Framework J2EE

Implementació diseny d'un framework per accelerar la productivitat en el desenvolupament d'aplicacions J2EE

Alumne: Sergi Calvet Rodríguez
Titulació: Enginyeria tècnica en informàtica de sistemes

Consultor: Joan Vicent Orenga Serisuelo

Data: 10/06/2012

Resum

Aquest treball consisteix en el disseny i la implementació d'un framework de desenvolupament que pugui ajudar a una organització a unificar tecnològicament i visualment els seus projectes informàtics i a l'hora accelerar els temps de desenvolupament.

Està ubicat dins de l'àrea J2EE i busca aprofitar la potència i maduresa d'aquest plataforma per tal de construir aplicacions robustes i escalables. A l'hora, també integra amb una sèrie de frameworks àmpliament estesos en el mercat i que es complementen i amplien les funcionalitats definides pels estàndards de la plataforma Java.

La organització del projecte és modular, de manera que cada mòdul cobreix una part dels requeriments que una aplicació empresarial pot tenir i pot ser ser importat o no en funció de les necessitats de cada projecte. Existeixen però tres mòduls "*boot*", "*dependencies*" i "*core*" que són imprescindibles i gestionen les funcionalitats bàsiques com ara l'arranc del context d'Spring, la gestió de dependències i les funcionalitats base.

El mòduls cobreixen aspectes com l'accés a dades, les traces, la configuració multi-entorn, missatges multi-idioma, la seguretat, la gestió de dependències, la publicació de serveis web, la generació d'informes, gestió d'excepcions, transaccionalitat ... però s'ha hagut d'ajustar als terminis i l'esforç previstos per aquest TFC. No obstant té vocació de ser estès afegint nous mòduls o ampliant els existents un cop s'implantés dins d'una empresa com a framework de referència.

Index

Index de figures

1. Introducció

1.1. Justificació i context del framework

Avui en dia, totes les organitzacions disposen d'una infraestructura informàtica i demanden, per tant tot tipus de programari per tal de projectar la seva imatge a la xarxa, millorar l'eficiència dels seus processos, millorar la relació amb els seus clients...
Un subconjunt de les necessitats de programari que requereixen aquestes organitzacions només pot ser coberta amb programari a mida, per la seva capacitat d'ajustar-se més al negoci de l'organització i a les seves necessitats concretes. Aquesta part sol ser encarregada a proveïdors externs coordinats per un departament d'informàtica més o menys gran en funció de l'empresa.
Aquesta situació té una sèrie de riscos que si no es tenen en compte pot portar a una mena de caos tecnològic amb els següents inconvenients:

- Dificulta que les aplicacions s'entenguin entre si, s'integrin correctament i es pugui moure parts d'una aplicació a una altres o que una pugui realitzar una petita part incrustable dins d'una altre.
- Dificulta que pugin ser migrades d'un entorn a un altre sense haver d'avaluar el cost individualment, aplicació per aplicació.
- Dificulta el manteniment d'aquestes i obliga als equips de manteniment a dominar una gran quantitat de tecnologies diferents.
- Dificulta el reaprofitament de codi i d'experiència, que es puguin desenvolupar components que puguin ser reutilitzats per diverses aplicacions, o que els equips de desenvolupament puguin cooperar i compartir coneixement.
- Ralentitza el temps de desenvolupament ja que cada aplicació comença des de zero. Ha de dissenyar una arquitectura, decidir quines tecnologies utilitzar per solucionar problemes bàsics i comuns.
- Fa costós mantenir un aspecte visual unificat que pugui anar evolucionant amb el temps ja que aquest problema no s'aborda d'una manera global.

És per tant, convenient per l'organització tractar d'unificar al màxim el desenvolupament per tractar d'evitar els riscos que s'han descrit.

1.2. Objectius

L'objectiu és el desenvolupament d'un framework de desenvolupament basat en estàndards i en tecnologies lliures i madures per tal que es pugui implantar sense costos de llicències. Ha de ser senzill d'utilitzar i permetre que els programadors es puguin centrar en desenvolupar les funcionalitats específiques de la seva aplicació ja que les solucions al problemes transversals han de ser clares i proporcionades per l'arquitectura.
Els objectius concrets que s'han fixat dins l'abast d'aquest TFC son:

a) Descripció de la base tecnològica

En primer lloc es descriuran les tecnologies sobre les que es sustenta el framework, tot aquell conjunt de llibreries elegides per tal de proporcionar la funcionalitat comuna a tots els projectes. Han de ser tecnologies madures, potents i que a l'hora es puguin integrar entre si per tal que el model de programació sigui el més uniforme possible.

S'explicarà quins beneficis aporten i quin paper juga cada una d'elles en l'arquitectura proposada.

b) Descripció de l'arquitectura

El framework i les seves eines conformen una arquitectura, es a dir una serie d'elements que actuen en temps d'execució, cada un amb els seus rols i unes relacions concretes entre ells (Arquitectura d'execució). També una sèrie d'eines i metodologies que ajuden al desenvolupador en la implementació del projecte (Arquitectura de desenvolupament). Un dels objectius és també descriure l'arquitectura a aquestos nivells.

c) Implementació del framework

La part més important d'aquest projecte consisteix en la implementació d'un marc de treball sobre el qual es puguin construir aplicacions robustes, que:

- Permeti maneig de forma senzilla conceptes com la transaccionalitat, les traces, les excepcions, la configuració, la persistència, la generació d'informes...
- Amb una gestió de dependències centralitzada que permeti crear configuracions per tal de desplegar en múltiples servidors d'aplicacions *java*.
- Fàcilment extensible i ampliable amb nous requeriments.
- Que s'ajusti a molts dels estàndards proposats per la plataforma Java Enterprise Edition (JEE) com ara Servlet 3.0, JSF 2.0, JTA 1.1, JPA, Bean Validation entre d'altres.
- Que integri tota una sèrie de llibreries madures i àmpliament acceptades en mercat com ara Spring, Hibernate, Richfaces, CXF, Jasper Reports... sense requerir gaire configuració per al programador.

d) Implementació d'una aplicació de referència

Es realitzarà una aplicació que inclogui tots els mòduls del framework i disposi d'un cas d'ús real seguint l'arquitectura del marc de treball. L'objectiu d'aquest aplicació és:

- Demostrar que el framework és capaç de funcionar en un servidor real.
- Demostrar l'us de mòduls i serveis.
- Servir d'exemple de configuració i d'implementació per a aplicacions noves.

e) Implementació d'arquetipus

Amb l'objectiu d'accelerar l'arranc en la fase de desenvolupament d'un projecte, i homogeneïtzar d'una manera natural l'estructura dels projectes es proporcionaran una serie d'arquetipus que permetran la creació d'un nou projecte ja funcionant partint d'una sèrie de paràmetres. Els arquetipus seran els següents:

- Projecte nou: crearà un projecte nou, amb els mòduls configurats (base de dades, seguretat, configuració multi-entorn, i18n ...) llest per arrencar.
- Mòdul de serveis: afegirà a un projecte framework un mòdul per a publicar serveis web.

1.3. Enfocament i mètode seguit

Cal distingir clarament la metodologia que s'ha seguit per realitzar aquest treball amb la metodologia que hauria de seguir un cop implantat en una organització a l'hora d'evolucionar-lo ja que es tracta de 2 contextos diferents.

a) Metodologia seguida en aquesta assignatura

Aquest treball està emmarcat dins l'assignatura *TFC*, per tant s'ha hagut d'ajustar a una temporització i unes fites molt concretes que han estat: Realització d'un pla de treball, Anàlisis funcional, Disseny tècnic, Implementació, Memòria i Presentació. L'enfoc metodològic que més s'ajusta a aquesta temporització es el desenvolupament en cascada on es poden mapejar fàcilment les fases a les fites i entregables de l'assignatura. És a dir:

- Anàlisis de requisits: en aquest cas s'acorden els requisits amb el consultor, es plasmen en l'anàlisi funcional i s'entreguen durant la PAC2.
- Disseny del sistema: es defineix l'estructura global que tindrà en sistema, es descriu cada una de les seves parts i com es relacionaran. Tot això queda plasmat en el disseny tècnic que s'entrega en la PAC2.
- Codificació: s'implementa el codi font del programa que s'entregarà juntament amb la PAC3 i l'entrega final. Aquesta implementació es farà mòdul a mòdul, començant pels components obligatoris "*tfc-core*", "*tfc-boot*" i "*tfc-dependencies*" que juntament amb "*tfc-test*" en permetran provar unitariament cada una de les noves funcionalitats que anem implementant. Així, la estratègia serà definir la interfície del servei, programar un test unitari contra aquesta interfície i posteriorment implementar el servei, de manera que bona part del codi estigui provada individualment mitjançant tests unitaris que validin tant la configuració com la implementació de cada un dels serveis.
- Proves: Cal realitzar un seguit de proves abans d'entregar el codi a la PAC3 i l'entrega final.
- Verificació: El consultor i el tribunal verifiquen que el sistema es comporti tal i com diu a les especificacions.

