programador, escrivint aquesta descripció en l'atribut "error" en l'acció que l'ha causat.

Existeix un inconvenient, el SappFramework no inicialitza aquest atribut cada cop que executa una acció, per tant, si l'atribut anteriorment queda definit amb un error, i una acció posterior acaba amb error i no es sobreescriu amb l'error que correspongui, es visualitzarà l'anterior. En posteriors versions del framework aquest inconvenient es podrà millorar amb la inicialització automàtica de l'atribut "error" del controlador.

### 12.2.9 Pàgines JSP

Aquest framework es basa amb el tipus de pàgines JSP. Les sortides i configuracions que trobarem al llarg de l'explicació de les anotacions del SappFramework, fan referència a aquest un únic tipus de sortida. S'ha pres aquesta decisió degut a que les pàgines JSP son elements flexibles als qual es pot incorporar codi JAVA que enriqueixen les possibilitats d'implementació. Així doncs a de quedar clar pel programador que la base Web ha d'estar sempre dirigida cap a aquest tipus de pàgines.

### 12.2.10 Atribut "ultimapagina"

S'ha afegit un atribut a la gestió del servlet que es creu cal comentar. Al "ServletContex" es guarda l'atribut "utimapagina" per tal de memoritzar quina ha estat l'última pàgina vàlida que s'ha enviat al servidor Web. Aquest atribut soluciona l'aspecte de navegació per retornar a la pàgina de on procedia l'acció en cas de que aquesta acció no tingui una pàgina configurada com a retorn.

Aquesta funcionalitat facilita la configuració de les accions, ja que en molts casos l'acció ha de mostrar determinades dades en la mateixa pàgina que s'estan demanant. Amb aquesta solució, si es dona aquest cas, no caldrà afegir a l'anotació de l'acció quina és la pàgina de retorn, ja que per defecte navegarà cap a la pàgina de on prové la petició de l'acció.

### 12.2.11 Les TagLib

Aquest nou framework incorpora les eines necessàries per gestionar les dades que es reben i s'envien de i cap la Web d'una manera automatitzada, aquesta funcionalitat es dona a través de la implementació de les anotacions als paràmetres que rep una acció, i com a sortida, la recuperació dels atributs dels controladors que es fa a través del "HashMap" "Sapp".

Arribats a aquest punt, hem de dir que una part important del codi es pot arribar a donar en les pàgines JSP, ja que per recuperar les dades que proporcionen els controladors caldrà accedir a la "request", obligant a fer, importacions de paquets o iteracions i consultes dels objectes. És en aquest punt on entren en joc els TagLib amb l'objectiu de minimitzar el codi a les pàgines JSP. Per aquest motiu, es proporcionen una sèrie de TagLib's, funcionals totes elles, i a mode d'exemple, per que el programador pugui desenvolupar les seves.

Es força difícil esbrinar quines son les necessitats que pot arribar a tenir un programador en el moment de la confecció de les pàgines Web, d'això es pot deduir que no hi ha cap repertori de TagLib's, per nombrós que sigui, que solucioni tota la casuística en aquest àmbit. S'han escollit quatre casos, del més senzill com extreure un atribut d'un controlador o llegir el nom d'un usuari, a més complex com generar una taula que accedeix a diversos atributs de forma selecctiva. En definitiva haurà de ser el programador el que es vagi confeccionant les seves eines i si és el cas compartir el coneixement amb la resta professionals que utilitzen aquest framework com a solució.

### 12.2.12 Opció per la utilització d'un altre framework

Per acabar parlarem d'un mètode que incorpora el SappFramework a la classe "ServletFrontController". Es tracta del mètode "adaptador", aquest mètode es cridat en última instancia abans de sortir de l'execució d'una acció, abans de carregar a la "request" el "HashMap" "Sapp" i abans d'executar el "response". En cas que es vulgui adaptar la sortida a qualsevol altre framework, com per exemple freemarker, en aquest mètode es poden fer les adaptacions pertinents per que la sortida a través de "response" sigui la desitjada.

Cada framework te uns requisits determinats, que segurament requeriran d'una inicialització, durant l'arrencada del "servlet", al mètode "init" queda molt ben definida la secció del SappFramework, i on es pot afegir codi auxiliar, deixant-lo a disposició del programador a l'hora de introduir nou codi. Però la finalització d'una petició pot ser una mica més difícil de localitzar dins el codi del SappFramework, per aquet motiu s'ha implementat aquest mètode, per tal d'ubicar el punt concret on finalitzen les peticions, i per tant, el punt on es pot intervenir per adaptar la sortida obtinguda de la petició al framework.

## *12.3 Configuració i implementació*

### 12.3.1 Servlet i web.xml

Per acomplir l'objectiu de configuració cero s'ha minimitzat la necessitat de complimentar el fitxer de configuració del servlet. Només és necessari indicar el nom que assignat al servlet, la classe que l'implementa, i la pàgina d'inici definició imprescindible en la utilització dels servlets, a més de la ruta dins el projecte en la que es troben els nostres controladors. Això ho farem a través de les etiquetes i paràmetres següents:

```
<servlet-mapping>
  <servlet-name>Sapp</servlet-name>
  <url-pattern>/Sapp/*</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>Sapp</servlet-name>
  <servlet-class>SappFramework.Frontend.ServletFrontController</servlet-class>
  <init-param>
    <param-name>ruta.base</param-name>
    <param-value>meeting.presentation</param-value>
  </init-param>
<servlet>
<welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
```

El paràmetre "ruta.base" indicarà la ruta en la que es trobaran els controladors de l'aplicació, en el cas del nostre exemple el "package" que contindrà els controladors que s'han d'avaluar i gestionar serà "meeting.presentation". El paràmetre "<welcome-file>" indica quina serà la pàgina d'inici amb la que arrencarà l'aplicació.

El SappFramework incorpora tres eines més que han d'estar configurades a dins de la etiqueta "<servlet>". La primera correspon a la definició d'una pàgina d'error per defecte, default.page.error que ens proporciona l'avantatge de no caldre programar una pàgina d'error en cada circumstància d'error, controlada o no, aquesta configuració facilita la redirecció a una pàgina determinada en cas d'error si no s'indica el contrari a través de l'anotació "retornError". El <param-value> s'omplirà amb el nom de la pàgina obviant l'extensió ".jsp". La implementació d'aquesta pàgina és lliure pel programador.

```
<init-param>
  <param-name>default.page.error</param-name>
  <param-value>paginaErrorGeneral</param-value>
</init-param>
```

La segona eina és similar a la primera, que correspon a la sortida en cas d'un error durant la validació de privilegis d'una acció, o sigui, en el cas d'utilitzar l'eina de control d'usuaris del SappFramework, quan el rol d'un usuari no correspon amb els permisos per executar una determinada acció, aquesta serà la pàgina a que es redirigirà l'error. El <param-value> s'omplirà amb el nom de la pàgina obviant l'extensió ".jsp". De la mateixa manera que en el cas anterior, aquesta pàgina es pot implementar com el desenvolupador cregui convenient.

```
<init-param>
  <param-name>page.error.privilegis</param-name>
  <param-value>errordeprivilegis</param-value>
</init-param>
```

Per últim, la tercera eina correspon a la llibreria de Tags que incorpora SappFramework. Per poder tenir accés i utilitzar les seves funcionalitats caldrà indicar a l'arxiu web.xml on es pot trobar l'arxiu que les defineix. Aquest arxiu es subministra amb el framework i la seva ubicació habitual és a la carpeta WEB-INF del projecte, la configuració concreta en aquest cas és la següent:

```
<jsp-config>
  <taglib>
    <taglib-uri></taglib-uri>
    <taglib-location>/WEB-INF/SappFramework-taglib.tld</taglib-location>
  </taglib>
</jsp-config>
```

Aquestes configuracions son les que s'han d'implementar al codi del arxiu web.xml pel cas del SappFramework. Existeixen d'altres que son pròpies del sistema servlet que es poden utilitzar per personalitzar el funcionament, un exemple comú és la definició de la durada d'una sessió, per definir una durada de 30 minuts es faria de la següent manera:

```
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
```

Per obtenir més informació es pot consultar la pàgina:

http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html

### 12.3.2 Annotations

Com s'ha explicat, el SappFramework està basat en la tecnologia Java Annotations per tal de ser configurat i definir el comportament dels controladors i de les accions. A continuació s'exposa cadascuna de les anotacions que implementa el nou framework.

#### 12.3.2.1 @Controlador

Amb l'objectiu de definir que una classe ha de tenir un comportament de controlador, a la seva definició ha d'incorporar aquesta anotació. Un controlador serà instanciat en el moment de la crida d'una de les seves accions, o per la crida a través d'un altre controlador. L'objecte creat amb els resultat de les accions es guardarà en un HashMap anomenat "Sapp" salvat com a un atribut en la sessió en curs, sent accessible des del web consultant-lo al "HashMap" "Sapp" a través del nom assignat. L'anotació @controlador permet dos configuracions a través dels seus atributs:

**String nom() default ""**

L'atribut "**nom**" defineix el nom que tindrà el controlador en el web. Existeixen dues possibilitats per tal d'identificar un controlador. La primera consisteix en no definir aquesta atribut a l'anotació, assignant com a identificador per defecte el nom de la classe en minúscules. La segona opció està en definir aquest atribut amb un identificador. Aquesta segona possibilitat és la més recomanable, ja que possibilita el crear diferents versions de controladors, i simplement canviant el nom de l'identificador permet que l'aplicació Web utilitzi una classe diferent.

**String tipusexecucio() default Constants.TIPUS_EXECUCIO_SESSIO**

Aquest atribut assigna al controlador el tipus d'execució que s'ha de processat. Existeixen dues possibilitats. La primera i definida per defecte és "TIPUS_EXECUCIO_SESSIO", execució de sessió, la qual significa que aquest controlador en crear-se l'objecte de la classe instanciada, es guardarà al "HashMap" d'objectes controladors de la sessió, a més també el podrem trobar al "HashMap" "Sapp" i sempre correspondrà al mateix objecte. La segona possibilitat correspon a "TIPUS_EXECUCIO_PR", execució de petició/resposta, la diferencia amb l'anterior es que no es guarda al "HashMap" d'objectes controladors de sessió, cada cop que s'executa una acció es crea un nou objecte que només es guarda al "HashMap" "Sapp" per poder ser utilitzat al Web.

En la següent definició d'exemple el controlador agafarà el nom de la classe "concuodada" com a nom de controlador per ser utilitzat al Web, i que l'execució serà de sessió que es l'opció per defecte, per tant, l'objecte de la classe instanciada es guardarà al "HashMap" d'objectes controladors de sessió i es mantindrà durant tota la durada d'aquesta sessió.

```
@Controlador
public class ConCuocdada {}
```

El següent exemple el controlador obtindrà el nom de l'atribut "nom" de l'anotació i s'anomena "comentaris". El tipus d'execució serà de petició/resposta, per tant, l'objecte creat per l'execució de l'acció només serà carregat al "HashMap" "Sapp", a cada crida de qualsevol acció d'aquest controlador es crearà un nou objecte.

```
@Controlador (nom="comentaris", tipusexecucio= Constants.TIPUS_EXECUCIO_PR )
public class ConComentaris {}
```

Cal recorda que per realitzar la consulta dels objectes passats al web, en el primer cas el nom que s'ha de buscar al "HashMap" "Sapp" correspondrà amb el de la classe en minúscules "concuodada", i en el segon cas haurà de ser el nom definit a l'atribut de l'anotació "comentaris".

#### 12.3.2.2 @Accio

El controladors son els objectes que modelen el comportament de l'aplicació, aquest comportament be expressat per les accions que es poden dur a terme a través d'ell. Per definir un mètode d'un controlador com a acció, només caldrà afegir l'anotació @acció, i a partir d'aquest moment es tindrà accés al mètode. Per facilitar la tasca al desenvolupadors, es poden definir tres atributs:

**String nom() default ""**

L'atribut "**nom**" defineix el nom que tindrà l'acció en el web. Existeixen dues possibilitats per tal d'identificar una acció. La primera consisteix en no definir aquesta atribut a l'anotació, assignant com a identificador per defecte el nom del mètode en minúscules. La segona opció està en definir aquest atribut amb un identificador. Aquesta segona possibilitat és la més recomanable, ja que possibilita intercanviar mètodes sense això afecti a la implementació al web, ja que el nom continuarà sent el mateix, simplement canviant el nom de l'identificador permet que l'aplicació Web utilitzi un mètode diferent.

**String retorn() default ""**

Aquest atribut està destinat a facilitar la navegació, ja que definint el nom d'una pagina "jsp", el SappFramework dirigirà la navegació cap a aquesta pàgina en cas de que no hi hagin errors. En el cas de que no es defineixi cap nom en aquest atribut, la pàgina redirigida serà l'última vàlida, o sigui de la que procedeix la petició de l'acció.

**String retornError() default ""**

Continuant amb el concepte de facilitar la navegació, l'objectiu d'aquest atribut és el de redirigir la navegació a la pàgina definida en cas de que hi hagi un error en l'execució de l'acció. S'ha d'assenyalar que una acció dona un resultat d'error en el cas de que el booleà de retorn es defineixi a l'estat "cert". O sigui si en l'execució d'un mètode declarat com a acció es detecta una error, forçarem el valor de retorn a "true", el framework en trobar aquest estat redirigirà la navegació cap a la pàgina d'error. Cal remarcar que si no hi ha pàgina definida, la navegació es dirigirà cap a la pàgina d'error per defecte definida en l'aplicació a través de l'arxiu web.xml en el cas de que s'hagi configurat aquest aspecte. En cas contrari la navegació pot quedar buida.

#### 12.3.2.3 @ParamWeb

Les accions acostumen a rebre paràmetres a través del Web, a través d'atributs de formularis o encaixats a la mateixa URL. Per facilitar l'accés a aquest paràmetres sense necessitat de consultar la "request" del servlet cada cop que hem de rebre dades, s'ha habilitat aquesta anotació que forma part de la declaració de paràmetres d'un mètode definit com una acció. Aquesta anotació només necessita un atribut que va lligat directament als atributs rebuts a través de la "request".

**String nom() default ""**
L'atribut "**nom**" defineix el nom de l'atribut que s'ha de rebre de la "request".

Existeixen una sèrie de normes per la utilització d'aquesta anotació:

- El nom ha de coincidir exactament amb el utilitzat al web.
- Ha d'existir l'atribut la classe del controlador.
- Han d'existir els mètodes "get" i "set".
- El tipus del paràmetre associat a l'atribut a de ser el mateix el de la classe.

Proposem un exemple:

```
@Controlador (nom="comentaris", tipusexecucio= Constants.TIPUS_EXECUCIO_PR )
public class ConComentaris {
   private int Idquedada;
   /** Obtenir la llista de comentaris d'una quedada */
   @Accio (nom="comentquedada")
   public boolean getLlistComentarisQuedada(@ParamWeb (nom ="Idquedada") int Idquedada){
     boolean result =true;
           Codi
    return result;
  }
   public int getIdquedada() {
      return this.Idquedada;
   }
   public void setIdquedada(int Idquedada) {
      this. Idquedada = Idquedada;
   }
}
```

En aquest exemple el controlador "comentaris" implementa un mètode anomenat "getLlistComentarisQuedada" el nom que s'utilitzarà al web per accedir a aquesta acció serà "comentquedada". El paràmetre que es passarà del web a l'aplicació s'anomena "**Idquedada**" que es de tipus "**int**", i la classe disposa dels mètodes "getIdquedada()" i setIdquedada(**int Idquedada**). Al web podríem tenir un codi com els següent:

```
<a href="/Awebpfcw/Sapp?/comentaris.comentquedada?&Idquedada=16">títol</a>
```

Aquest codi fa referència a una adreça relativa a un servlet anomenat "Sapp" i que a la seva URL incorpora un paràmetre anomenat "**Idquedada**" amb el valor 16. Aquest valor és el que es podrà recuperar al paràmetre del mètode a que va referència l'adreça.

**S'ha de tenir en conte que el SappFramework utilitza els mètodes "get" i "set" per tal de convertir els paràmetres rebuts en "String" des del Web, al tipus definit a l'atribut de la classe del controlador.**

#### 12.3.2.4 @ParamRequest

Encara que amb el sistema d'accés als atributs del Web a través de l'anotació @ParamWeb i la carrega dels objectes instanciats com a controladors al "HashMap" "Sapp", semblen prou per poder gestionar qualsevol tipus d'aplicació, es troba que s'ha de deixar la porta oberta per tal de poder accedir a la "request" en curs d'una sessió. Per tant, s'ha implementat una anotació per importar-la, llegir-la i manipular-la. Aquesta anotació només es composa d'un atribut que no te cap utilitat pel SappFramework però s'ha incorporat per homogeneïtat, i si és el cas, es vol incorporar la capacitat al SappFramework de gestió diferents "request" en un mateix mètode.

**String nom() default ""**
L'atribut "**nom**" defineix el nom de la "request".

Pot semblar evident però s'ha de puntualitzar que el tipus del paràmetre que ha de passar ha de ser del tipus "HttpServletRequest". Qualsevol altre paràmetre generà un error no permetent l'accés a l'acció. Com el tipus de paràmetre sempre és el mateix no cal que el controlador implementi l'atribut, ni tampoc els mètodes "get" i "set" d'aquest. El següent codi és un exemple d'utilització:

```
@Accio (nom="compses", retornError="errorcomp", retorn = "compok")
 public boolean comprovarSessio(@ParamRequest HttpServletRequest rec){
   boolean result =true;
        Codi exemple
        String session=rec.getSession().getId();
        :
        :
  return result;
}
```

Aquesta acció amb el nom "compses", la pàgina de retorn en cas d'error, o sigui en cas que el mètode de l'acció retorni "true", serà "errorcomp.jsp", com a pàgina de retorn d'una execució correcta, o sigui que el mètode de l'acció retorni "false", serà "compok.jsp", se l'hi passa com a paràmetre la "request" i no s'hi afegeix el nom de l'anotació ja que no es rellevant. Dins del codi podem veure com es pot utilitzar qualsevol atribut que contingui la "request".

#### 12.3.2.5 @ParamApp

Fins ara les anotacions que es refereixen al pas de paràmetres en una acció, sempre han estat referides a la "request" o sigui els paràmetres i atributs rebut i enviats des del Web. Per poder afegir interacció entre els diferents controladors que pot contenir una aplicació, s'ha implementat @ParamApp per tal de poder consultar els objectes controladors creats en una sessió, si son de sessió tipus "TIPUS_EXECUCIO_SESSIO", o instanciar de nous si son de tipus "TIPUS_EXECUCIO_PR".

Aquesta anotació només es composa dos atributs el primer es el nom que no te cap utilitat pel SappFramework però pot indicar quin es el pseudònim del controlador i el segon atribut indica la classe del controlador que s'ha de consultar o instanciar segons sigui el cas.

**String nom() default “”**
L’atribut “**nom**” pot definir el pseudònim de la classe del controlador.

**String nomclasse() default “”**
L’atribut “**nomclasse**” identifica la classe del controlador a la que es fa referència.

S’ha de tenir en conte que només es pot fer referència a les classe que estiguin definides com a controladors a través de l’anotació “@Controlador” i que siguin a dins l’àmbit de l’aplicació definit en l’arxiu web.xml pel paràmetre “ruta.base”. Un exemple d’implementació seria el següent:

```
/**
 * Acció que cerca una quedada per la seva id i els comentaris associats
 */
@Accio (nom= "quedada", retorn = "quedada")
public boolean getQuedada(      @ParamWeb (nom ="Idquedada")int Idquedada,
                                @ParamApp (nom="comentaris") ConComentaris commts){
  boolean result =true;
        Codi exemple
        commts.getLlistComentarisQuedada(Idquedada);
        :
        :
  return result;
}
```

Per aquesta acció amb el nom “quedada”, la pàgina de retorn en cas d’error serà la definida per defecte en l’arxiu web.xml al paràmetre “default.page.error”, com a pàgina de retorn d’una execució correcta, serà “quedada.jsp”. Se l’hi passa com a paràmetre un enter anomenat “Idquedada” que el rebrà com a paràmetre des de la Web, i incorpora com a paràmetre de a nivell d’aplicació la classe “ConComentaris”.

En l’exemple proporcionat per l’anotació @ParaWeb, aquesta classe s’ha definit del tipus “TIPUS_EXECUCIO_PR”, per tant, com el controlador és de petició/resposta la classe no estarà instanciada, el SappFramework s’encarregarà d’instanciar-la i passar-la com a paràmetre a l’acció, per posteriorment passar-la al “HashMap” “Sapp” per la seva utilització a la Web, aquest objecte només persistirà durant aquesta execució de l’acció.

#### 12.3.2.6 @Preexecucio

Una necessitat habitual en la confecció d'una aplicació és la necessitat de tenir un prepocessament de les dades, per tal de realitzar comprovacions, conversions de tipus entre d'altres. L'anotació **@Preexecucio** proporciona una eina per tal de definir etapes anteriors a l'execució d'una acció. Aquestes etapes es configuren i es programen a dins de la classe del controlador, definint diferents mètodes amb aquesta anotació.

Es important remarcar que es poden encadenar diferents preexecucions en l'ordre que es configuri, això permet dividir els preprocessament, de tal manera que es pot aprofitar la reutilització de forma més eficient. La capacitat de reutilització que implementa és molt útil degut a que en molts casos es repeteix la necessitat d'un mateix preprocessat per diferents accions. S'ha d'anar amb conte ja que els atributs que es modifiquin durant aquesta etapa del processament quedaran guardats pels processaments posteriors, o sigui qualsevol atribut de la classe controlador que es modifiqui es persistent a l'objecte controlador en que s'executa.

El funcionament es semblant al d'una acció, es poden incorporar les mateixes anotacions als seus paràmetres. De la mateixa manera que una acció en executar-se pot retornar un resultat correcte o un d'erroni definit pel valor booleà (true=error, false=ok). Un mètode definit de preexecució te el mateix funcionament. En cas de que el resultat de mètode sigui "true", o sigui amb error, el framework no continuarà amb l'execució de l'acció i modificarà la navegació cap a la pàgina d'error que s'hagi definit. En cas que l'execució sigui correcta, continuarà l'execució poden encadenar amb el següent mètode de preexecució si n'hi ha o executant l'acció que correspongui.

Per configurar l'anotació **@Preexecucio** existeixen dos atributs, el primer és multi propòsit i el segon correspon a la pàgina de retorn en cas d'error. No existeix una pàgina de retorn en cas d'una execució correcta, degut a que no te cap sentit ja que l'execució del preprocessat en cas d'èxit continua amb un altre presprocessat o amb l'acció que correspongui.

**String[] metodeacio()**

Aquest atribut és multi propòsit, amb això es vol dir que els paràmetre i el seu ordre tenen un sentit determinat. Per definir un preprocessat per un mètode s'ha d'incloure dos entrades a l'array, el primer paràmetre correspon al nom real del mètode de l'acció, no s'ha d'utilitzar el seu àlies, el segon paràmetre correspon a l'ordre en que s'ha d'executar el preprocessat, això és així degut a que una acció pot tenir diferents mètodes de pretractament i s'ha d'indicar en quin ordre s'han d'anar executant. El paràmetres es poden repetir tantes vegades com calgui per la reutilització de la preexecució en diverses accions.

**String retornError() default ""**

L'objectiu d'aquest paràmetre és el mateix que en el cas de l'anotació **@accio**. En cas de que la preexecució acabi en error, el SappFramework redirigirà la navegació cap a la pàgina que s'hagi definit, si no es defineix cap, es dirigirà a la que s'hagi configurat per defecte.

El següent exemple servirà per aclarir l'anotació **@prexecucio**:

```
/**  Control del format de la data d'inici   */
@Preexecucio (retornError="errorformat",
                    metodeacio={"getQuedadesPerData", "1", "afegirNovaQuedada", "1"})
public boolean controlDataInici (@ParamWeb (nom ="datainici")String datainici){
        boolean result =true;
        Codi
        return result;
}

 /** Control del format de la data final  */
@Preexecucio (retornError="errorformat",
                    metodeacio={"getQuedadesPerData", "2"})
public boolean controlDataFinal (@ParamWeb (nom ="datafinal")String datafinal){
        boolean result =true;
        Codi
        return result;
}

 /** Control que la data d'inici sigui anterior o igual a la data final */
@Preexecucio (retornError="errorformat",
                    metodeacio={"getQuedadesPerData", "3"})
public boolean controlDates (     @ParamWeb (nom ="datainici")String datainici,
                                  @ParamWeb (nom ="datafinal")String datafinal){
        boolean result =true;
        Codi
        return result;
}
```

Al codi anterior identifiquem tres mètodes de preexecució del mateix controlador, es pot observar que es passen paràmetres des del web per tal de tractar-los com si es tractés d'una acció. El primer mètode "controlDataInici" s'executarà en primera instància abans de l'acció que implementa el mètode "**getQuedadesPerData**", però també es reutilitza pel mètode "**afegirNovaQuedada**" executant també com a primer.

El segon mètode "controlDataFinal" del l'exemple s'executarà com a segon pas de la prexecució de l'acció que implementa el mètode "**getQuedadesPerData**". Ens hem de fixar en el nombre enter després de cada definició, indica l'ordre d'execució. Finalment trobem el tercer mètode "controlDates" que també s'ha d'executar abans del mètode "**getQuedadesPerData**" però en tercer lloc. Si les execucions han estat correctes arribats a aquest punt s'executarà l'acció, si durant el procés s'ha modificat atributs de la classe controlador es mantindran els canvis.

**Es molt important tenir en conte que el número d'ordre ha de començar pel 1 no pot haver més de 9 preexecucions encadenades, a més no es poden saltar nombres o sigui han de ser consecutius. En cas contrari el SappFramework en iniciar-se aportarà la informació de l'error en no poder endreçar correctament l'ordre d'execució.**

#### 12.3.2.7 @Postexecucio

Com es pot deduir del nom d'aquesta anotació, els mètodes que l'incorporen s'executaran una vegada l'acció hagi realitzat la seva tasca i el resultat no hagi estat d'error. Les característiques son les mateixes que el cas de l'anotació **@Preexecucio**, però en aquest cas, si existeix un error en el postprocessat, la navegació es dirigirà cap a la pàgina d'error en contes de dirigir-se a la pàgina de sortida de l'acció.

Els atributs de la **@Postexecucio** son els mateixos que la **@Preexecucio** i el seu funcionament és el mateix.

**String[] metodeacio()**
Aquest atribut és multi propòsit, amb això es vol dir que els paràmetre i el seu ordre tenen un sentit determinat. Per definir un postprocessat per un mètode s'ha d'incloure dos entrades a l'array, el primer paràmetre correspon al nom real del mètode de l'acció, no s'ha d'utilitzar el seu àlies, el segon paràmetre correspon a l'ordre en que s'ha d'executar el postprocessat, això és així degut a que una acció pot tenir diferents mètodes de posttractament i s'ha d'indicar en quin ordre s'han d'anar executant. El paràmetres es poden repetir tantes vegades com calgui per la reutilització de la postexecució en diverses accions.