En aquest cas, però, al no tractar-se d'una aplicació si no d'un marc de treball per a construir aplicacions, l'actor d'un hipotètic cas d'ús seria el programa mateix construït amb el framework, i per tant l'anàlisi funcional acabaria convertint-se en un disseny del programa ja que estaria descrivint les relacions entre components de software.
Per aquest motiu, a l'anàlisi funcional es descriuen els casos genèrics que el marc de treball resol ja sigui parcial o totalment des del punt de vista d'un usuari d'una aplicació framework. És a dir, a l'anàlisi funcional es troben descrits els possibles requisits que pugui tenir un projecte i que el framework ja resol, a més, els requisits descrits es estan implementats en l'aplicació de referència. Alguns d'ells son, "Autenticació d'usuari", "Elecció de l'idioma de l'aplicació", "Generació d'un informe" ...
En canvi, els requisits tècnics del marc de treball estan definits dins el disseny tècnic, on es pot veure com s'ha de comportar cada un dels mòduls i quina funcionalitat aporta al programa.

b) Metodologia per a implantar i mantenir

Amb la base que hi ha desenvolupada es podria implantar aquest marc en una organització. Per tal que això tingués èxit probablement serien necessàries una serie de millores i adaptacions, per exemple connectors amb els sistemes d'informació, desenvolupament de més mòduls, certificació del marc de treball en els servidors d'aplicacions i bases de dades on ha d'anar instal·lat, adaptació a l'aspecte corporatiu ... Seria bo que s'entres en un procés d'evolució constant de manera que no es quedés estancat la manera com l'organització construeix les noves aplicacions, i es poguessin introduir millores i correccions que repercutissin en les aplicacions ja existents.
Una metodologia que s'ajustaria a aquesta evolució i millora constant podria ser SCRUM per les seves característiques:

- S'identifiquen un conjunt de tasques no prioritzades que s'ubiquen en el "Product Backlog", es poden afegir tasques en qualsevol moment, no existeix un període tancat com en el cas del desenvolupament en cascada.

- Es realitzen cicles de 15 o 30 dies (sprints) on es planifiquen un seguit de tasques extretes del "Product Backlog" que cal realitzar, al final de cada sprint es genera un entregable, una nova versió.
- La metodologia també defineix una sèrie de rols com el "Product owner" que interacciona amb el client i defineix els requisits. L'"Scrum Master" que ajuda als equips a assolir els objectius dels sprints. Els equips de desenvolupament, en el cas del marc de treball, els desenvolupadors també haurien de poder entrar requisits tècnics, en funció de millores que puguin detectar.

1.4. Planificació del projecte

Per tal de dur a terme el treball he desglossat les accions que em caldrà dur a terme de forma ordenada.

Elecció i justificació tecnològica, estimo una setmana de temps per a poder comparar les diferents alternatives per a la capa de presentació i justificar així l'elecció tecnològica.

7 dies per a realitzar l'anàlisi funcional.

10 dies per a realitzar el disseny tècnic, estaria enllestit el dia 9 d'Abril.

De cara la PAC2 hauria d'implementar el projecte pare i alguna funcionalitat del mòdul core (càrrega de configuració i traces) i algun test per tal de poder entregar un codi que pogués ser executat i demostrés el bon funcionament de la funcionalitat implementada.

A continuació desglosso les tasques d'implementació amb una estimació per dies.

A continuació es desglossen les tasques d'implementació amb una estimació per dies.

Implementació

- framework
  - projecte pare – gestió de dependències, perfils per servidor – (2dies).
  - mòdul core (7 díes)
  - mòdul boot (2 dies)
  - implementació mecanisme de testing (2 dies)
  - mòdul persistència (4 dies)
  - mòdul web (7 dies)
  - mòdul seguretat (4 dies)
  - mòdul informes (3 dies)
  - mòdul serveis web (3 dies)
- arquetipus
  - arquetipus projecte (2 dies)
  - arquetipus mòdul serveis web (1 dia)
- implementació aplicació de referència (2 dies)

La data per acabar la implementació és el 18 de Maig, així deixa uns dies per a la memòria i la per preparar la presentació.

Redacció memòria.

Preparar presentació.

Aquestes tasques donen com a resultat el següent cronograma.

		Name	Duration	Start	Finish	Predecessors
1		**Elecció i justificació...**	**7 days?**	**16/03/12 08:00**	**22/03/12 17:00**	
2		Justificació JSF	2 days?	16/03/12 08:00	17/03/12 17:00	
3		Eleció fwk sobre JSF	5 days	18/03/12 08:00	22/03/12 17:00	2
4		Anàlisis funcional	7 days?	23/03/12 09:00	30/03/12 09:00	1
5		Disseny técnic	10 days	31/03/12 08:00	09/04/12 17:00	4
6		**Implementació fram...**	**37 days?**	**10/04/12 08:00**	**16/05/12 17:00**	5
7		projecte pare	2 days?	10/04/12 08:00	11/04/12 17:00	
8		mòdul core	7 days	12/04/12 08:00	18/04/12 17:00	7
9		mòdul boot	2 days	19/04/12 08:00	20/04/12 17:00	8
10		mecanisme testing	2 days?	21/04/12 08:00	22/04/12 17:00	9
11		mòdul persistència	4 days?	23/04/12 08:00	26/04/12 17:00	9;10
12		mòdul web	7 days?	27/04/12 08:00	03/05/12 17:00	9;11
13		mòdul seguretat	4 days?	04/05/12 08:00	07/05/12 17:00	12
14		mòdul reporting	3 days?	08/05/12 08:00	10/05/12 17:00	12;13
15		mòdul web services	3 days	10/05/12 17:00	13/05/12 17:00	12
16		arquetipus maven	2 days?	14/05/12 08:00	15/05/12 17:00	14;15
17		arquetipus mòdul w...	1 day?	16/05/12 08:00	16/05/12 17:00	16
18		Implementació aplicac...	2 days?	16/05/12 17:00	18/05/12 17:00	12
19		Memòria	25 days?	18/05/12 17:00	12/06/12 17:00	
20		Presentació	6 days	13/06/12 08:00	18/06/12 17:00	19

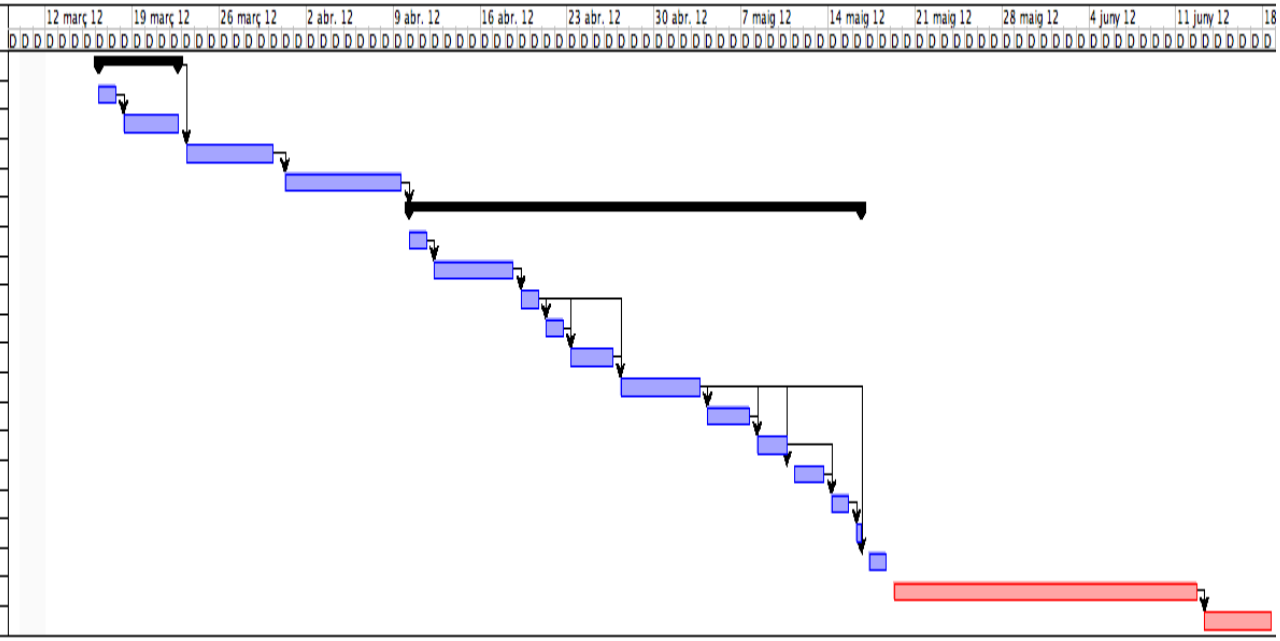

Figura 1: Cronograma planificació

1.5. Productes obtinguts

Els productes obtinguts amb aquest TFC son:

- El framework (tfc.zip). El contingut és els mòduls i arquetipus preparats per a compilar amb maven (executant *mvn clean install*, en primer lloc del mòdul tfc-dependencies i després del projecte arrel).

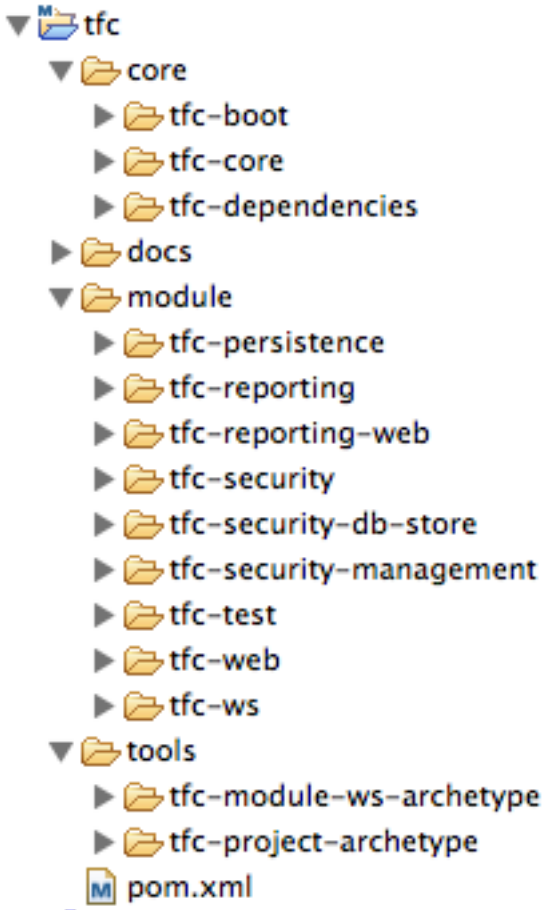

Figura 2: Contingut paquet framework

- Aplicació de referència (demo-tfc.zip) que es pot empaquetar executant "*clean package*" al projecte arrel.