**String retornError() default ""**
L'objectiu d'aquest paràmetre és el mateix que en el cas de l'anotació **@accio**. En cas de que la postexecució acabi en error, el SappFramework redirigirà la navegació cap a la pàgina que s'hagi definit, si no es defineix cap, es dirigirà a la que s'hagi configurat per defecte.

Un senzill exemple seria el següent:

```
 /** Postexecució per per mostrar la data i l'hora de forma correcta de una quedada  */
@Postexecucio ( metodeacio={ "getQuedada","1"})
public boolean convertirDada(){
        boolean result =true;
        Codi
        return result;
}
```

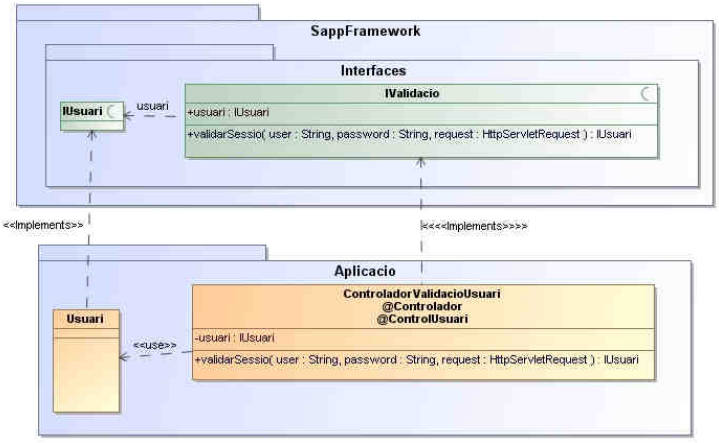
En aquest exemple trobem una postexecució que s'executarà després del mètode "**getQuedada**". No incorpora informació sobre la pàgina d'error per tant serà la definida per defecte.

**Es molt important tenir en conte que el número d'ordre ha de començar pel 1 no pot haver més de 9 postxecucions encadenades, a més no es poden saltar nombres o sigui han de ser consecutius. En cas contrari el SappFramework en iniciar-se aportarà la informació de l'error en no poder endreçar correctament l'ordre d'execució.**

#### 12.3.2.8 @ControlUsuari i el @Rol

Una de les dificultats en que es troben els programadors a l'hora d'implementar una aplicació web, és el control d'usuaris, els seus rols i privilegis. Amb aquestes dues anotacions es proposa una solució per poder implementar un control d'usuaris de forma ràpida i eficaç.

L'anotació **@ControlUsuari** s'aplica sobre les característiques que defineixen un controlador. A més de definir el rol de controlador a través de l'anotació **@Controlador** en una classe que volen que faci de control d'usuaris, s'ha d'afegir l'anotació **@ControlUsuari**. Aquesta anotació va lligada a dues interfícies, la primera "Ivalidacio" l'ha d'implementar la classe controlador per tal d'haver de declarar i programar el mètode "validarSessio". La segona interfície és IUsuari, que implementa una sèrie de mètodes bàsics pel control d'usuari, l'ha d'implementar un classe que es trobi dins la ruta base del nostre projecte.




L'acció especial "validarSessió" és l'única que te un tipus de retorn diferent, una acció normal retorna un booleà, per indicar el resultat d'aquesta OK/NOK, en canvi aquesta acció especial retorna un objecte del tipus de la classe que implementi la interfície "IUsuari" del nostre projecte, en el cas del UML de més amunt seria "Usuari". Aquesta acció en cridar-se des de la Web desencadena determinats mecanismes al SappFramework per controlador l'usuari que s'intenta validar.

El SappFramework, guardarà l'objecte controlador de la nostra aplicació, que en l'exemple serà "ControladorValidacioUsuari", l'objecte "Usuari" que conté el controlador s'ha de guardar sense el password visible. Si s'ha validat correctament també s'ha de guardar el seu rol. Si no s'ha validat correctament, el controlador es guarda de totes maneres però el rol ha d'estar buit per no permetre validar cap acció que o requereixi.

Per accedir a l'usuari només caldrà cercar-lo dins de el "HashMap" "Sapp" que s'envia a través de la "request" al web, de la mateixa manera que passa amb la resta de controladors de l'aplicació.

Fins aquest punt en vist la manera de preparar el sistema per poder gaudir dels avantatges que proporciona el control d'usuaris. Ara exposarem com s'ha d'implementar el sistema de rols a les accions que o requereixin.

Una acció determinada que requereixi que un usuari estigui validat i que disposi d'un determinat rol per poder ser executada, ha d'implementar l'anotació **@rol**. Aquesta anotació permet un anàlisi previ a l'execució de l'acció, comparant el rol de l'usuari amb els rols que permeten la seva l'execució. Aquesta anotació només incorpora un atribut, el qual podrà definir els diferents rols de l'aplicació.

**String[] rols()**
Aquest atribut correspon a un l'array de String's, aquest array contindrà un o més rols que permeten l'execució d'una acció. El nom de cada rol i els seus privilegis han d'estar definits per programador, queda sota el seu control i el de l'aplicació la forma d'accés o de recuperació d'aquestos rols, per exemple a través de la consulta d'una BD que retorni els atributs d'un usuari el qual haurà de tenir un que determini el seu rol.

Amb el següent exemple aclarirem la forma d'implementació del control d'usuaris:

```
public class Usuari implements IUsuari {}

@ControlUsuari
@Controlador (nom="usuaris")
public class ConUsuaris implements IValidacio {
   private Usuari usuari;
   private String correu;
   private String password;
   private String session;
   private String error;

  @Accio (nom="validasession", retornError="errorvalidacio", retorn = "welcome")
   public IUsuari validarSessio(      @ParamWeb (nom ="correu")String correu,
                                      @ParamWeb (nom ="password") String password,
                                      @ParamRequest HttpServletRequest rec){
     DBdades db = new DBdades();
     Usuari usr = new Usuari();
     usr = db.cercaUsuariPerCorreu(correu);

     //Comprovar si l'usuari és correcte
     try{
        if (usr.getCorreu().equals(correu) && usr.getCorreu()!=null ){
           //Comprovar si el password es correcte
           if (usr.getPassword().equals(password) && usr.getPassword()!=null ){
              //Validar la sessió
              session=rec.getSession().getId();
              usr.setValidat(true);
              //borrem el pasword per que no es vegi
              this.password ="";
              usr.setPassword("");

           }else{
              error = "Password incorrecte";
              usr.setRol("");
```

```
            }

        } else{
            error = "Usuari no existeix";
            usr.setRol("");
        }
    } catch(NullPointerException n){
        error = "Usuari o password incorrectes";
        usr.setRol("");
    }
   usr.setPassword("");
   this.usuari = usr;
   return usr;
  }
}
```

En quest exemple es pot veure com la classe "ConUsuaris" se l'hi assigna el nom "usuaris" a més a través de l'anotació "@ControlUsuari" es defineix que aquest controlador utilitzarà la funcionalitat de rols del SappFramework. Com hem dit abans, en la utilització de l'anotació "@ControlUsuari" en una classe controlador serà necessari implementar la interfície "**IValidacio**" per tal d'utilitzar el mètode "**validarSessio**", aquest mètode definit com una acció a través de l'anotació "**@accio**" retornarà un objecte que implementa la interfície "**IUsuari**" que el SappFramework utilitzarà pel control de rols. La resta de codi s'utilitza verificar l'usuari i esborrar el password i el rol en cas de que no es validi correctament.

```
/** Acció que retornar la llista d'usuaris d'una quedada  */
@Rol (rols ={"user", "admin"})
@Accio (nom="quiva", retorn="quiva")
public boolean quiVa(){
   boolean result =true;
       Codi
   return result;
}
```

Aquest exemple especifica la manera d'utilitzar els rols per controlar l'execució d'una acció determinada. L'anotació "**@Rol**" amb el seu atribut "**rols**" ofereixen la llista de rols que poden executar aquesta acció. En cas que el rol de l'usuari no coincideixi amb cap de la llista, el SappFramework redirecionarà la navegació cap a la pàgina definida al framework en l'atribut "page.error.privilegis" configurat en el arxiu web.xml.

**En l'ús d'aquesta eina a part de la obligatorietat d'implementar el sistema com es descriu, també s'ha de configurar la pagina d'error de privilegis en l'arxiu web.xml a través de l'atribut "page.error.privilegis". Si no es realitza aquesta configuració o no es crea la pàgina, després d'un error de privilegis, o sigui si un usuari intentat executar una acció que no te el seu rol assignat, la navegació apareixerà buida o mostrarà un error que no troba el recurs demanat.**

### 12.3.3 Tags

Per tal de donar una primera entrada a la utilització del framework i que els programadors tinguin eines per iniciar una implementació del SappFramework, es proporcionen una sèrie de tags d'ús comú que permeten construir una aplicació web. S'utilitzarà la llibreria "javax.servlet.jsp.*" per implementar-los, i les dades sempre partiran del "HashMap" "Sapp".

Aquestos tags son un exemple bàsic de les capacitats del nou framework, no queda lloc a dubte de la necessitat d'ampliació d'aquestos per poder assolir les prestacions que requereix una implementació real. Per tant, aquests tags s'han de prendre a mode d'exemple per construir d'altres.

Per accedir a la llibreria de tags de SappFramework s'ha de configurar al web.xml on es troba l'arxiu ".tld" que conté la descripció dels tags, en el cas del nou framework aquest arxiu s'anomena "SappFramework-taglib.tld" i es subministra amb el framework per ser inclòs al projecte. Una ubicació habitual d'aquest arxiu és la carpeta WEB-INF del projecte, en aquest cas la configuració al web.xml seria la següent:

```
<jsp-config>
  <taglib>
    <taglib-location>/WEB-INF/SappFramework-taglib.tld</taglib-location>
  </taglib>
</jsp-config>
```

#### 12.3.3.1 datactr

Aquest tag ens permetrà l'accés a les dades guardades als controladors que es passen a la Web a través del "HashMap" "Sapp". Es composa de dos atributs, "controlador" per determinar de quin controlador de la nostra aplicació volem extreure la dada. I el "atribut" per identificar quina dada és la que es vol recuperar el valor. El retorn serà un String amb el valor de l'atribut. Es poden extreure dades de diferents nivells, o sigui, si un Controlador te un atribut complex que es tracta d'un objecte d'una altra classe, també es podrà accedir, separant per un punt "." cada nivell d'atribut. El següent codi és un exemple:

Primer s'ha de Definir la llibreria de tags i el seu nom a dins la pàgina:

```
<%@ taglib uri="WEB-INF/SappFramework-taglib.tld" prefix="spf" %>
```

Un exemple d'us simple, cridem al tag per extreure el valor d'un atribut de nom "datainici" del controlador "concuocdada":

```
<spf:datactr controlador="concuocdada" atribut="datainici"/>
```

Un exemple d'us a diferents nivells, cridem al tag per extreure el valor d'un atribut que te el nom de "titol" de l'objecte "quedada" que està contingut dins el controlador "concuocdada":

```
<spf:datactr controlador="concuocdada" atribut="quedada.titol"/>
```

**El nom de l'atribut sempre ha de coincidir en majúscules i minúscules.**

#### 12.3.3.2 llistanorm

Una necessitat habitual és crear taules a partir d'una llista d'objectes. Aquest tag és la solució per generar una taula amb un nombre de columnes que es poden configurar i relacionades amb els atributs dels objectes continguts en una llista. Com és normal, els objectes que conté una llista han de ser tots del mateix tipus.

Els paràmetres necessaris a introduir al tag son els següents:

- **titolllista:** Defineix el títol de la taula que apareixerà al capdamunt.
- **controlador:** S'ha de parametritzar amb el nom del controlador que conté la llista.
- **nomllista**: nom de la llista a dins del controlador.
- **nomcolumnes**: És un array de Strings separant per comes amb el nom de les columnes que volem que aparegui al capdamunt de la taula, no tenen per que correspondre amb els atributs.
- **nomatributs**: És un array de Strings separat per comes, amb els nom dels atributs que es volen llegir.

S'ha de tenir en conte que l'ordre del nom de les columnes i el ordre dels atributs han de ser el mateix, en cas contrari la informació no encaixarà. Han de coincidir majúscules i minúscules en el paràmetre "controlador" i "nomatributs". Ha d'haver el mateix nombre de paràmetres en el "nomcolumnes" i en el "nomatributs".

El següent codi és un exemple de la utilització del tag:

```
<spf:llistanorm
    titolllista="COMENTARIS"
    controlador="concuocdada"
    nomllista="llistacomentaris"
    nomcolumnes="NOM USUARI, COMENTARI"
    nomatributs="nomusuari, comentari"/>
```

La sortida que provoca serà una taula amb dues columnes amb la informació que contingui la llista, en la següent imatge es pot veure la sortida del exemple anterior:

COMENTARIS

| NOM USUARI | COMENTARI |
|---|---|
| Carles Masses | Rico Rico. |
| Joan Perez | Només van faltar les sevillanes. |

#### 12.3.3.3 llistaref

També és molt comú la necessitat de crear taules amb algun atribut referenciat, o sigui, que un dels components de la llista sigui un enllaç amb un atribut que indica un índex, per tal de executar una acció que aporti més informació sobre aquesta fila de la taula. Aquest tag és la solució per generar una taula amb les columnes que es configurin, i afegirà un enllaç del tipus "<href>" a la primer columna de la taula.

Els paràmetres necessaris a introduir al tag son els següents:

- **titolllista:** Defineix el títol de la taula que apareixerà al capdamunt.
- **controlador:** S'ha de parametritzar amb el nom del controlador que conté la llista.
- **nomllista**: nom de la llista a dins del controlador.
- **ref**: referència que s'ha d'afegir a la columna que s'indiqui.
- **columnaref**: indica l'atribut que s'ha d'utilitzar com a índex.
- **nomcolumnes**: És un array de Strings separant per comes amb el nom de les columnes que volem que aparegui al capdamunt de la taula, no tenen per que correspondre amb els atributs.
- **nomatributs**: És un array de Strings separat per comes, amb els nom dels atributs que es volen llegir.

La confeció de l'enllaç es fa de la següent manera:
<a href="**ref** " + "**=**" + "**valor de la columnaref**">

El següent codi és un exemple de la utilització del tag:

```
<spf:llistaref titolllista="QUEDADES"
    controlador="concuocdada"
    nomllista="llistaquedades"
    ref="/Awebpfcw/Sapp?/concuocdada.quedada?&Idquedada"
    columnaref="Idquedada"
    nomcolumnes="QUEDADA, DATA, HORA"
    nomatributs="titol, data, hora"/>
```

Un exemple en codi HTML que pot generar la primera columna de l'exemple anterior i que podem veure a la imatge de més avall seria:

<a href="/Awebpfcw/Sapp?/concuocdada.quedada?&Idquedada=5">Reunió alumni</a>

QUEDADES

| QUEDADA | DATA | HORA |
|---|---|---|
| Reunió alumni | 01/06/2012 | 20:10 |
| UocGresca | 28/07/2012 | 23:00 |
| UocGresca | 28/07/2012 | 23:00 |
| Cinema a la fresca | 02/08/2012 | 23:10 |
| Curset de Patinatge | 08/10/2012 | 16:45 |

La primera columna correspon a un enllaç que executarà l'acció "quedada" del controlador "concuocdada" i l'hi passarà el paràmetre "Idquedada" amb el valor que correspongui a cadascuna de les files de la taula.

#### 12.3.3.4 usuarival

És habitual proporcionar informació a l'usuari validat sobre el seu estat. S'ha volgut proporcionar una eina per poder visualitzar el nom de l'usuari validat i el seu rol. Aquest tag com els anteriors presentats extreu les dades del "HashMap" "Sapp". Només es necessari un atribut. El del controlador que definit com a controlador d'usuari que s'ha definit com a tal amb l'anotació **@ControlsUsuari.**

El paràmetre necessari a introduir al tag és el següent:

- **controladorusuari:** S'ha de parametritzar amb el nom del controlador definit com a controlador d'usuari que s'ha definit com a tal amb l'anotació **@ControlsUsuari**.

El següent codi és un exemple de la utilització del tag:

```
<spf:usuarival controladorusuari="usuaris"/>
```

Un exemple de sortida d'usuari validat seria el següent:

**Usuari: Joan Perez el teu rol es user**

Un exemple de sortida si no hi ha un usuari validat seria la següent:

**Usuari: Convidat**

Com es pot apreciar les sortides son diferents en el cas de d'usuari validat i usuari no validat. Aquesta informació és útil a l'hora d'informar a l'usuari si ha entrat al seu perfil de la Web i pot tenir un nivell de privilegis superior, o si encara no s'ha validat i està navegant en mode convidat.

## *12.4 Opcions de millora de SappFramework*

Encara que en aquest framework s'han intentat implementar eines útils per la seva utilització, tornem a repetir que en l'aplicació en casos concrets i a través de l'experiència, és on es generen la majoria d'idees per crear els recursos. Per això, aquest tipus d'aplicacions es creen a partir d'un grup de treball composat per professionals que poden aportar els coneixements i les inquietuds, per la confecció d'una bona utilitat.

En el nostre cas hi ha dos aspectes importants per millorar en quan al funcionament, del primer ja hem parlat, es tracta de la inicialització de l'atribut error dels controladors abans de la crida de qualsevol acció, amb l'objectiu de no incorre en interpretacions errònies per mostrar un error anterior que no correspon a l'acció en curs.

El segon aspecte de millora en el funcionament correspon a l'esborrat de la sessió al "HashMap" "mapasessions", responsable de memoritzar els objectes creats en cada sessió, aquesta funcionalitat és necessària si s'intenta utilitzar el framework en un cas real, ja que actualment els objectes s'acumulen en aquest "HashMap" sense que cap operació els buidi de la memòria, encara que la sessió s'hagi tancat o hagi caducat. S'ha de crear un mètode que es cridi intervals regulars de temps i que analitzi si les sessions guardades encara son operatives, si no és així esborrar-les.

Tal com he dit, de funcionalitats es poden crear tantes com les necessitats demanin. En quan al tipus d'eines d'obtenció i enviament de dades de i cap a la Web es creu que son complertes. Però si parlem de les eines del tipus Tag, aquestes son escasses, ja s'ha exposat en un punt anterior que es molt necessari que un framework incorpori les màximes possibles per facilitar l'accés a les dades a la web, modelar la sortida i aconseguir una bona interfície gràfica.

Una millora passaria per la incorporació d'altres frameworks per enriquir aquest aspecte. Per tant, podem dir que la inclusió d'un framework que ajudi a renderitzar les sortides i aporti moltes utilitats en aquest àmbit, seria una millora substancial. S'ha incorporat un mètode per facilitar aquest aspecte, però el fet que ja vingui programat facilitaria molt el procés d'integració.

## *12.5 Conclusions*

Amb tot l'exposat durant l'explicació del Sappframework del seu desenvolupament i funcionalitats, podem afirmar que no és trivial el plantejament d'un projecte d'aquestes característiques. Sens dubte és necessària la intervenció de diferents professionals de diferents disciplines que aportin la seva experiència per obtenir un producte amb la usabilitat i eficàcia que demana un "programa per programar".

Un framework com aquest demana del coneixement profund de diferents disciplines i tècniques, per tal que les decisions de disseny i implementació siguin preses correctament i que hi hagi un equilibri entre els diferents components que l'integren. El fet que aquest framework hagi estat desenvolupat per un sol professional, fa que hi hagi aspectes que segurament no s'hagin enfocat correctament per extreure el màxim partit de les tecnologies aplicades.

En el transcurs de la confecció del SappFramework, han aparegut entrebancs que s'han solucionat de la manera que s'ha cregut convenient, però sense poder avaluar diferents punts de vista d'aquests problemes, no es pot afirmar que les solucions proposades siguin les més adequades, però sí funcionals. En definitiva un producte que funciona però millorable.

Per acabar direm que els frameworks d'estudi del projecte han estat una referència per importar idees, tant a nivell de desenvolupament con a nivell de funcionalitats implementades, no és una copia, si no un recull del que es creu son les millors característiques de cadascun d'ells.

# 13 Web CuocDades

## *13.1 Introducció*

Per tal d'oferir un exemple el qual implementi totes les eines que SappFramework incorpora, s'ha construït una aplicació web que implementa una xarxa social per alumnes de la UOC, destinada a oferir informació sobre esdeveniments tant de la institució com de fora d'ella. Aquesta xarxa social es gestionada per un conjunt d'administradors que la moderen per tal de donar d'alta tant a usuaris com a esdeveniments.

L'aplicació ofereix informació sobre esdeveniments passats i futurs que s'han dut i es duran a terme. Ofereix eines per cercar per dates o veure tots els esdeveniments per tal de visualitzar la informació detallada, l'adreça i l'hora del esdeveniment. Pels usuaris enregistrats també permet realitzar comentaris o preguntes sobre l'esdeveniment, a més de poder apuntar-s'hi i veure els participants que hi aniran.

Aquesta aplicació està basada completament en el framework Sapp com a ajuda a la capa de presentació. Internament com a base de dades s'ha utilitzar Postgresql 9.1, i el servidor que suporta el sistema és Glassfish 3.1. Son tot eines que tenen la opció de programari lliure, encara que existeixen versions de pagament amb característiques ampliades que no es requereixen per aquesta aplicació.

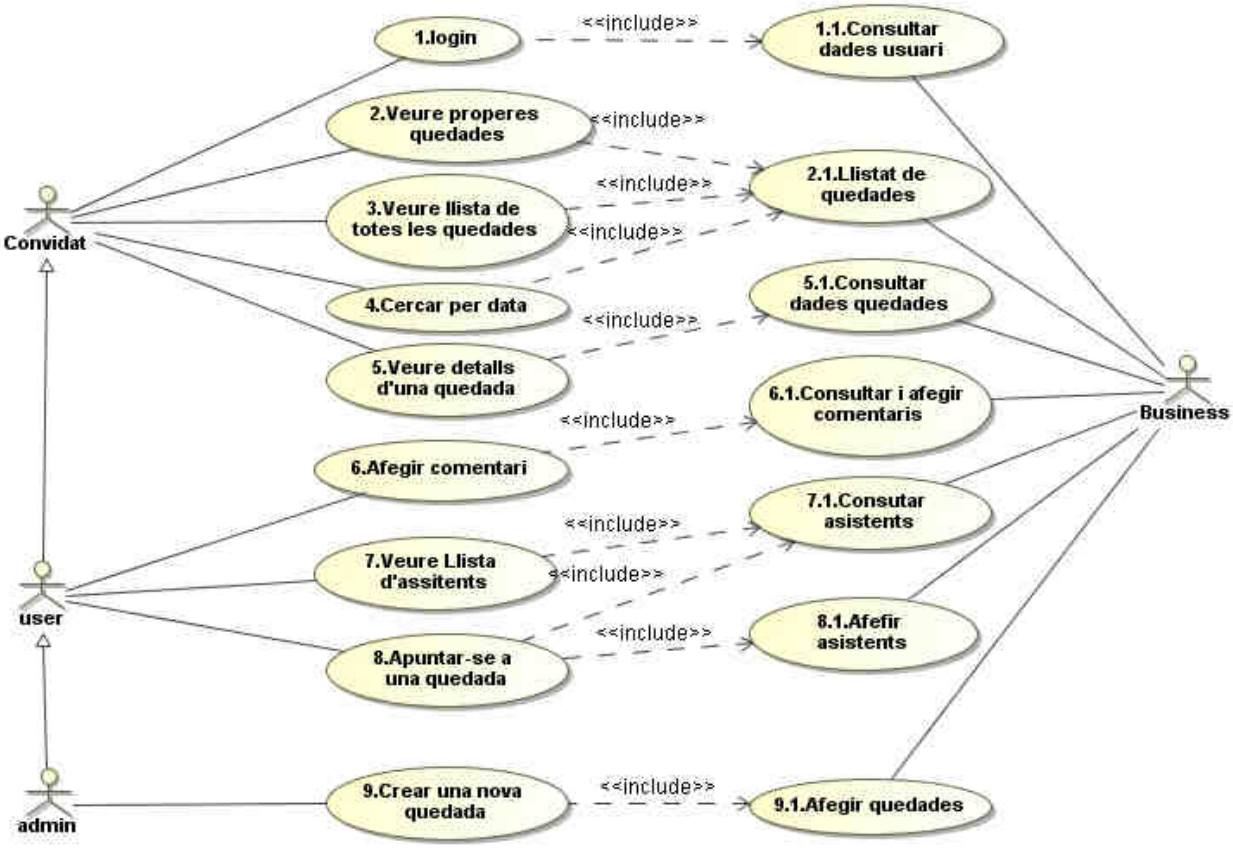
A l'hora de desenvolupar l'aplicació s'ha realitzat de tal manera que incorpori tots els avantatges del SappFramework, amb un intent de ser un exemple d'utilització d'aquest nou Framework, però també per que serveixi a altres programadors com a guia, tant pel seu ús, com per l'enteniment del funcionament d'una eina oberta, amb l'objectiu que sigui fàcil de modificar i ampliar la seva funcionalitat a través de l'experiència de la seva aplicació. S'ha inclòs la inclòs documentació en **"javadoc"** de tot el codi per tal de facilitar la seva comprensió i com a guia sobre el comportament de l'aplicació.

En els següents apartats s'exposarà el projecte de web, i a través del requisits de l'aplicació i les solucions aplicades es podrà entendre la utilització de determinades eines del SappFramework que trobarem implementades al seu codi. Conceptes com Controlador, Acció, Rol, paràmetres, etc... que hem estat constantment nombrant en el transcurs de l'explicació del framework, es veuran d'una manera més clara i entenedora a través del codi d'aquesta aplicació.

Finalment cal dir, que per arribar a ser una aplicació completament real s'haurien d'implementar més funcionalitats en el projecte Web, com son l'alta d'usuaris, més gestió a les quedades com esborrar, donar d'alta, etc..., en definitiva un aspecte més confortable. Totes aquestes funcionalitats no aportarien més coneixement sobre els SappFramework, que en definitiva és l'objectiu de l'aplicació Web d'exemple.

## *13.2 Casos d'ús*

### 13.2.1 Diagrama de casos d'ús




### 13.2.2 Especificació detallada de casos d'ús

Els casos d'ús que s'especificaran seran els principals.

**1. Login**

- **Nom:** Login
- **Resum de la funcionalitat:** Valida l'usuari a través a la web per obtenir els privilegis que corresponguin al seu estatus, "user" o "admin"
- **Casos d'us relacionats:** 1.1 Consultar dades d'usuari a la BD per obtenir les dades necessàries per validar l'usuari.
- **Flux d'esdeveniments principal:**
  1.1) L'usuari selecciona l'opció "Login " del menú principal.
  1.2) Es mostra la pàgina de login.
  1.3) L'usuari ha d'introduir el seu correu electrònic i el seu password.
  1.4) L'usuari selecciona l'opció "Acceptar".
  1.5) En validar-se es mostra la pàgina de benvinguda.
- **Flux d'esdeveniments alternatiu:**
  1.4.1) Si hi ha un error durant la validació es mostrarà la pàgina d'error amb la possible incidència durant la validació.