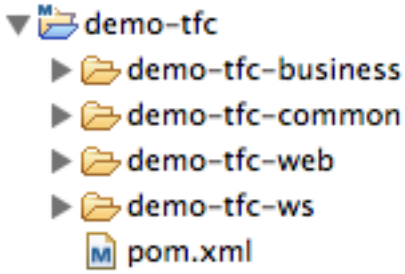

Figura 3: Contingut paquet demo

- Memòria scalvetr_tfc_memoria.pdf, que és el document que està llegint ara mateix.
- Presentació scalvetr_tfc_presentacio.ppt, presentació del treball.

1.6. Descripció

El propers capítols son una descripció del producte final on es veu com resol cada un dels objectius proposats. Es descriu les tecnologies utilitzades, l'arquitectura d'execució, l'arquitectura de desenvolupament, un disseny de cada un dels mòduls. Finalment, es mostra l'aplicació de referència com a exemple d'aplicació desenvolupada amb framework on es pot veure exactament en quins punts o quins mecanismes té l'aplicació per a utilitzar les funcionalitats del marc de treball, ja siguin serveis, plantilles de presentació, manteniments complerts (com els de seguretat) o funcionalitats transversals, com ara la transaccionalitat declarativa.
El context tecnològic sobre el que està desenvolupat el framework és molt ampli i costaria molt descriure'l amb detall, per tant la descripció s'intentarà centrar en allò que és propi i esporàdicament farà alguna pinzellada a alguna funcionalitat rellevant proporcionada per el marc tecnològic de base.

2. Descripció de l'arquitectura

En aquest apartat es dona una visió general de l'arquitectura, s'expliquen les tecnologies principals utilitzades, es descriu l'arquitectura d'execució per entendre com es comporta una aplicació, el mecanismes, capes, el seu model de desplegament. També es descriu l'arquitectura de desenvolupament, és a dir les eines que proporciona al desenvolupador per a construir una aplicació.

2.1. Base tecnològica

a) Spring

Spring és un contenidor d'objectes, per tant controla el cicle de vida d'aquestos. És a dir, s'encarrega de crear-los, inicialitzar-los i destruir-los. Això permet implementar patrons com *Factoria d'objectes* o *Sigleton* de forma còmode.

Una de les característiques més apreciades d'aquest *framework* és la injecció de dependències mitjançant el patró Inversion of Control (IoC). Consisteix en que quant un objecte necessita un altre per realitzar una tasca, delega al contenidor en temps d'execució la recuperació d'una instància d'aquesta classe. Això permet un desacoblament entre mòduls ja que no cal conèixer com es construeixen objectes d'altre mòduls, ni quines son les implementacions concretes (que les calcula el contenidor en temps d'execució).

A més, ofereix altres característiques avançades per a facilitar la Programació Orientada a Aspectes (AOP), gestió de transaccions, seguretat, accés a a dades, testeig...

Les principals característiques que ofereix són les següents:

- Injecció de dependències seguin el patró IoC.
- Programació orientada a aspectes.
- Transaccionalitat declarativa.
- Seguretat (amb spring security).
- Integració (Serveix Web, LDAP, planificadors, etc.)

b) RichFaces

RichFaces (RF4) és un *framework* construït sobre la base de Java Server Faces JSF 2, això li dona una capacitat per a construir interfícies d'usuari riques i ben enllaçades amb la cap de control, a nivell d'events i enllaç de dades. Li proporciona també un joc de components visuals bàsics que RF4 estén per a donar-li més capacitats AJAX i de recàrrega parcial de la vista.

A més, el mecanisme de plantilles Facelets 2 com a substitut de JSP, permet definir plantilles amb el mateix llenguatge que es defineix la vista pròpiament dita, és a dir XHTML i també permet definir *tags* mitjançant la composició de components JSF.

En resum, aquesta combinació de tecnologies permet implementar a les aplicacions les següents funcionalitats:

- Intercanvi de dades transparent entre la vista i el controlador.
- Validació de dades integrat amb l'estàndard bean-validator.
- Internacionalització.
- Ampli conjunt de *tags* de presentació.
- Integració amb AJAX.

- Composició de vistes i plantilles cross-aplication.

c) Hibernate

Hibernate és una llibreria que s'ocupa del mapeig entre el model relacional i el model orientat a objectes. Resol de forma elegant els tradicional problemes d'impedància entre aquestos dos models i s'integra perfectament amb Spring.

Les principals característiques que ofereix son les següents:

- ORM (mapeig d'objectes al model relacional)
- Independent de la base de dades.
- memòria cau de segon nivell.
- Model d'esdeveniment i escoltadors
- Suporta múltiples orígens de dades.

d) CXF

Framework per a exportar i consumir serveis web, suporta els estàndards JAXB i JAXWS i implementa moltes de las especificacions definides per OASIS (per exemple *WS-Security*, *WS-Policy*, *WS-Addressing*…), també permet la implementació de serveis REST, suportant l'estàndard JAXRS.

Les principals característiques que ofereix son les següents:

- Suport per als principals estàndards per a publicar serveis, tant basats en *XML* (JAXWS) com serveis *REST* (JAXRS).
- Abstracció de la capa de transport amb suport per a múltiples transport com ara HTTP, JMS, SMTP, TCP ... convertint-lo en un peça clau per a integrar una aplicació amb tot tipus de sistemes a l'organització.
- Proporciona una serie d'interceptors i la possibilitat de programar-ne de propis que actuen en la cadena d'entrada (*services*) i de sortida (*clients*) de les peticions.
- Arquitectura extensible mitjançant listeners i features que permeten configurar tot tipus de funcionalitats de manera global o específica per a un servei.

e) JasperReports

Es una llibreria para la generació d'informes en molts formats. Proporciona una eina visual iReport per a dissenyar les plantilles, s'integra amb les fonts de dades de la nostra aplicació.

f) Maven

Eina per a construir empaquetar i testejar els projectes *java*. Proporciona una manera homogènia d'estructurar els projectes i multitud de connectors per a fer infinitat de tasques. Està suportat per gran part dels entorns de desenvolupament, així com per eines d'integració contínua,.

Permet construir diferents plantilles per a projectes i mòduls que son bastant parametritzables proporcionant un bon punt d'arrencada per a començar a desenvolupar.

La seva característica més interessant és la gestió de dependències, té moltes opcions i és molt

flexible facilitant el desplegament en diferents servidors.

2.2. Arquitectura d'execució

Anem a definir clarament com organitza el *framawork* totes aquestes tecnologies que hem vist, quins elements ens trobarem en l'aplicació, quin tipus d'*API*s, quin comportament podem esperar, com s'accedeix a aquestes *API*s, quina estructura tindrà l'aplicació, quins rols desenvoluparà cada classe.

a) Estructura del framework

TFC Framework proporciona tota una sèrie de funcionalitats a les aplicacions, tal i com ja anirem veient, existeix unes funcionalitats bàsiques i unes funcionalitats que opcionalment les aplicacions poden incorporar en forma de mòduls.

Els mòduls que trobarem disponibles en una aplicació *framework* son els següents:

Figura 1: Diagrama de components

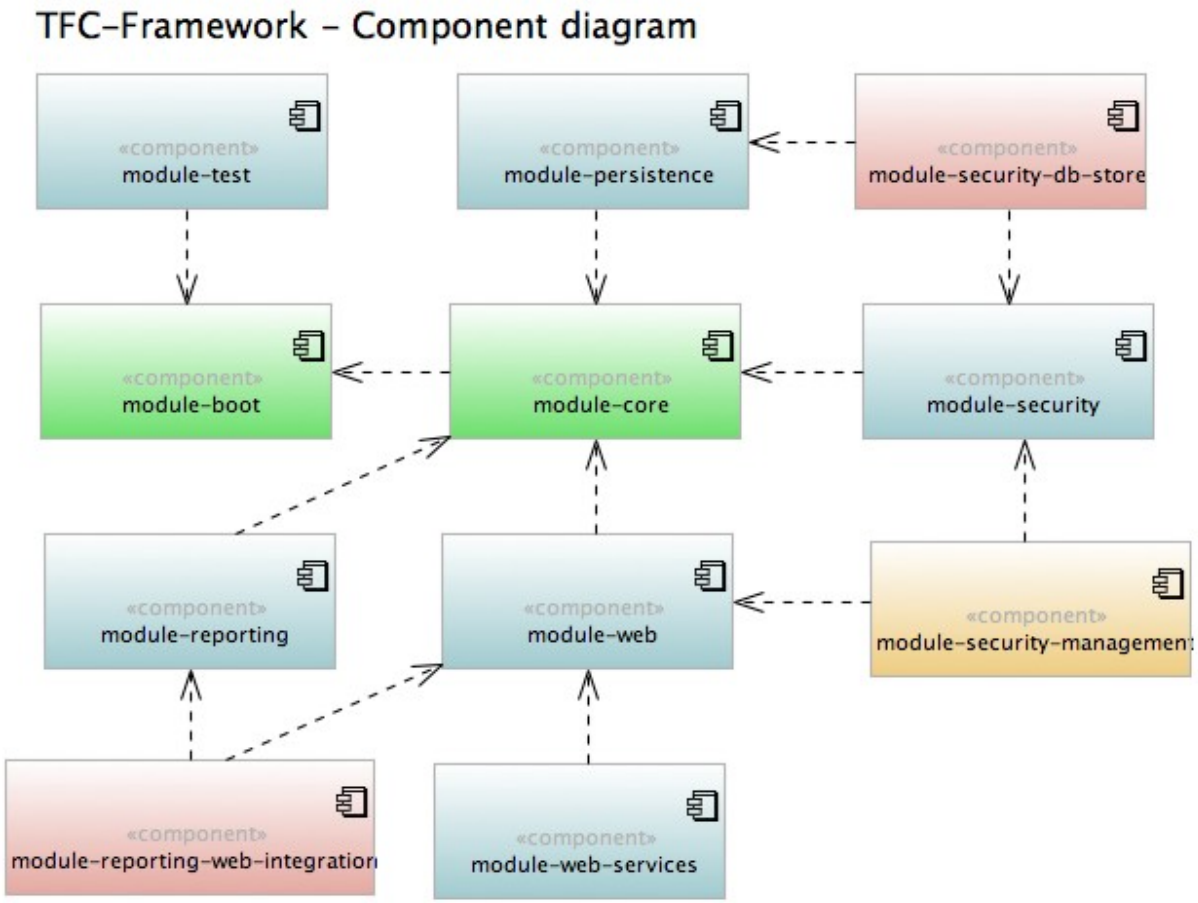

Figura 4: Diagrama de components

Els únic mòduls, obligatoris en tota aplicació TFC Framework són el module-boot i el module-core, que s'encarreguen de gestionar el comportament bàsic. Les aplicacions web, també requereixen el module-web. S'ha tingut en compte que es pugui fer servir el mòdul de generació d'informes sense necessitat de tenir el mòdul web (per exemple un test que generi un informe o una aplicació que no

requereixi visualització d'informes, simplement els generi i emmagatzemi) separant la integració web en el mòdul module-reporting-web-integration. El mòdul de seguretat també és independent d'on s'emmagatzema la informació d'usuaris, grups, rols ... però proporciona una implementació en base de dades amb el mòdul module-security-db-store.

El mòdul module-security-management empaqueta un manteniment complert, per tant té una estructura semblant a un mòdul d'una aplicació, a dins hi trobem el objectes de negoci, la capa de control, les vistes i la configuració necessària per a poder ser incrustat dins una aplicació i proporcionar-li la funcionalitat complerta de la gestió d'usuaris.

b) Estructura dels mòduls

Més endavant es detalla quins són aquestos mòduls i com estan implementats i quina funcionalitat proporciona cada un. Aquí ens ocuparem de descriure de forma genèrica quina estructura interna tenen.

Veiem l'estructura d'un dels mòduls:

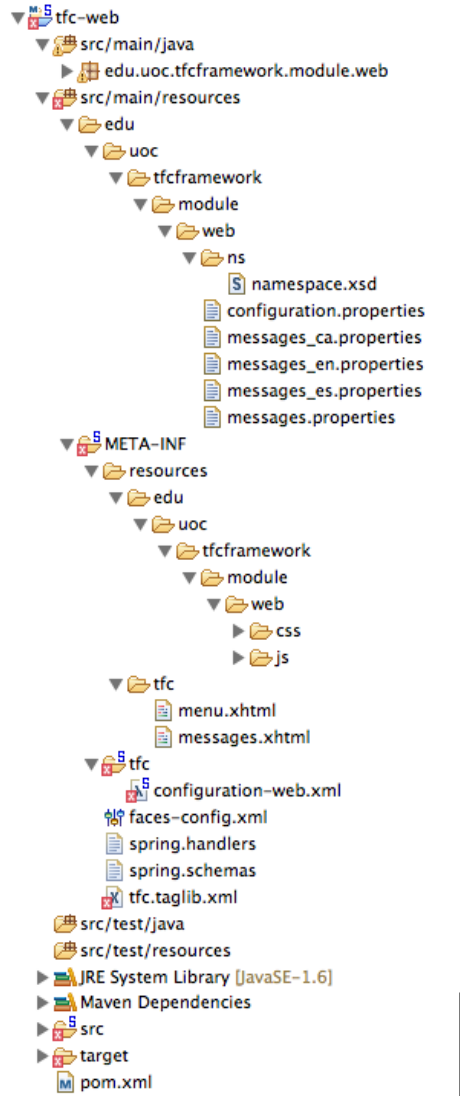

Figura 5: Estructura d'un mòdul

Identifiquem els següents tipus d'elements.

Codi java

En la carpeta "src/main/java" es troba ubicat el codi java del mòdul. Els tipus de classe poden ser ben diversos, des de components interns, extensions d'alguna de les llibreries utilitzades, normalment, però, dins de cada mòdul hi ha un o més serveis, és a dir objectes amb una *API* que poden ser utilitzats per l'aplicació.

El diagrama de classes d'un mòdul podria ser aquest:

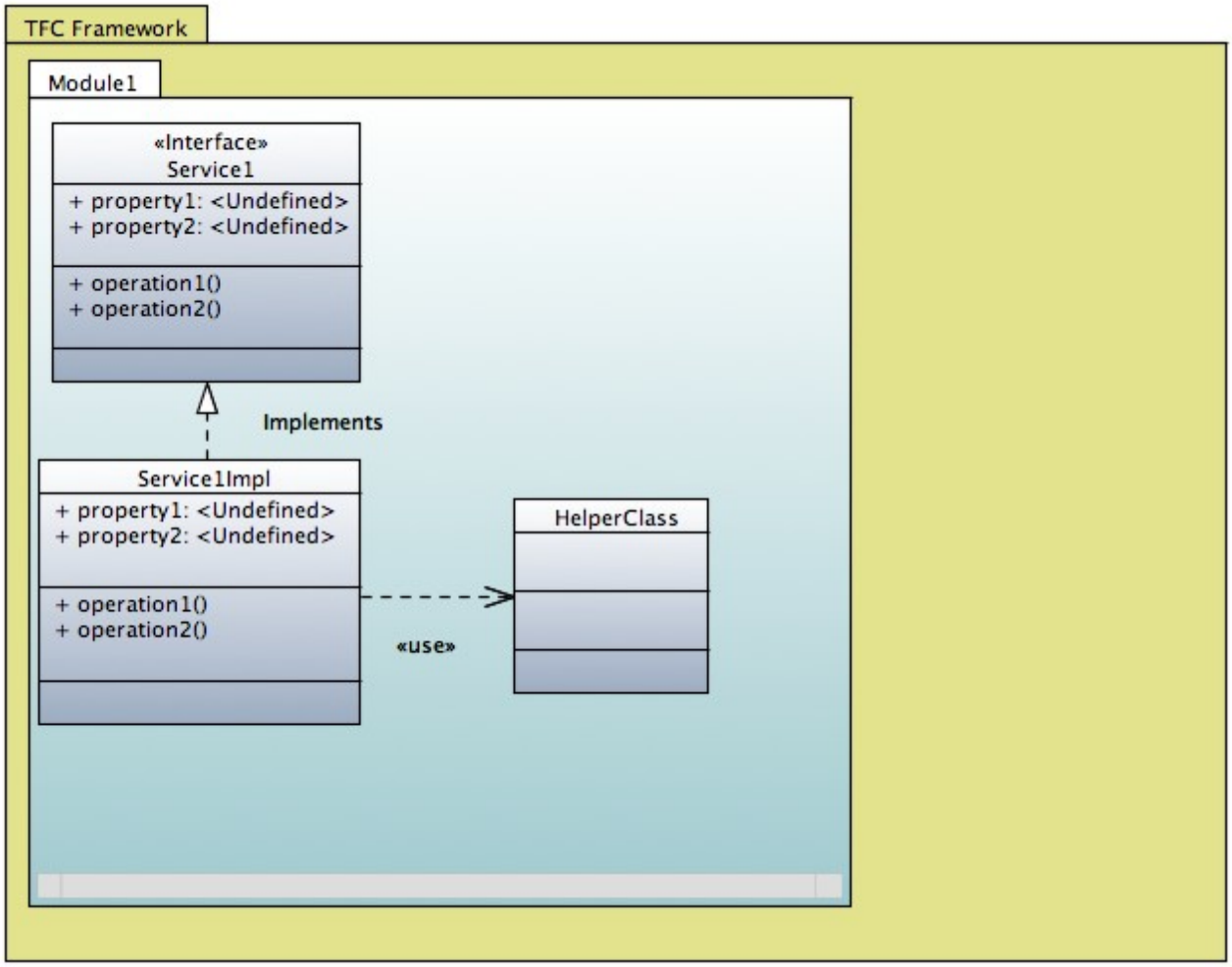

Figura 6: Diagrama de classes d'un mòdul

És a dir, una o unes interfícies de servei, la seva implementació, que es pot ajudar d'altres classes dins el mòdul o fins i tot d'altres servies d'altres mòduls.

Les classes servei es distingeixen per tenir una anotació *edu.uoc.tfcframework.core.stereotype.Service*. Així, la definició d'un servei podria ser:

```
/**
 * {@link ConfigurationService} implementation.
 *
 * @author Sergi Calvet <scalvetr@uoc.edu>
```

```
 */
@Service
public class ConfigurationServiceImpl implements ConfigurationService {
```

Definició d'un espai de nom de configuració

Un dels mecanismes que pot proporcionar un mòdul per configurar-se és proporcionar un espai de noms. Aquest, pot ser utilitzat en els fitxers de configuració de les aplicacions. (Veure *Spring Extensible XML authoring* a la bibliografia).

Els fitxers implicats son:

- src/main/resources/<<paquet_del_modul>>/ns/namespace.xsd es troben definits els elements de la configuració.
- src/main/resources/META-INF/spring.schemas indica una ubicació virtual per la ubicació física de l'esquema namespace.xsd.
- src/main/resources/META-INF/spring.handler vincula la ubicació virtual a la classe que parsejarà la configuració.

Precàrrega de configuració

El mòduls poden carregar una certa configuració pel sol fer de ser importats, per exemple poden publicar un servei en el contenidor d'objectes.