**2. Veure properes quedades**

- **Nom:** Veure properes quedades
- **Resum de la funcionalitat:** L'acció retorna la llista de properes quedades a partir de la data actual.
- **Casos d'us relacionats:** 2.1 Consultar dades de quedades de la BD a partir de la data actual.
- **Flux d'esdeveniments principal:**
  2.1) L'usuari selecciona l'opció "Cuocdades" del menú principal.
  2.2) Es mostra la pàgina amb la llista de les properes quedades.
- **Flux d'esdeveniments alternatiu:**
  2.1.1) Si no hi ha quedades a partir de la data es mostrarà la llista buida.

**3. Veure llista de totes les quedades**

- **Nom:** Veure llista de totes les quedades.
- **Resum de la funcionalitat:** L'acció retorna la llista de totes les quedades que hi ha, tant passades com futures.
- **Casos d'us relacionats:** 2.1 Consultar la BD per obtenir la llista de totes les quedades.
- **Flux d'esdeveniments principal:**
  3.1) L'usuari selecciona l'opció "Llista totes les Cuocdades".
  3.2) Es mostra la pàgina amb la llista totes les quedades.
- **Flux d'esdeveniments alternatiu:**
  3.1.1) Si no hi ha quedades es mostrarà la llista buida.

**4. Cercar per data**

- **Nom:** Cercar per data.
- **Resum de la funcionalitat:** Cercar quedades entre una data inicial i una data final.
- **Casos d'us relacionats:** 2.1 Consultar la BD per obtenir la llista de les quedades entre les dues dates.
- **Flux d'esdeveniments principal:**
  4.1) L'usuari Introdueix paràmetres de "Data Inici" i "Data Final".
  4.2) L'usuari selecciona "Cercar".
  4.3) Es mostra la pàgina de quedades entre dates, amb informació de les dates escollides i la llista de quedades que coincideix a el paràmetres de la cerca demanada.
- **Flux d'esdeveniments alternatiu:**
  4.2.1) Si no hi ha quedades es mostrarà la llista buida.
  4.2.2) Si les dates no estan correctament formades es mostra la pàgina d'error de format.
  4.2.3) Si la data inicial es superior a la final es mostra la pàgina d'error amb el missatge de dates incorrectes

**5. Veure detall d'una quedada**

- **Nom:** Veure detall d'una quedada.
- **Resum de la funcionalitat:** Mostra tota la informació de la quedada seleccionada a través de l'enllaç de la llista de quedades.
- **Casos d'us relacionats:** 5.1 Consultar la BD per obtenir els detalls de la quedada seleccionada.
- **Flux d'esdeveniments principal:**
  5.1) L'usuari selecciona una quedada a través de l'enllaç de la llista de quedades que es mostra per pantalla.
  5.2) Es mostra la pàgina de detall de la quedada seleccionada.
    a. Detalls de la quedada
    b. Comentaris de la quedada

**6. Afegir comentari**

- **Nom:** Afegir comentari.
- **Resum de la funcionalitat:** Afegir un comentari a la quedada que es mostra actualment.
- **Casos d'us relacionats:** 6.1 Inserir nou comentaria a la BD de la quedada que es mostra en pantalla.
- **Flux d'esdeveniments principal:**
  6.1) L'usuari introdueix el comentari.
  6.2) L'usuari selecciona afegir.
  6.3) Es mostra a la mateixa pàgina el comentari que ha afegir l'usuari.
- **Flux d'esdeveniments alternatiu:**
  6.2.1) Si l'usuari no està validat es mostra la pàgina d'error de privilegis.
  6.2.2) Si el comentari està buit es mostra la pàgina d'error indicant que s'ha d'omplir el comentari

**7. Veure llista d'assistents**

- **Nom:** Veure llista d'assistents.
- **Resum de la funcionalitat:** Mostra la llista d'assistents de la quedada que la qual es mostren els detalls.
- **Casos d'us relacionats:** 7.1 Consultar a la BD la llista d'assistents de la quedada que es mostra.
- **Flux d'esdeveniments principal:**
  7.1) L'usuari selecciona la opció "Qui va?".
  7.2) Es mostra una nova pàgina amb els detalls de la quedada i la llista d'usuaris que assisteixen a la quedada.
- **Flux d'esdeveniments alternatiu:**
  7.1.1) Si l'usuari no està validat es mostra la pàgina d'error de privilegis.

**8. Apuntar-se a una quedada**

- **Nom:** Apuntar-se a una quedada.
- **Resum de la funcionalitat:** L'usuari s'afegeix a la llista d'assistents de la quedada.
- **Casos d'us relacionats:** 7.1 Consultar a la BD la llista d'assistents de la quedada que es mostra per comprovar que no hi es encara, inserir un nou assistent si és el cas.
- **Flux d'esdeveniments principal:**
  8.1) L'usuari selecciona la opció "ANAR A LA QUEDADA".
  8.2) A la mateixa pàgina es mostra l'usuari a la llista d'assistents.
- **Flux d'esdeveniments alternatiu:**
  8.2.1) Si l'usuari no està validat es mostra la pàgina d'error de privilegis.
  8.2.2) Si l'usuari ja està a la llista d'assistents es mostra la pàgina d'error demanant de verificar la seva assistència.

**9. Crear una nova quedada**

- **Nom:** Crear una nova quedada.
- **Resum de la funcionalitat:** Afegeix una nova quedada a la llista de quedades de la BD.
- **Casos d'us relacionats:** 9.1 Inserir una nova quedada a la llista de quedades de la BD.
- **Flux d'esdeveniments principal:**
  9.1) L'usuari selecciona l'opció "Admin" del menú principal.
  9.2) L'usuari selecciona l'opció "Agefir Cuocdades"
  9.3) Es mostra la pàgina per introduir les dades de la nova quedada.
  9.4) L'usuari ha d'omplir tots els camps: "Titol", "Descripció", "adreça", " Data", "Hora".
  9.5) L'usuari selecciona l'opció "Afegir".
  9.6) Es mostra la pàgina de quedada creada.
  9.7) L'usuari selecciona "Torna".
  9.10) Es mostra la pàgina de properes quedades.
- **Flux d'esdeveniments alternatiu:**
  9.1.1) Si l'usuari no està validat o no te privilegis d'administració amb el rol "admin" es mostra la pàgina d'error de privilegis.
  9.5.1) Si algun cap està buit es mostra la pàgina d'error que ho indica.
  9.5.2) Si la data no està ben formada es mostra la pàgina d'error de format.
  9.5.3) Si l'hora no està ben formada es mostra la pàgina d'error de format.

## *13.3 Model conceptual de dades*

L'aplicació està desenvolupada sobre la base de dades Postgresql 9.1. a continuació es mostra el model conceptual de dades del sistema.

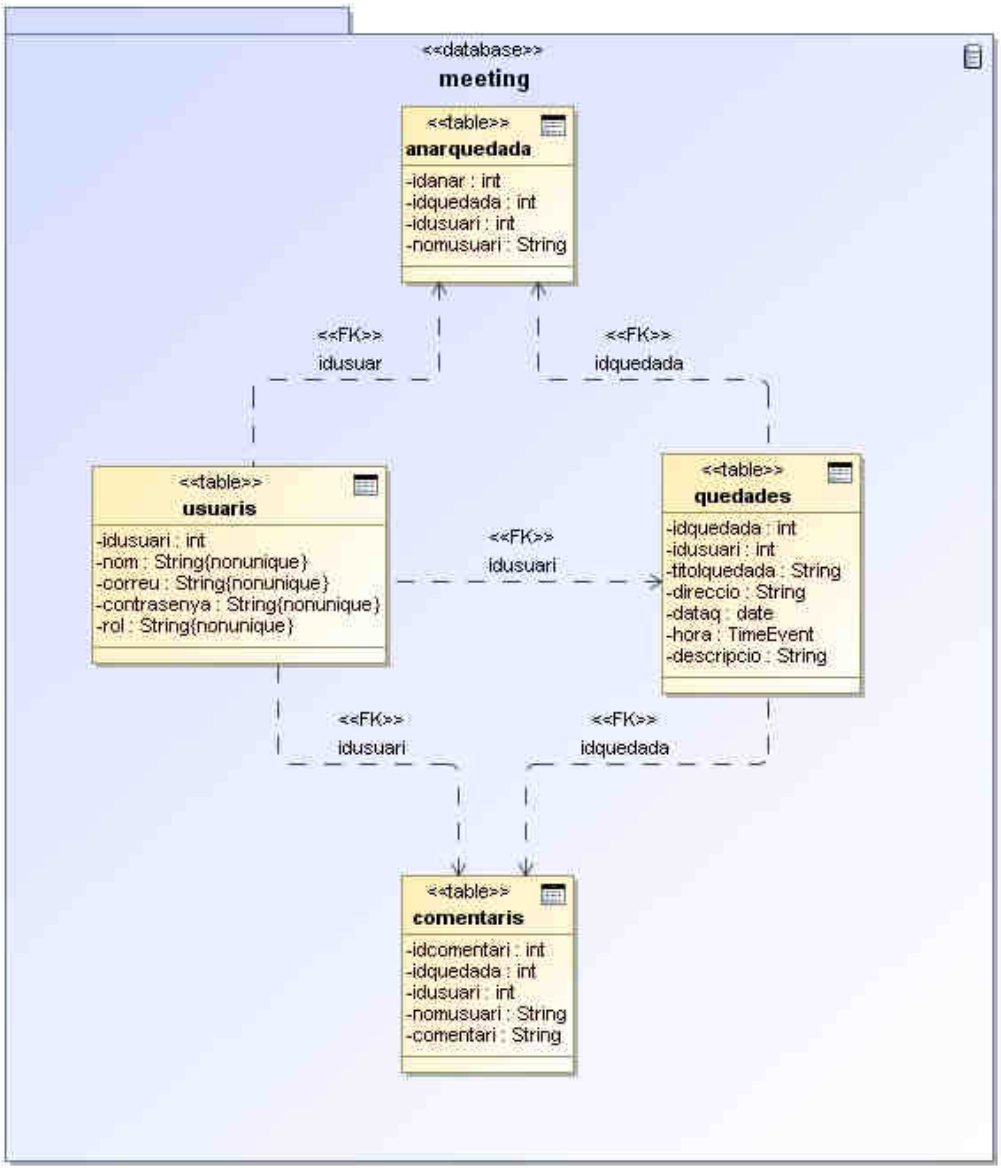



Fig. Model conceptual de dades

Cal tenir en compte que s'ha de generar la base de dades per tal de que l'aplicació funcioni correctament. Per facilitar la seva instal·lació es proporcionen l'arxiu Script d'SQL anomenat "crearBD.sql"per generar les taules de la base de dades "meeting", i amb les insercions a la base de dades per tenir un joc de proves amb el que fer funcionar l'aplicació.

Per la connexió a la base de dades existeix la dependència de la llibreria de connexió a Postgresql basada en "jdbc". No caldrà importar aquesta llibreria ja que el projecte la incorpora. El driver de connexió es pot descarregar de la següent pàgina http://jdbc.postgresql.org/download.html .

### 13.3.1 Restriccions d'integritat

Pel bon funcionament de l'aplicació, s'han definir les següents restriccions d'integritat per les dades i les taules que composen la base de dades. En aquesta llista de restriccions queden definides les entitats que intervenen al model de dades.