Els fitxers implicats son:

- src/main/resources/META-INF/configuration-<<nom_modul>>.xml es tracta d'un fitxer de configuració spring que es carregarà pel sol fet d'importar la dependència del mòdul.
- src/main/resources/edu/uoc/tfcframework/module/<<nom_modul>>/configuration.properties totes aquelles propietats de configuració específiques del mòdul amb els seus valors per defecte, podran ser sobreescrites per l'aplicació en el seu fitxer tfc-config.properties.
- Per tal que siguin carregats aquestos valors per defecte és necessària la següent configuració en el fitxer META-INF/configuration-<<nom_modul>>.xml.

```
<!-- setup module configuration -->
<tfc-core:configurationFile
path="classpath:edu/uoc/tfcframework/module/reporting/configuration.properties" />
```

Missatges multiidioma

El mòduls poden carregar una sèrie de missatges multi-idioma que els podran utilitzar a l'hora de llençar excepcions o per treure missatges per pantalla.

Els fitxers implicats son:

- src/main/resources/edu/uoc/tfcframework/module/<<nom_modul>>/messages_<<idioma>>.properties cal un fitxer d'aquests per cada idioma que es vulgui suportar.
- Per tal que aquestos missatges puguin ser utilitzats cal que siguin carregat amb la següent configuració en el fitxer META-INF/configuration-<<nom_modul>>.xml.

```
<!-- Loads i18n messages -->
<tfc-core:loadResourceBundle
     baseName="edu.uoc.tfcframework.module.<<nom_modul>>.messages" />
```

Llibreries de tag per a la capa de presentació

El mòduls poden definir tota mena de components visuals.

Els fitxers implicats son:

- src/main/resources/META-INF/<<nom-taglib>>.taglib.xml fitxer taglib de Facelets 2.0.
- src/main/resources/META-INF/<<nom-taglib>>/*.xhtml tot tipus de "Composite Components" (veure a la bibliografia).

Recursos i configuració web

El mòduls poden empaquetar imatges, css, javascript, plantilles... es a dir tota mena de recursos web.

Els fitxers implicats son:

- src/main/resources/META-INF/resources/ tots els fitxers ubicats en alguna subcarpeta podrà ser accedit des de la vista amb l'expressió "#{carpteta:recurs}".
- src/main/resources/META-INF/faces-config.xml en aquest fitxer es pot posar tot tipus de configuració JSF que serà importada pel projecte.

Tests del mòdul

Els mòduls poden incloure una sèrie de tests, aquests no seran empaquetats, la seva funció no és en temps d'execució si no en temps de desenvolupament.

Els fitxers implicats son:

- src/test/java/ codi java dels tests.
- src/main/resources recursos necessaris per als tests.

c) Contenidor d'objectes

Tots els objectes, ja siguin de l'aplicació o del *framework* estan gestionats per spring. Ja s'ha explicat que es tracta d'un contenidor que en controla el cicle de vida i permet que puguin ser *injectats* en el nostre codi. Per tal que la càrrega sigui ordenada i els objectes de l'aplicació no puguin interferir en les classes del *framework* aquest s'organitza en una sèrie de contexts:

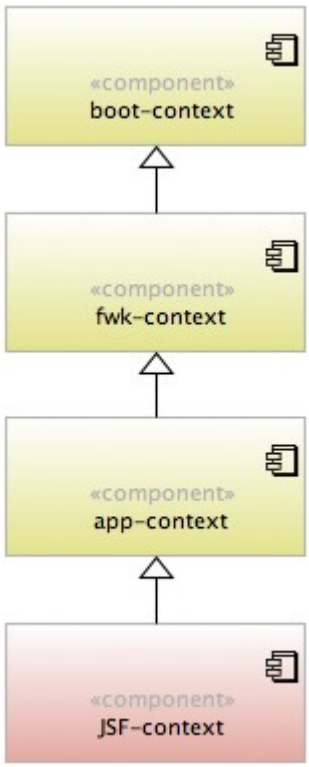

Figura 7: Contexts d'spring

boot-context

Es carrega en primer lloc i conté informació de l'entorn necessària per a inicialitzar els altres contexts.

fwk-context

Conté tots els objectes propis del *framework* inicialitzats sense que l'aplicació pugui interferir provocant errors difícils de diagnosticar (mitjançant aspectes, *post-processors*, *listeners* ...).

app-context

És un context que estén fwk-context i que per tant té accés a tots els objectes definits en aquest. Es carrega amb posterioritat, un cop el *framework* ha estat inicialitzat, amb els objectes i fitxers de configuració que es troben dins la pròpia aplicació.

JSF-context

No és un context spring, és el context que aixeca JSF. En aquest cas està configurat per que delegui totes les cerques a app-context, amb això s'aconsegueix que JSF pugui accedir als objectes de la nostra aplicació.

d) Capes

A l'hora de desenvolupar una aplicació és important definir una sèrie de capes. És clar que tècnicament és possible programar un component JSF que accedeixi directament a base de dades, però de cara a fer un codi més mantenible, on sigui senzill reaprofitar, és imprescindible separar la

implementació en diferents classes cada una desenvolupant un rol diferent.

L'exemple del component que accedeix a base de dades, s'hauria de programar de la següent manera:

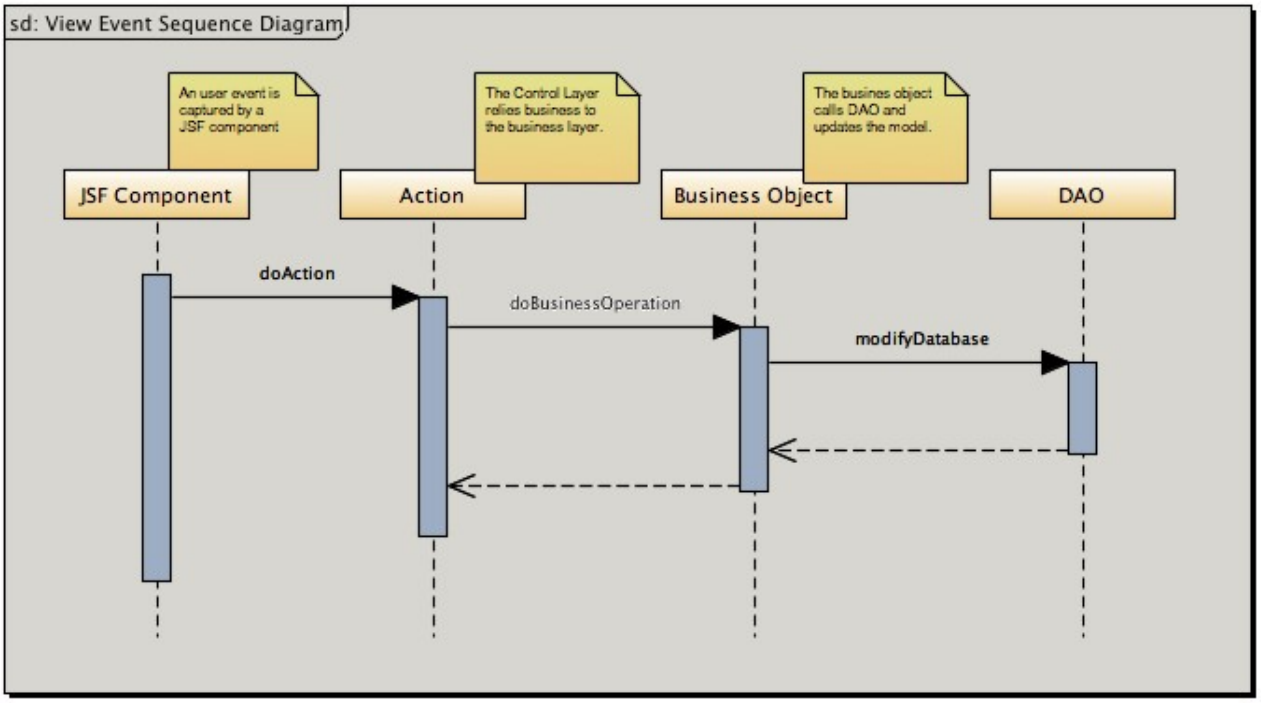

Figura 8: Capes de l'aplicació

En el cas del *framework*, això és més importat per que afecta a aspectes tant importants com la gestió d'excepcions, la transaccionalitat entre d'altres. En el cas de TFC Framework es distingeix les funcions d'una classe mitjançant anotacions, que en determinen el rol. Apart, una bona organització de paquets i mòduls també ajuda a la legibilitat del codi. Per tant, les eines del *framework*, que es veuran més endavant proporcionen una estructura de paquets per defecte, una organització del projecte en mòduls i exemples d'esquelets de classes correctament esterotipades.

Els tipus de classe que es distingeixen son:

- Model: Son objectes comuns a tota l'aplicació, totes les capes hi tenen accés, tan la vista com el negoci com la capa d'accés a dades, actuen com a contenidors de dades, sense operacions. En cas que s'hagin de persistir, han d'estar anotades amb anotacions de JPA que especifiquen el mapeig amb el model relacional, per exemple *javax.persistence.Entity*.
- Action: implementa les accions desencadenades per l'usuari. Pot estar associada a components de presentació i a *view helpers*, per tal de donar suport a la vista. Delega totes els operacions de negoci en la capa de negoci. Cal anotar-la com a *edu.uoc.tfcframework.core.stereotype.Action.*
- BusinessObject: implementa les operacions de negoci que acostumen a actuar sobre el model, aquest objectes són marcats pel *framework* com a transaccionals i els seus mètodes són executats de manera atòmica, veurem en la descripció del mòdul core com configurar aquest comportament. Cal anotar-la com a *edu.uoc.tfcframework.core.stereotype.BusinessObject*.
- Dao: implementa l'accés a dades, consultes, insercions... Cal anotar-la com a

edu.uoc.tfcframework.core.stereotype.Dao.

- WebService: implementa un servei web exposat, cal anotar-la amb les anotacions JAXWS i JAXB per definir exactament el contracte que hauran d'implementar els clients, per exemple *javax.jws.WebService*.
- Component: son objectes interns en qualsevol de les capes, que ajuden a... Cal anotar-la com a *edu.uoc.tfcframework.core.stereotype.Component.*

Els paquets que consta la aplicació son:

<<paquetBase>>.web.action: per a les accions.

<<paquetBase>>.model: per a les classes del model.