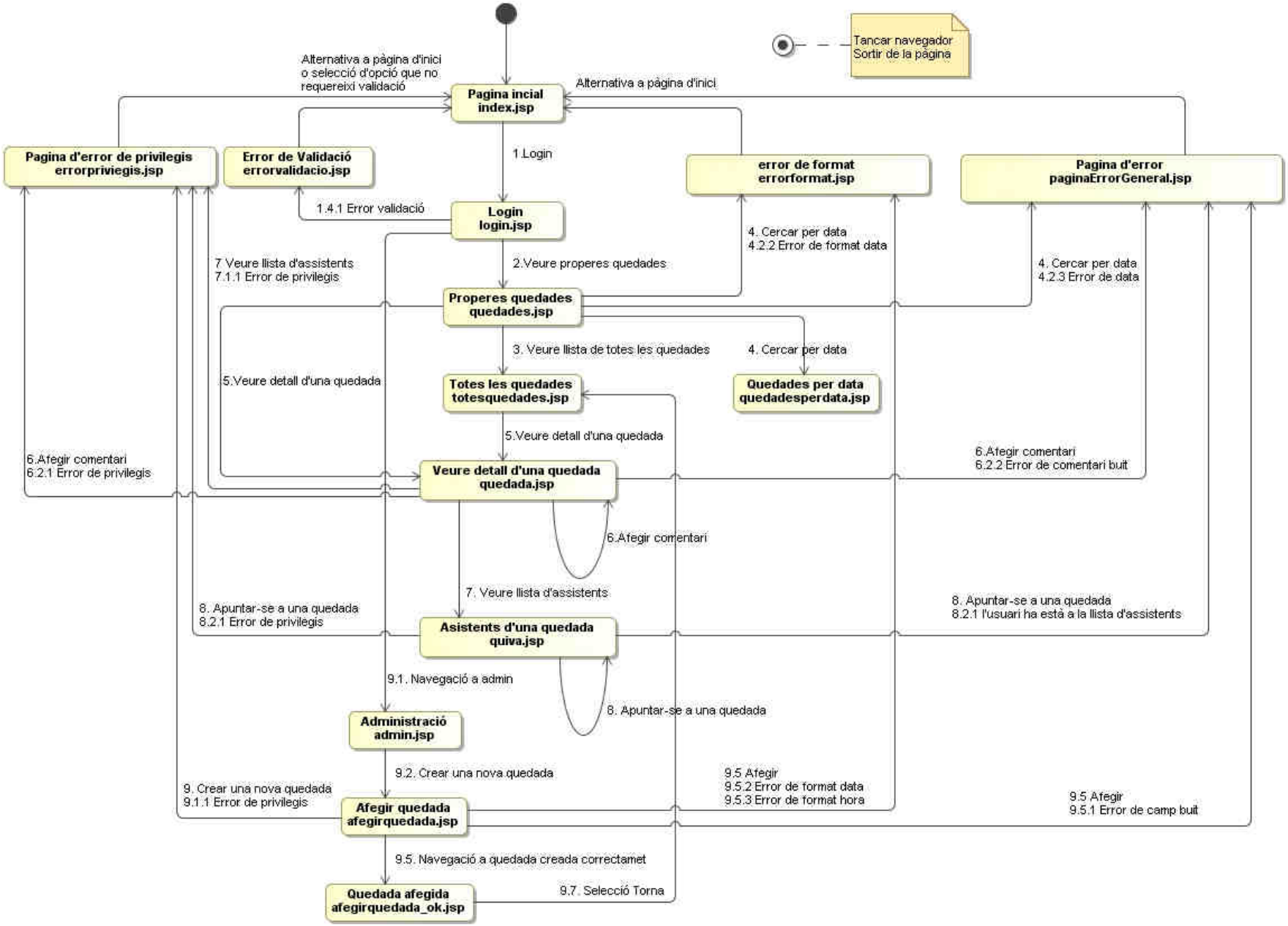
1. Un usuari s'identifica per "idusuari" i per "correu".
2. Una quedada s'identifica per "idquedada".
3. Un comentari s'identifica per "idcomentari" i per "idusuari"+"idquedada".
4. Un assistent s'identifica per "idanar" i per "idusuari"+"idquedada"
5. Una quedada està creada per un únic usuari amb rol "admin".
6. Un usuari pot generar diversos comentaris d'una quedada.
7. Un usuari només pot estar una vegada relacionat amb l'assistència a una quedada, relació (idusuari-idquedada) →taula anarquedada.

## *13.4 Diagrama d'estats de la interfície gràfica d'usuari*

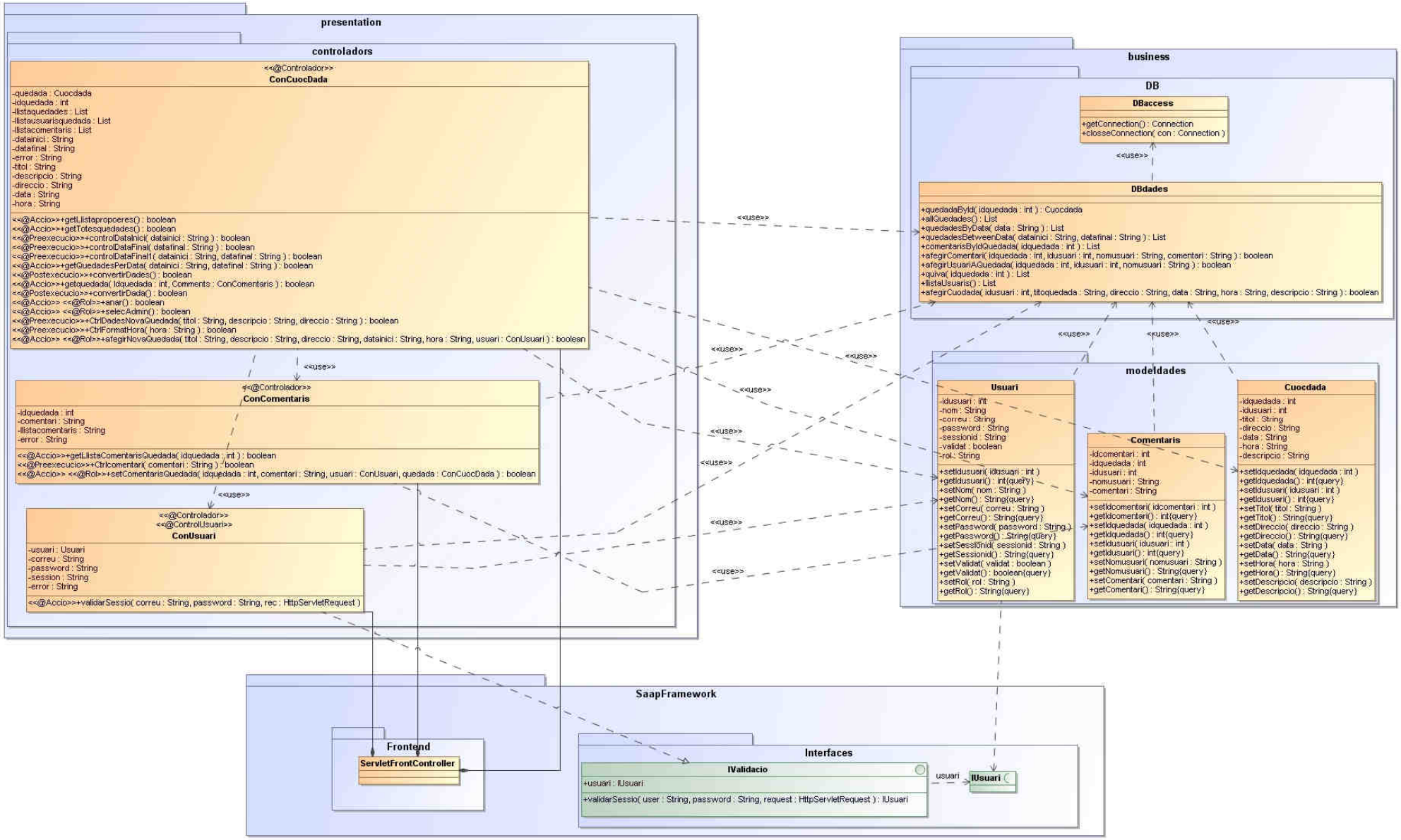
Per interpretar correctament el següent diagrama d'estats, s'ha de tenir en conte que a les pàgines que s'han implementat existeix un menú general des del qual es pot navegar a d'altres pàgines. En cas d'error s'ofereix l'alternativa d'anar a la pàgina d'inici, però també es podria navegar cap a la pàgina de login, la pagina per veure les quedades actuals o la pàgina d'administració.

Els casos d'ús queden reflectits amb la numeració definida en aquest document i es poden troba a les línies que representen les transicions entre pàgines. Quan existeix un flux alternatiu a l'etiqueta apareix el cas d'us principal i com a segon comentari es pot reconèixer la identificació del camí alternatiu amb la seva descripció.

Per últim indicar que existeixen accions que retornen a la mateixa pàgina, reflectides al diagrama amb fletxes cap a la mateixa pàgina. No existeix un enllaç per sortir de la pàgina o de la sessió, per tant la sortida s'ha representat en una etiqueta que indica que la sortida de l'aplicació és amb el tancament del navegador o la sortida de la pàgina.


Tancar navegador
Sortir de la pàgina
Alternativa a pàgina d'inici
o selecció d'opció que no
requereixi validació
Alternativa a pàgina d'inici
Pagina incial
index.jsp
Pagina d'error de privilegis
errorpriviegis.jsp
Error de Validació
errorvalidacio.jsp
1.Login
error de format
errorformat.jsp
Pagina d'error
paginaErrorGeneral.jsp
1.4.1 Error validació
Login
login.jsp
4. Cercar per data
4.2.2 Error de format data
7 Veure llista d'assistents
7.1.1 Error de privilegis
2.Veure properes quedades
4. Cercar per data
4.2.3 Error de data
Properes quedades
quedades.jsp
3. Veure llista de totes les quedades
4. Cercar per data
5.Veure detall d'una quedada
Totes les quedades
totesquedades.jsp
Quedades per data
quedadesperdata.jsp
5.Veure detall d'una quedada
6.Afegir comentari
6.2.1 Error de privilegis
6.Afegir comentari
6.2.2 Error de comentari buit
Veure detall d'una quedada
quedada.jsp
6.Afegir comentari
7. Veure llista d'assistents
8. Apuntar-se a una quedada
8.2.1 Error de privilegis
8. Apuntar-se a una quedada
8.2.1 l'usuari ha está a la llista d'assistents
Asistents d'una quedada
quiva.jsp
9.1. Navegació a admin
8. Apuntar-se a una quedada
Administració
admin.jsp
9.2. Crear una nova quedada
9.5 Afegir
9.5.2 Error de format data
9.5.3 Error de format hora
9. Crear una nova quedada
9.1.1 Error de privilegis
Afegir quedada
afegirquedada.jsp
9.5 Afegir
9.5.1 Error de camp buit
9.5. Navegació a quedada creada correctamet
Quedada afegida
afegirquedada_ok.jsp
9.7. Selecció Torna

## *13.5 Diagrama de classes de l'aplicació*



Al diagrama de classes de l'aplicació representat, es poden apreciar les relacions entre els diferents elements que la composen. S'ha detallat l'anotació utilitzada en cada cas com un estereotip. Cal destacar que per no fer excessivament complex aquest diagrama, s'han obviat els gets i sets dels atributs de les classes de tipus controlador, insistim en que si aquest atributs s'utilitzen com a paràmetres de tipus "@ParamWeb", la classe controlador ha de disposar dels gets i sets d'aquestos atributs per poder identificar el tipus de que es tracta a l'hora d'invocar els mètodes des del SappFramework.

Les classes "@Controlador" s'instancien des del el "ServletFrontControler", es pot apreciar la relació és composició entre ells, per tant, en destruir la sessió s'haurien d'eliminar els objectes associats. El paquet "bussines" és utilitzat per aquestes classes "@Controlador" per accedir a les dades i poder donar resposta a les peticions que realitzen els usuaris des de la Web. Per poder modelar les dades rebudes des de la BD, el sistema es recolza en les classes auxiliars del paquet "modeldades", facilitant l'accés als atributs del dels diferents elements que manipula l'aplicació.

Tal com s'ha explicat anteriorment SappFramework proporciona una eina pel control del rol dels usuaris a l'aplicació. La classe "ControlUsuari" implementa la interfície "Ivalidació" que obliga a l'ús del mètode "validarSessio" on es troba el codi que gestiona el "login" d'un usuari a la Web. L'objecte de la classe "Usuari" creat s'utilitzarà per consultar el rol de l'usuari validat. Ens hem de fixar que en algunes accions s'utilitza l'estereotip "Rol", indicant que l'acció requerirà un o més tipus de rol per poder se executada.

Només d'un cop d'ull, el propi diagrama de classes ens informa d'una separació clara entre les capes de l'aplicació. Aquest efecte és positiu ja que aquest desacoblament o separació és el proposat en el model MVC, que s'ha tractat de respectar tant en la confecció del framework com en la implementació d'aquest en l'aplicació Web que presentada.

Encara que no s'hagi detallat, es pot suposar que les pàgines JSP tindran una relació d'ús per parteix des del "ServletFrontControler". No s'han indicat les dependències amb les llibreries que utilitza el framework, ni tampoc l'utilitzada per gestionar la connexió de la base de dades PosgreSQl utilizada a la classe "DBacces", aquesta informació creiem que no aporta coneixement sobre l'aplicació.

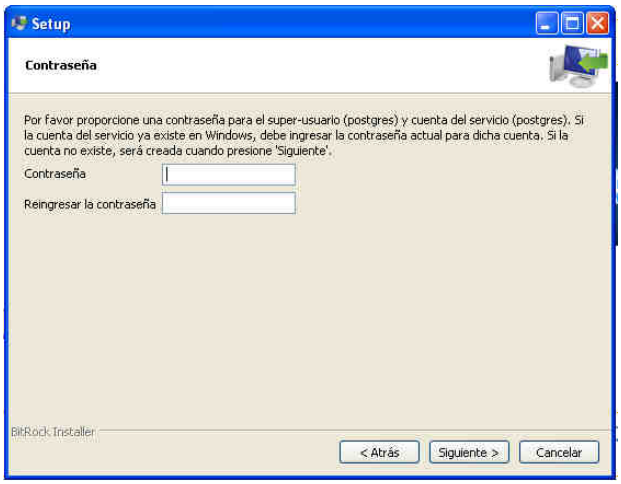
Finalment direm que en aquesta aplicació s'han utilitzat totes les funcionalitats que ofereix SappFramework. És un exemple interessant per tal d'estudiar el comportament d'aquest davant d'un programa real, encara que aquest funcionalment no incorpori totes les necessitats que es requeririen. Tenim els controladors, la validació d'usuaris, el control dels rols, les accions amb totes les possibilitats de configuració, preexecucions i postexecucions, el controls d'errors, la navegació, l'aprofitament de la configuració del "web.xml", i per últim la utilització dels tags a la pàgina Web, tot això implementat de diferents maneres a mode d'exemple i amb el "**javadoc"** corresponent per tenir el màxim d'informació sobre el desenvolupament.

## *13.6 Instal·lació de l'aplicació*

### 13.6.1 Instal·lació de la Base Dades Postgresql 9.1

1) Descarregar l'aplicació Postgresql 9.1, la següent adreça és vàlida:
http://www.enterprisedb.com/products-services-training/pgdownload#windows
2) Escollir l'arxiu en funció del SO i executar-lo.
3) Durant la instal·lació configurar els següent paràmetres:

3.1) Password: "admin"

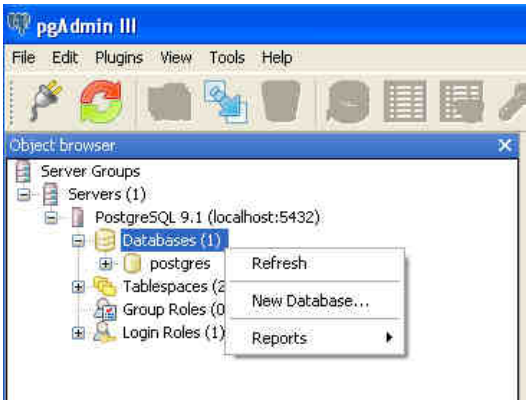



3.2) Port: "5432"
3.3) La resta de configuració ha de ser la que dona per defecte
3.4) El nom d'usuari de Postgresql ha de ser "postgres"
3.5) Servidor: "localhost" (en cas d'executar la BD en la mateixa màquina).

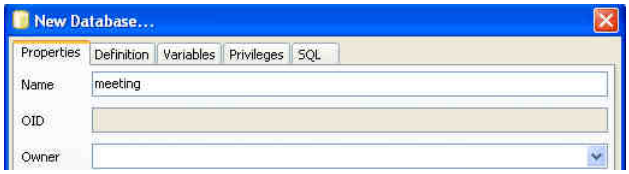
4) Executar l'aplicació PgAdmin III del menú "Inicio" a la carpeta PostgreSQL 9.1
5) Fer doble clic sobre [PostgreSQL 9.1 (localhost:5432)]

5.1) Si demana password introduir "admin" que és el que hem definit durant la instal·lació.

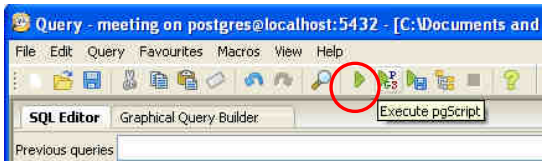
6) Fer clic amb el botó de la dreta sobre DataBases i escollir New DataBase... per crear la base de dades de l'aplicació.



7) Crear la base de dades amb el nom “meeting” i clicar OK




8) Fer doble clic sobre la nova base de dades que accedir
9) Seleccionar del menú principal la icona amb el símbol d’una lupa.
10) Es desplegarà el menú de l’aplicació “Query”, seleccionem file i open, s’ha d’ obri l’arxiu subministrat “crearDB.sql”
11) Clicar sobre la icona “Execute Pgscript”.

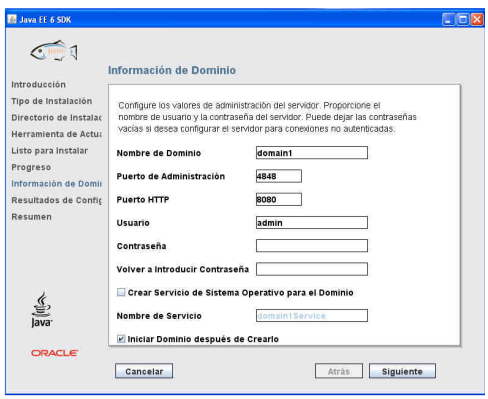



12) Es crearan les taules i les insercions del joc de proves.

En tot cas si un d’aquestos paràmetres varia i es vol corregir a l’aplicació, a la classe “DBacces.java” del projecte es poden modificar els atributs de connexió a la base de dades.

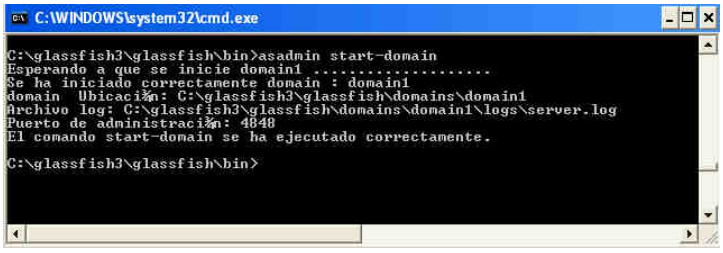
### 13.6.2 Instal·lació de J2EE i servidor GlassFish 3.1

1) Descarregar l’aplicació J2EE + GlassFish 3.1, a següent adreça és vàlida: http://www.oracle.com/technetwork/java/javaee/downloads/java-ee-sdk-6u3-jdk-7u1-downloads-523391.html
2) Escollir l’arxiu en funció del SO i executar-lo.
3) Durant la instal·lació acceptar les opcions per defecte:
    3.1) “Instalación típica”
    3.2 els ports 4848 i 8080 han d’estar lliures, si no és així s’haurà de modificar la configuració per poder accedir a la consola i a l’aplicació.

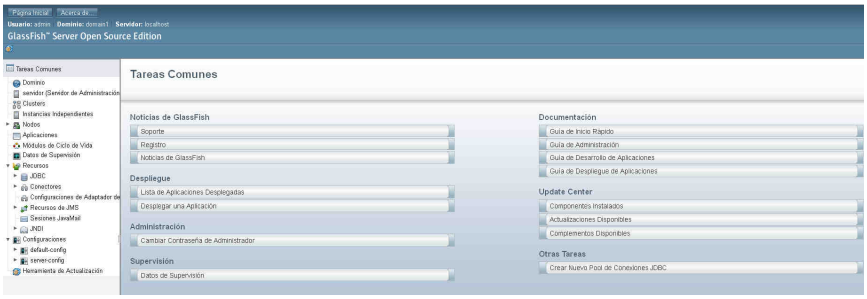



3.3) Escollim la contrasenya i cliquem a “Siguiente”

Cal remarcar que el servidor GassFish en parar i arrencar la màquina no arrencarà automàticament i caldrà iniciar el servidor manualment. Obrirem una finestra amb el símbol de sistema executant la comanda "cmd" des de "ejecutar" del menú inici. Ens situarem al subdirectori on s'hagi instal·lat el servidor, amb la configuració per defecte ha de ser a c:\glassfish3, i des de aquest subdirectori hem d'arribar a c:\glassfish3\glassfish\bin , en aquest punt executarem la comanda que inicia el servidor "asadmin start-domain" i es ficarà en servei.




Per comprovar que tot es correcte obrirem el navegador i introduirem la URL "localhost:4848" ha d'aparèixer el següent resultat:



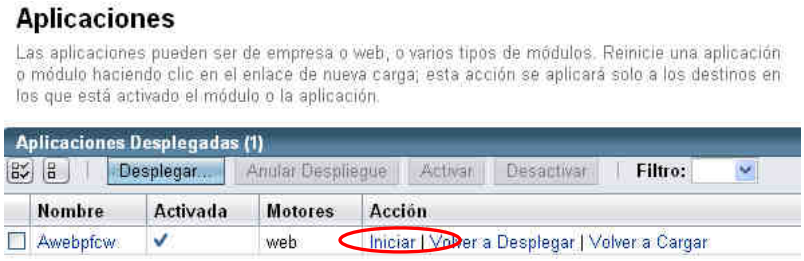

### 13.6.3 Desplegat de l'aplicació

Amb el projecte es subministra l'arxiu "Awebpfcw.war", preparat amb totes les llibreries necessàries per no tenir problemes amb les dependencies i no haver de cercar llibreries per incorporar-les al projecte. Per tal de poder fer el desplegament obrirem una consola del servidor GlassFish3 a través del navegador introduir la següent URL, "localhost:4848". En cas de que en demani Usuari i contrasenya utilitzarem les que s'ha configurat durant la instal·lació.

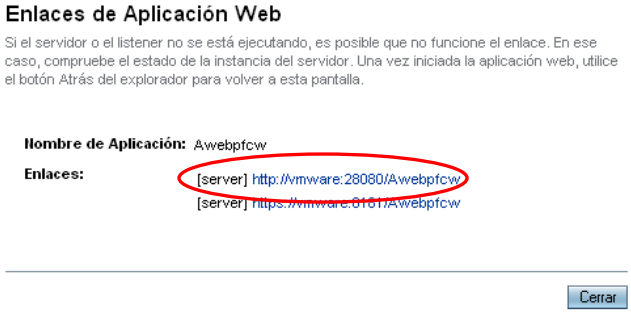
Del menú de l'esquerra escollirem l'opció "Aplicaciones", una vegada a la pàgina "aplicaciones" escollirem l'opció "Desplegar":

A la següent pàgina escollirem "Examinar" i seleccionarem l'arxiu "Awebpfcw.war" subministrat d'allà on l'haguem desat. I clicarem sobre l'opció "Aceptar" de la pàgina a dalt a la dreta:

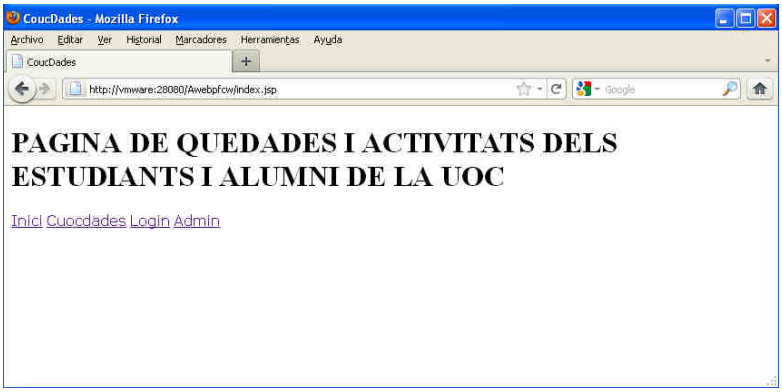

Una vegada fet el desplegament caldrà iniciar l'aplicació, això ho durem a terme clicant sobre l'opció "Iniciar" de la pàgina que ens ha aparegut després de fer el desplegament:




Ens oferirà dues opcions amb ports diferents per poder accedir a l'aplicació, en el cas que es mostra a sota, el port 8080 estava ocupat i s'ha escollit el 28080 com a opció durant la instal·lació, serà aquest enllaç el que hem d'escollir per tal d'anar a la pàgina d'inici.



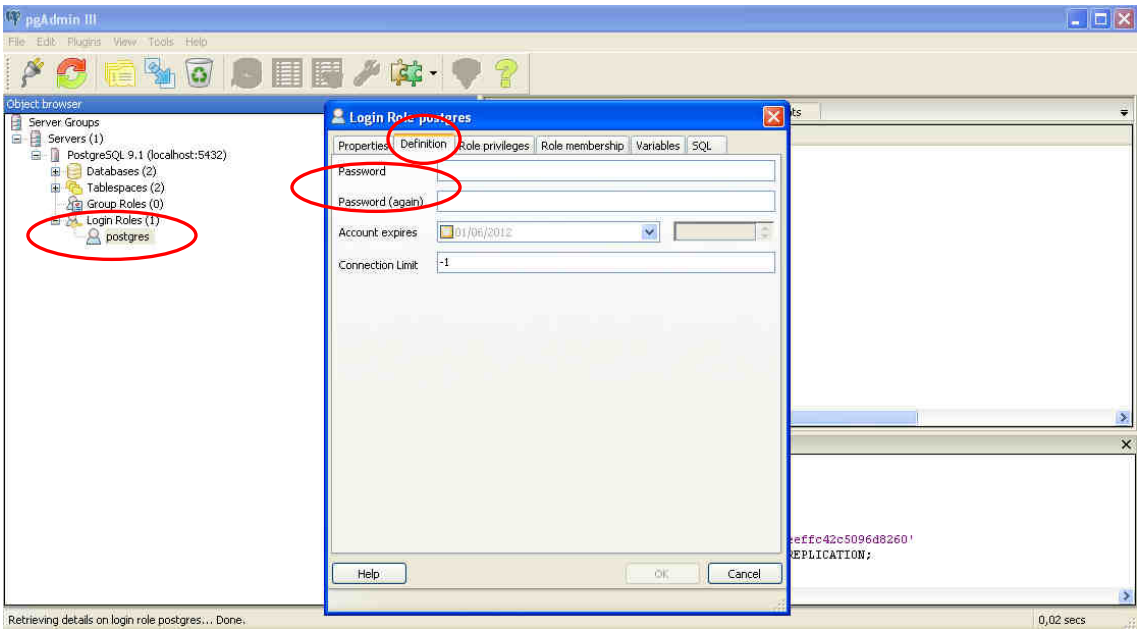

S'obrirà un nou navegador que ens mostrarà la pàgina d'inici de l'aplicació CuocDades, a partir d'aquí es pot utilitzar el digrama de navegació proporcionat juntament amb els cassos d'ús per fer les proves de funcionament de l'aplicació i del SappFramework.



### 13.6.4 En cas d'error

En el moment de fer la instal·lació, es possible que la configuració de la base de dades no es pugui fer amb la contrasenya que requereix l'aplicació, això serà causa d'error en el moment d'intentar navegar. Cal recordar que a l'aplicació la configuració de l'enllaç realitzada a la classe "DBacces.java", es pot modificar l'aplicació i tornar-la a compilar o es pot aplicar la solució que exposem més a baix.

A través del pgAdmin III, l'administrador de PostgreSQL, seleccionarem "Login Roles" i clicant amb el botó de la dreta escollirem "Properties" anirem a la solapa "Definition" i introduirem el Password "admin".