<<paquetBase>>.business: per a les interfícies de negoci.

<<paquetBase>>.business.impl: per a les implementacions del negoci.

<<paquetBase>>.dao: per a les classes d'accés a dades.

<<paquetBase>>.service.<<nomServei>>: per a les interfícies d'un servei web.

<<paquetBase>>.service.<<nomServei>>.impl: per a les implementacions d'un servei web.

Poden existir més paquets en el cas que es facin servir altres classes per donar suport a els implementacions, cal però especificar en el nom del paquet la capa a la que pertanyen, per exemple:

- Un conversor de dades de la capa de presentació, estaria en el paquet *<<paquetBase>>.web.converter* i implementaria la interfície *javax.faces.convert.Converter*.
- Un adaptador d'un objecte de model a un representació de la vista, estaria en el paquet *<<paquetBase>>.web.adapter* i estaria anotat, com a tota classe de suport com a *edu.uoc.tfcframework.core.stereotype.Component*.

e) Model de desplegament

Aquesta separació lògica en paquets, no impedeix, però que des d'un objecte de negoci es pugui fer referència a una acció de la vista. Per tant, per evitar errors s'intensifica en una separació física en mòduls per tal d'assegurar que aquest separació en capes s'ha fet correctament.

L'organització bàsica del projecte és aquesta:

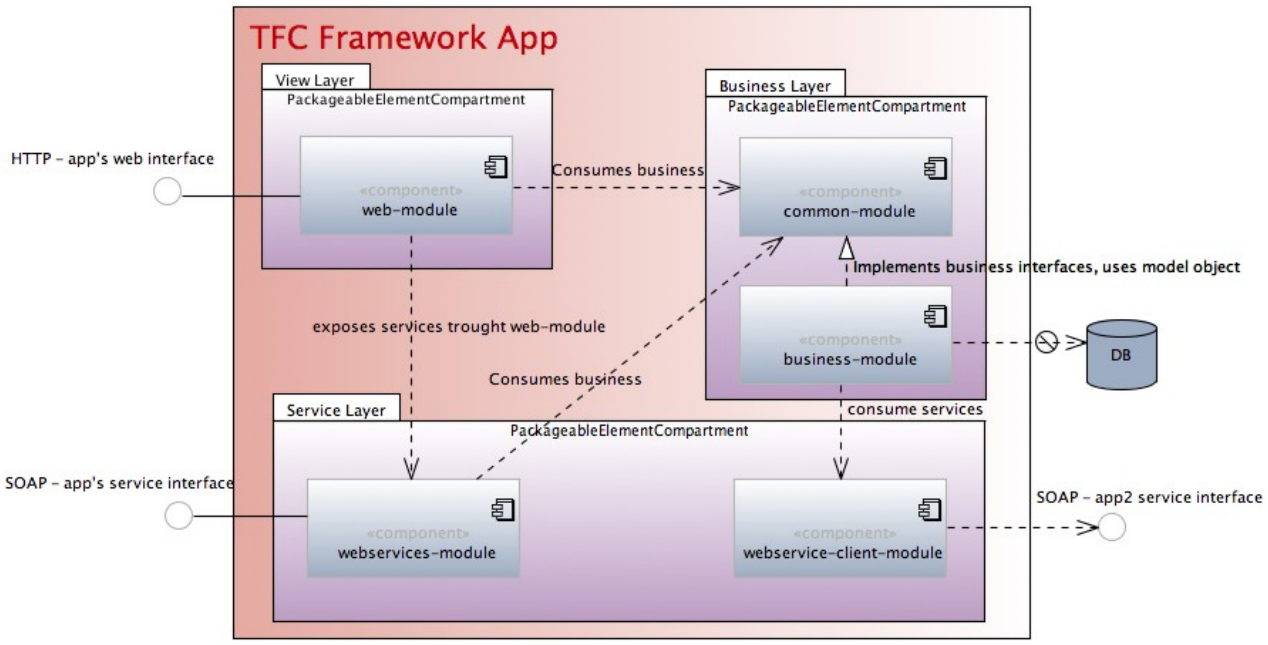

Figura 9: Diagrama de desplegament

Així, un servei web o un objecte de negoci no poden fer referència a objectes de presentació, i un client de serveis web no té accés al model ni al negoci de l'aplicació si no que son utilitzats per aquest.

common-module

Es tracta d'una llibreria (arxiu jar), amb totes les classes comuns a totes les capes, és a dir les que es troben en els paquets *<<paquetBase>>.business* i *<<paquetBase>>.model*. Per tant, les interfícies de negoci i els objectes de model.

web-module

Es tracta d'un mòdul web (arxiu war), té els fitxers de configuració propis (web.xml, faces-config.xml), les pàgines *.xhtml, i les classes dels paquets *<<paquetBase>>.web.**. Accedeix ales interfícies de negoci i al model important el mòdul common.

business-module

Es tracta d'una llibreria (arxiu jar), amb tots els fitxers de configuració i les classes que intervenen en el negoci, es a dir les que estan en el paquet *<<paquetBase>>.business.impl* i els fitxers de configuració spring ubicats a la carpeta *configuration* amb el nom *tfc-*.xml* que registren aquestes implementacions en el contenidor d'objectes.

El propi *framework* proporciona una implementació bàsica i genèrica de l'accés a dades que pot ser utilitzada per el mòdul de negoci. En el cas que l'aplicació requereixi realitzar consultes complicades que involucrin molt de codi d'accés a dades, caldria estendre aquest Dao genèric i crear un nou mòdul dao-module on ubicar-lo, amb el paquet *<<paquetBase>>.dao*.

webservices-module

Es tracta d'una llibreria (arxiu jar), amb totes les definicions de servei web ubicades en el paquet *<<paquetBase>>.service.<<nomServei>>*, les seves implementacions ubicades en el paquet

<<paquetBase>>.service.<<nomServei>>.impl i els fitxers de configuració que registren els endpoints per tal de ser publicats pel mòdul web. Depèn del mòdul common per a accedir al negoci de l'aplicació.

webservices-client-module

Es tracta d'una llibreria (arxiu jar), amb la implementació del client de servei web ubicades en el paquet *<<paquetBase>>.service.<<nomServei>>.client*, i els fitxers de configuració que registren els clients en el contenidor d'objecte. no depèn de cap altre mòdul ja que se centra en la tasca de consumir el servei web.

f) Injecció de dependències

Tenint en compte la jerarquia de contextos i les relacions de dependència que hi ha entre mòduls es pot deduir el tipus de dependència que hi pot haver entre els diferents objectes que hi ha registrats en els contexts de l'aplicació (un objecte d'aplicació pot requerir un servei del *framework*, però no a l'inrevés, una acció web pot requerir un objecte de negoci, però no a l'inrevés). Doncs sense perdre la vista aquestes regles, el contenidor d'objectes dona un mecanisme per associar els diferents components que implementen l'aplicació. Aquest és la injecció.

A continuació tenim un exemple de com des de la vista s'accedeix al negoci per a modificar un objecte del model.

```
package edu.uoc.tfcframework.demo.web.action;

import javax.inject.Inject;

import edu.uoc.tfcframework.core.stereotype.Action;
import edu.uoc.tfcframework.demo.business.UserBO;
import edu.uoc.tfcframework.demo.model.User;

/**
 * Controls all UI events related with model object {@link User}. Most
 * of them comes from the views "/pages/user/list" and
 * "/pages/user/edit".
 *
 * @author Sergi Calvet <scalvetr@uoc.edu>
 */
@Action 1
public class UserAction {
     private @Inject UserBO userBO; 2

     /**
      * action called from "/pages/user/edit.xhtml", returns to the
      * list.
      * @param user
      */
     public String save(User user) {
          userBO.save(user);
```

```
            return "/pages/user/list";
        }
}
```

1: anotem la classe com a *Action*.

2: injectem l'objecte de negoci que necessitem.

El contenidor d'objectes cerca un objecte registrat que implementi la interfície UserBO (probablement UserBOImpl) in ens inicialitza amb aquest objecte la propietat userBO.

2.3. Arquitectura de desenvolupament

TFC Framework proporciona una sèrie d'eines que faciliten la creació i desplegament de projectes. Aquestes eines estan basades en maven, per tant a l'hora d'elegir un entorn de desenvolupament s'ha de tenir en compte que estigui integrat amb maven. Un opció lliure i que funciona és "Eclipse IDE for Java EE Developers" (disponible a http://www.eclipse.org/downloads/) afegint-li el *plugin* m2e que es pot obtenir fàcilment des de l'Eclipse Marketplace tal i com es veu a la captura següent.

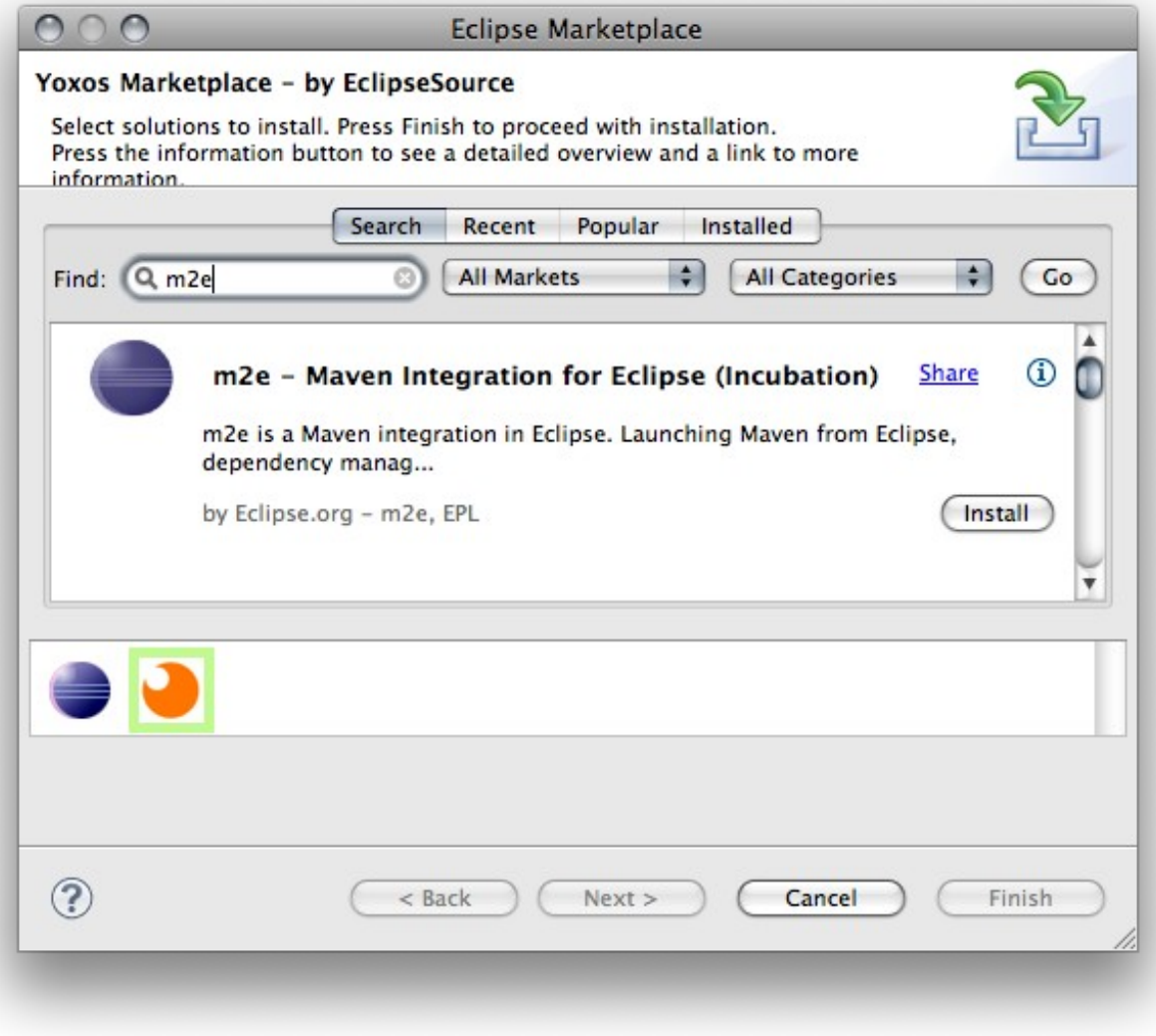

Figura 10: Instal·lació connector m2e

a) Estructura d'un projecte

Com que l'eina que fem servir en temps de desenvolupament, per construir, provar, empaquetar i desplegar és al maven cal que l'estructura interna dels mòduls s'ajusti a unes normes.

Descriptor: Es tracta d'un fitxer anomenat pom.xml ubicat a l'arrel de cada un del mòduls. Es descriu detalladament a (http://maven.apache.org/guides/introduction/introduction-to-the-pom.html).

Carpeta fonts java: Ubicada a src/main/java és on s'ubiquen tots els fitxers java que cal compilar, desplegar i empaquetar.

Carpeta de recursos: Ubicada a src/main/resources és on s'ubiquen tots els fitxers de configuració que cal desplegar i empaquetar.

Carpeta fonts java de test: Ubicada a src/test/java és on s'ubiquen tots els fitxers java que cal compilar per tal d'executar els tests.

Carpeta de recursos de test: Ubicada a src/test/resources és on s'ubiquen tots els fitxers de configuració necessaris per a executar els tests.

Carpeta de recursos web: Ubicada a src/main/webapp és on s'ubiquen tots els fitxers propis d'un mòdul web, és a dir les interfícies d'usuari, el descriptor web.xml, el descriptor faces-config.xml.

Per tant l'estructura d'un projecte mínim queda així:

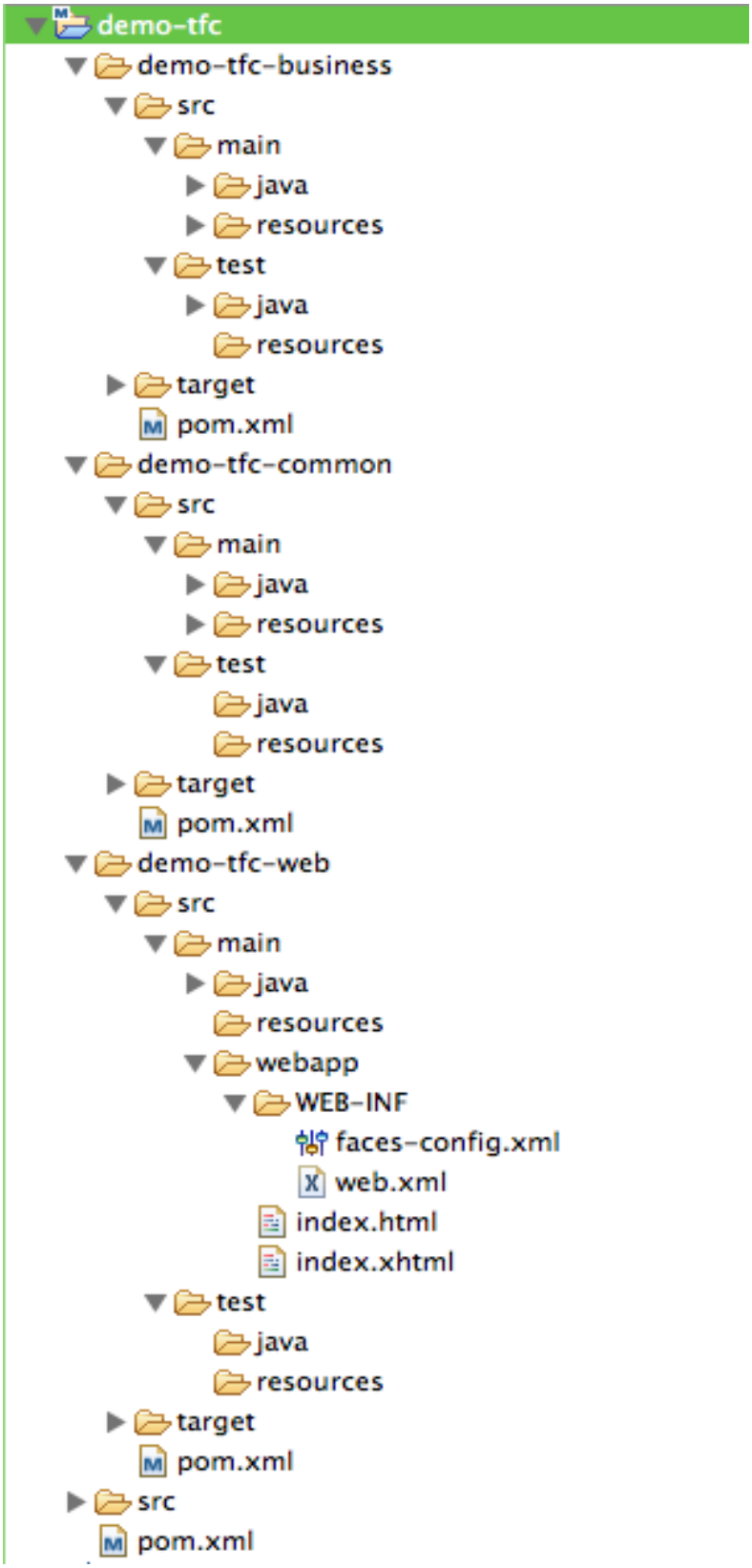

Figura 11: Estructura d'un projecte TFC

b) Arquetipus projecte nou

Per tal de començar una nova aplicació ja configurada, llesta per desplegar i arrencar en un servidor, es proporciona un arquetipus basat en maven, que crea l'estructura bàsica de mòduls i paquets i els fitxers de configuració que ja hem explicat.

L'arquetipus s'anomena "tfc-project-archetype" i es pot utilitzar des de línia de comandes mitjançant la comanda:

```
mvn archetype:generate -DarchetypeArtifactId=tfc-project-archetype
-DarchetypeGroupId=edu.uoc.tcfframework.tools -DDarchetypeVersion=1.0.0
-DgroupId=edu.uoc.samples -DartifactId=test -Dversion=0.0.1 -Dpackage=edu.uoc.samples.test
```

La implementació es basa en el maven-archetype-plugin (http://maven.apache.org/archetype/maven-archetype-plugin/). En aquest cas, com que es tracta d'un multi-module project, caldrà fer-ho tal i com s'explica aquí.

c) Arquetipus mòdul serveis web

Així mateix, per tal d'afegir un mòdul de serveis a una aplicació existent, existeix un arquetipus anomenat "tfc-module-ws-archetype".

d) Gestió dependències

La gestió de dependències es realitza mitjançant maven de la següent manera:

- Cada mòdul declara les seves dependències de manera que afegint el mòdul al nostre projecte ja arrosseguem les dependències necessàries per compilar i executar els testos.
- Els projectes han d'estendre el modul "module-dependencies" on es definiran una sèrie de perfils maven (profiles), per als servidors tomcat7 i jboss7. Empaquetant amb algun d'aquestos perfils activat es generarà una versió llesta per córrer en el servidor corresponent.

e) Test

Tal i com es veurà en el capítol dedicat al mòdul de test, el projectes importaran la dependència a aquest mòdul, especificant l'scope a test. Això els donarà un conjunt d'eines necessàries per a poder provar el seu negoci en la fase de test del procés de construcció del projecte maven. El mecanisme es basa en JUnit i spring-test i l'estén per tal d'aconseguir una càrrega de context *framework* tal i com si s'estigués arrencant l'aplicació.