### 13.6.5 Joc de proves

La base de dades es proporciona unes dades carregades prèviament per poder navegar i per tal que l'aplicació ofereixi les taules amb informació que provin el funcionament. Aquestes dades es poden conèixer, o a través de la base de dades, a través d'una consulta SQL o obrin l'arxiu l'Script proporcionat per la creació de la base de dades a través de l'aplicació "Query", a la qual s'accedeix a través de la icona "SQL" amb el símbol de la lupa de l'aplicació pgAdmin III.

Cal destacar que només un dels usuaris inserits te el rol de administrador "admin". Per validar-se com a tal, caldrà introduir com a usuari jmartinez@correu.com i la contrasenya "2234". Si no es fa cap login es podrà navegar amb restriccions de privilegis, i si es valida per exemple amb l'usuari jperez@correu.com i contrasenya "1234" que te el rol de "user" no es podran inserir noves quedades que una opció destinada a l'administrador únicament, però es podrà accedir a opcions com introduir comentaris o veure qui va a una quedada.

## *13.7 Conclusions de l'aplicació Web*

La utilització del SappFramework per la implementació d'aquesta web ha tingut dues conseqüències. La primera és la correcció d'errors no detectats durant la implementació, situació inevitable en qualsevol nou projecte. I la segona i més interessant, ha estat la millora del sistema introduint modificacions per salvar entrebancs apareguts durant la seva implementació, com ha estat el cas de poder executar una acció des de més d'un rol d'usuari, la possibilitat d'accés a la "request" com a paràmetre d'una acció, o la incorporació de la tecnologia de Tags al framework.

Aquest últim cas és el més rellevant per tres causes, la dificultat en la seva implementació, el temps necessari per portar-la a terme i per últim el descobriment de la importància de la fase de construcció Web en una aplicació d'aquest tipus, aquesta fase requereix molt temps per acoblar integració i presentació a més de la interfície gràfica, i per la qual cosa s'ha arribat a la conclusió que és primordial que qualsevol framework que s'implementi com a eina per la capa de presentació, ha de tenir destinada una part molt important del desenvolupament a oferir funcionalitats que aportin solucions en aquesta etapa, per així eliminar el màxim de codi introduït a les pàgines Web, quedant una implementació neta i facilitant la tasca del programador.

Com es pot apreciar en els comentaris anteriors, ha estat positiva l'experiència de la realització del disseny i implementació del cas, ja que ha aportat el punt de realitat i experiència professional necessaris per oferir una solució funcional tal i com s'explicava en les conclusions de l'estudi del frameworks de mercat. Una eina confeccionada a partir de necessitats reals, obtindrà com a resultat solucions reals.

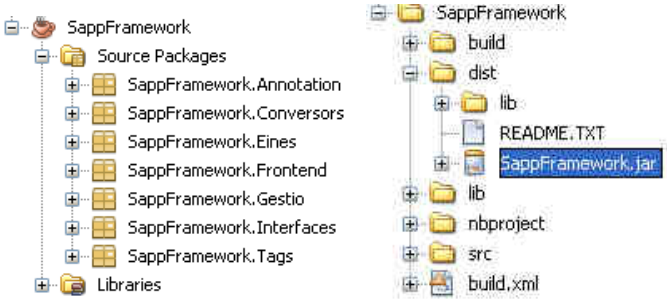
Com a comentari final, afegirem que aquest desenvolupament s'ha fet amb ànims de ser un exemple clar, més que no pas una aplicació viable i utilitzable, implementant el codi a mode de guia per l'usuari del SappFramework, que no del de l'aplicació en si, això es podrà deduir en la quantitat de comentaris inserits al codi, a més del "javadoc" que acompanya a l'aplicació, al qual es recomanable fer-l'hi un cop d'ull per poder seguir l'exemple correctament.

# 14 Lliurables

## *14.1 SappFramework*

En l'apartat del nou framework, s'entrega un projecte complert desenvolupat en NetBeans 7.0.1. La configuració ha estat realitzada de tal manera que les llibreries dependents queden integrades per tal de no haver de cercar-les, i que no hi hagi problemes de compatibilitat en el moment de l'execució d'una aplicació basada en aquest framework.

En el subdirectori "dist" de la carpeta del projecte trobarem l'arxiu SappFramework.jar, aquest arxiu és la distribució del framework per ser utilitzat a qualsevol aplicació sense necessitat de resoldre dependències de cap tipus. A la següent imatge es mostren els paquets que composen el framework i la distribució de carpetes amb el detall de l'arxiu de distribució.

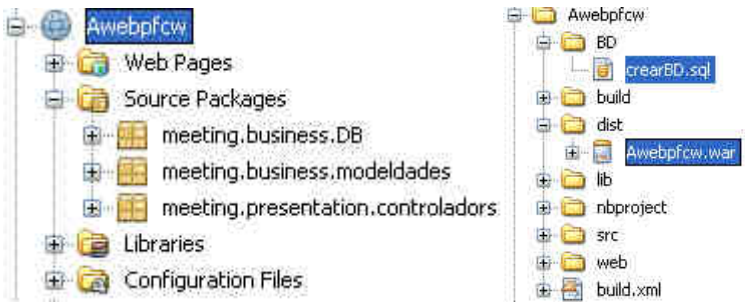



## *14.2 Web CuocDades*

El projecte Web donat com a exemple d'execució del framework està realitzat en NetBeans 7.0.1. La configuració ha estat realitzada de tal manera que les llibreries dependents queden integrades per tal de no haver de cercar-les, i que no hi hagi problemes de compatibilitat en el moment de l'execució de l'aplicació.

El projecte principal està en la carpeta Awebpfc i el de l'aplicació web el trobarem a la carpeta Awebpfcw. Al subdirectori "dist" de la carpeta "Awebpfcw" trobarem l'arxiu Awebpfcw.war, aquest arxiu és la distribució de l'aplicació, recordem que s'ha de desplegar en un servidor GassFish 3.1. Per generar el joc de probes de la base de dades, es proporciona l'arxiu crearBD.sql que el trobarem a la carpeta BD del projecte.

A la següent imatge es mostren els paquets que composen l'aplicació Web, i la distribució de carpetes amb el detall de l'arxiu de distribució i l'arxiu SQL pel joc de proves proporcionat.



# 15 Conclusions del Projecte

Al llarg del projecte s'han dut a terme tasques de documentació, investigació i implementació de nou programari, amb l'objectiu de proporcionar un estudi sobre el frameworks per la capa de presentació aplicables a J2EE , així com un producte final que reculli les necessitats que s'han de cobrir en la implementació d'una eina d'aquest estil.

Ha quedat clar, que la implementació de programari que sigui útil pels desenvolupadors, no es una tasca gens fàcil. Ha d'estar dissenya i programada per un equip de professionals de diferents disciplines que aportin la seva experiència per enriquir l'eina i que incorpori funcionalitats pràctiques que s'ha s'adaptin a la realitat de la tècnica.

Un detall important que cal destacar, és la necessitat d'un sistema efectiu de control d'errors, que ajudi als programadors a detectar i corregir implementacions errònies. Aquest concepte en aplicacions normals només s'aplica al control d'errors d'aplicació, però per un framework s'ha d'anar més lluny per facilitar la tasca del programador durant la configuració i utilització de l'eina, afegint un nivell més de control d'errors, el d'implementació.

També han entrat en joc les tecnologies externes, amb la utilització de llibreries ja confeccionades que garanteixen la qualitat i els resultats, i eviten amb el seu ús haver de programar o implementar conceptes que altres ja han treballat i han ficat a disposició per poder gaudir de la reutilització del codi, aconseguint un estalvi de temps i recursos.

Com a tot projecte, el seu inici es distingeix per la falta de coneixements per dur-lo a terme. Durant la seva confecció es desencadena un procés d'aprenentatge de les tecnologies que l'envolten, necessàries per aconseguir els objectius, aquesta situació enriqueix las professionals que estan obligats a un reciclatge continu a cada nou repte que apareix a la seva carrera.

Per acabar afegirem que aquest aprenentatge ha estat la part més positiva de tot el projecte. La necessitat d'investigació d'arquitectures i tecnologies a aplicar, cóm són els productes de mercat, l'aprofundiment en l'ús de l'API de JAVA, la recerca de noves tendències com ara les "Annotations", els modes de configuració utilitzats habitualment, entre molts d'altres, han ampliat en gran mesura els coneixements que es tenien previs al projecte, aportant una visió diferent d'una implementació d'aquest tipus de la que es tenia al començament. Encara que aquesta tasca ha estat molt llarga i laboriosa, es valora molt positivament tot que s'ha aconseguit en termes d'aprenentatge i coneixements.

# 16 Glossari

**J2EE** Java 2 Enterprise Edition.

**FRAMEWOK** Marc de treball

**SERVLET** Component de la llibreria “javax”

**JSP** Java Server Pages

**MVC** Model Vista Controlador

**WEB SERVER** Servidor Web

**WEB CONTAINER** Contenidor capa Web

**EJB** Enterprise JavaBeans

**ANNOTATIONS** Anotacions java (@)

**LISTENER** Escoltadors d’esdeveniments

**SAPP** Sistema Àgil per la Producció de la capa de Presentació

**CONTROLADOR** Classe del tipus @Controlador

**Action** Mètode del tipus @Action

**ROL** Atributs i privilegis d’un usuari

**HASHMAP SAPP** Contenidor de dades de sortida a la Web

**TAG** Eina basada en TagLib de la programació Web

**DTD** Document Type Definition

**XML** Extensible Markup Language

**JAR** Java ARchive

**WAR** Web aplication ARchive

**JAVADOC** Documentació d’aplicacions Java

# 17 Bibliografia

Centre de terminologia; *Servei de consultes en línea* [en línea] http://www.termcat.cat/
[ data de consulta: del 2-3-2012 al 18-6-2012 ]

Camps, J.M.; (2006). "Mòdul 4 J2EE. Una plataforma de components distribuïda". *Enginyeria del Programari de Components i Sistemes Distribuïts*. Barcelona: Fundació per la Universitat Oberta de Catalunya.

González F.J., Pradell J., Raya J.A,; (2005).. *Enginyeria del Programari orientat a objectes*. Barcelona: Fundació per la Universitat Oberta de Catalunya.

Oracle.com; *Document Information. Your First Cup* [en línea]
http://docs.oracle.com/javaee/6/firstcup/doc/docinfo.html
[data de consulta: 5-3-2012]

Monografias.com; *Aplicación con Tecnología J2EE para la Administración y control en el Área de Logística. Figura nº06 conjunto de API's J2EE* [en línea]
http://www.monografias.com/trabajos82/aplicacion-tecnologia-j2ee/aplicacion-tecnologia-j2ee2.shtml
[ data de consulta: 8-3-2012]

Jtech.ua.es; *Introducción a los servidores de aplicaciones. Figura 1 Aquitectura J2EE* [en línea] http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm
[ data de consulta: 9-3-2012]

Oracle.com (abril 2010); *Introduction to Java Platform, Enterprice Edition 6. White Paper* [en línea] http://www.oracle.com/us/products/middleware/application-server/050871.pdf
[data de consulta: 12-3-2012]

Allamaraju S., Beust C., Davies J.; (2002). *PROFESIONAL Programación Java Server con J2EE Edición 1.3*. Anaya Multimedia / WROX.

Patronesdiseniophp.blogspot.com.es; *Patrones de diseño PHP* [en línea]
http://patronesdiseniophp.blogspot.com.es/p/que-es-un-patron_09.html
[data de consulta: 18-3-2012]

Javacamp.org; *About Design Pattern* [en línea]
http://www.javacamp.org/designPattern/
[ data de consulta: 20-3-2012]

The Apache Software Foundation; *About Apache Struts 2* [en línea]
http://struts.apache.org/2.2.1/docs/
[ data de consulta: 25-3-2012]

Roughley I.; (2006). *Practical Apache Struts2 Web 2.0 Projects*. USA. Apress. ISBN-13 978-1-59059-903-7

Springsource.org; *Reference Documentation Spring 3.1* [en línea]
http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/pdf/spring-framework-reference.pdf
[ data de consulta: 4-4-2012]

Oracle.com; Part II The Web Tier. Java Server Faces Technology. *The Java EE 6 Tutotial.* [en línea] *http://docs.oracle.com/javaee/6/tutorial/doc/bnapk.html*
[data de consulta: 15-4-2012]

Sicuma.uma.es; Tutorial de JavaServer Faces. [en línea]
*http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf*
[data de consulta: 16-4-2012]

Java Platform, Standard Edition 6; API Specification. [en linea]
http://docs.oracle.com/javase/6/docs/api/
[data de consulta: 18-4-2012]

Java Servlet Technology; Java Servlet Technology. [en linea]
http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
[data de consulta: 20-4-2012]

Sistemas de información; Tecnologías Web. Interactividad y envío de información Cliente->Servidor. Servlets. [en linea]
http://www.it.uc3m.es/mcfp/docencia/si/material/4_servlets_mcfp.pdf
[data de consulta: 21-4-2012]

Tutorials Point; Servlets Tutorial [en linea]
http://www.tutorialspoint.com/servlets/servlets_tutorial.pdf
[data de consulta: 22-4-2012]

The Java Tutorials; Annotations. [en linea]
http://docs.oracle.com/javase/tutorial/java/javaOO/annotations.html
[data de consulta: 24-4-2012]

Google Reflections; A Java runtime metadata analysis. [en linea]
http://code.google.com/p/reflections/
[data de consulta: 28-4-2012]

Blog de Dan Jared; Usar logs en Java. [en linea]
http://danjared.wordpress.com/2009/05/11/usar-logs-en-java-parte-1/
[data de consulta: 5-5-2012]

PostgreSQL ; Documentation. [en linea]
http://www.postgresql.org/docs/
[data de consulta: 16-5-2012]