3. Disseny mòduls

En aquest apartat s'entra en el detall de la implementació de cada un dels mòduls. Es descriuen les classes principals i la manera com aquestes interactuaran amb les tecnologies sobre les que es construeix el *framework* i amb les classes d'aplicació

3.1. Mòdul boot

El mòdul boot s'ocupa de la càrrega de l'aplicació, és utilitzat tant per el mòdul web com pel mòdul de test.

La càrrega de context és quelcom que ja fa spring, en el nostre cas tan sols necessitem modificar el

context generat (no ens importa com, pot ser en una aplicació web, que l'hagi creat el *org.springframework.web.context.ContextLoaderListener*, pot ser que en un test l'hagi creat *org.springframework.test.context.support.AnnotationConfigContextLoader*). El mòdul boot, és capaç d'agafar un context qualsevol i realitzar les següents accions.

- Crear un nou context anomenat boot-context, configurat de forma que sigui capaç de llegir informació de l'entorn i guardar-la en un objecte de tipus *java.util.Map*, registrat amb el nom de ENVIRONMENT. Aquesta informació és una clau anomenada "ENV" i una altra anomenada "DB" que estan registrats en el JNDI.
- Crear un nou context anomenat fwk-context, que estén boot-context i que per tant té accés a la informació d'aquest. Aquest context s'inicialitza activant dos perfils amb els valors de les variables ENV i DB. Això permet que la configuració pugui fer referència a aquestos perfils de la següent manera:

```
<beans profile="TEST">
     <jdbc:embedded-database id="dataSource">
          <jdbc:script location="classpath:db/create_tables.sql"/>
          <jdbc:script location="classpath:db/populate.sql"/>
     </jdbc:embedded-database>
</beans>
<beans profile="DEV,PRE,PRO">
     <jee:jndi-lookup id="dataSource" jndi-name="jdbc/datasource"/>
</beans>
```

  A més, carrega tot de fitxers de configuració del mòduls seleccionats com a dependència en l'aplicació. Això ho fa escanejant el classloader en busca de fitxers ubicats en la ruta "META-INF/tfc-fwk-context-*.xml". Això permet que els mòduls puguin registrar serveis, que després seran accessibles per l'aplicació.
- Anomena el perfil que rep com a app-context, i fa que estengui fwk-context. Els mòduls poden configurar també aquest context afegint fitxers amb el nom "META-INF/tfc-app-context-*.xml", per exemple, el mòdul web pot haver configurat un aspecte (amb AOP) que capturi les excepcions de negoci per tal que es mostrin correctament i en l'idioma que toca a l'usuari.

a) Diagrama de classes:

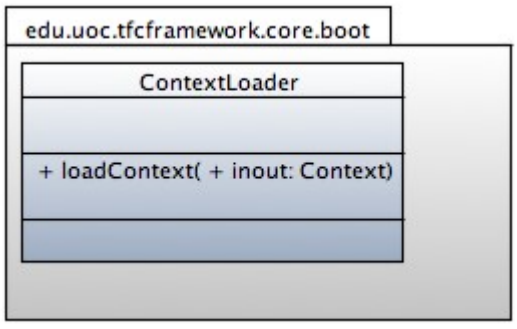

Figura 12: Diagrama de classes mòdul boot

ContextLoader: realitza les tasques que hem descrit anteriorment, rebent com a paràmetre

un context no arrencat amb tot de definicions de les classes de l'aplicació.

- *loadContext(*Context *lontext)*: carrega i inicialitza el context.

3.2. Mòdul core

El mòdul core proporciona una sèrie de funcionalitat bàsica que serà utilitzada tan per la resta de mòduls com per les aplicacions construïdes amb *framework*. Aquesta funcionalitat està dividida en diversos paquets funcionals tal i com es veu en el següent diagrama.

a) Diagrama de Classes

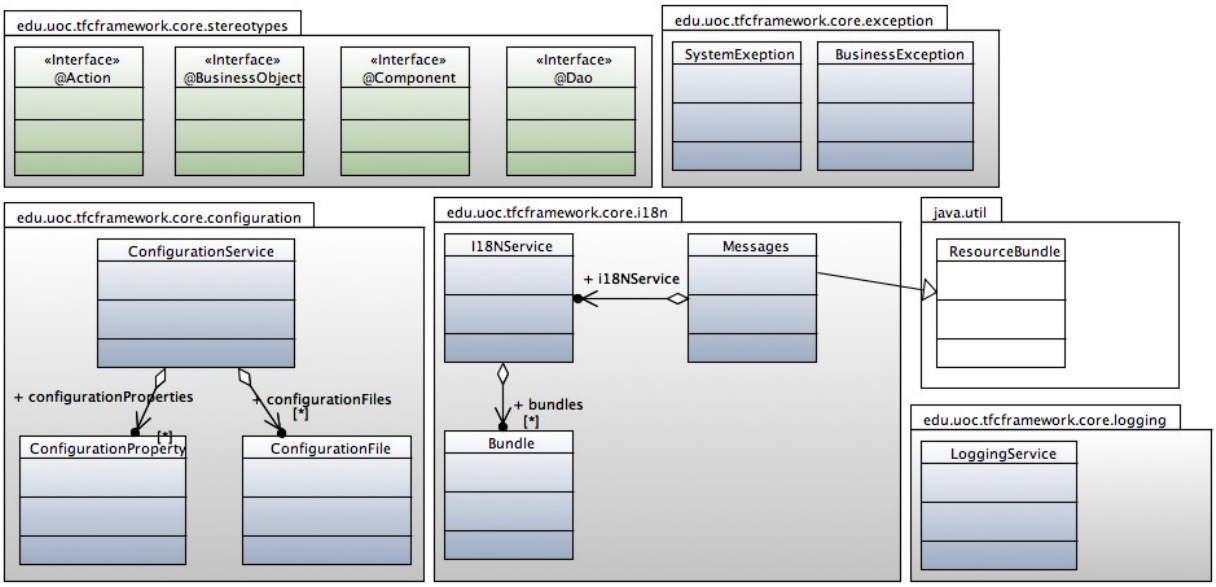

Figura 13: Diagrama de classes mòdul core

stereotypes: estereotips per a poder categoritzar les classes d'aplicació i també les de *framework*. Aquestos son Action, BusinessObject, Component i Dao, ja s'ha explicat què son cada un d'ells.

exception: defineix dos tipus d'excepcions que son les que el *framework* és capaç de gestionar.

- SystemException: és una excepció llençada pel propi *framework* quant es produeix una anomalia en l'ús d'algun dels seus serveis. Conté:
  - *message* (String, obligatori): missatge en anglès que sigui d'utilitat per un operador que el vegi en el fitxer de traces.
  - *code* (String, obligatori): codi *i18n* que identifica un missatge destinat a l'usuari, per tant, el servei que la llenci ha de registrar un Bundle amb les traduccions.
  - *arguments* (matriu d'objectes, opcional): objectes que s'utilitzen per formatar el missatge.
  - *cause* (Exception, opcional): excepció que ha causat el problema.
- BusinesException: és una excepció llençada per l'aplicació, els atributs que té són els mateixos que la SystemException.

configuration: gestiona la configuració de l'aplicació i treballa estretament amb el *org.springframework.beans.factory.config.PropertyPlaceholderConfigurer* d'Spring. S'ocupa de

carregar la configuració adient en funció de l'entorn, la fa accessible als mecanismes d'Spring i permet accedir-hi de manera programàtica.

El mecanisme de càrrega d'un fitxer de configuració té en compte els valors de context ENV i DB, poden existir variants d'un fitxer amb el prefix indicant a quin valor d'ENV i un segon prefix indicant quin valor de DB redefineixen. Per exemple un fitxer amb la ruta *classpath:configuration.properties* que tingui una especialització per l'entorn de PRE *classpath:configuration_PRE.properties* i una especialització per la base de dades ORACLE *classpath:configuration_ANY_ORACLE.properties*. Veiem que ANY actua coma comodí.

- ConfigurationFile: encapsula la informació d'accés a un fitxer de configuració. Aquesta és:
  - *path* (String, obligatori): ruta al fitxer, aquest pot estar ubicat en el classpath i la ruta haurà de començar amb el prefix "classpath:", també pot ser un URL.
  - *priority* (Integer, opcional, valor per defecte 0): nombre que indica la prioritat (com més alt més prioritari) si es carrega la mateixa clau de configuració en 2 fitxers manarà el que tingui priority més alt.
  - *optional* (Boolean, opcional): indica si cal causar error en cas que no existeixi el recurs especificat.
- ConfigurationProperty: encapsula la informació d'una propietat, amb el seu nom i valor. Aquesta és:
  - *name* (String, obligatori): nom de la propietat de configuració.
  - *value* (String, opcional, valor per defecte 0): valor de la propietat.
  - *priority* (Integer, opcional, valor per defecte 0): nombre que indica la prioritat (com més alt més prioritari) si es carrega la mateixa clau de configuració 2 cops manarà el que tingui priority més alt.
  - *optional* (Boolean, opcional): indica si cal causar error en cas que el valor sigui nul.
- ConfigurationService: s'encarrega de la carrega de la configuració cercant en el context d'spring els objectes de tipus ConfigurationFile i ConfigurationProperty.

  Atributs:
  - *configurationFiles* (Llistat de ConfigurationFile) llistat amb totes les definicions de fitxers de configuració, és carregat per el context d'spring.
  - *configurationProperties* (Llistat de ConfigurationProperties) llistat amb totes les definicions de propietats de configuració, és carregat per el context d'spring.
  - *configuration* (Mapa de tipus String) llistat amb totes les parelles clau valor, s'omple en el mètode initialitze.

  Mètodes:
  - *initialitze()*: cridat quant s'aixeca la instància (anotat com a @PostConstruct), ordena els ConfigurationFile en funció del camp priority i els carrega com a fitxers de propietats, sobreescrivint les duplicades en el mapa configuration. Posteriorment ordena els ConfigurationProperty, i els carrega també en el mapa configuration.
  - *getProperty(String key): String*: obté el valor d'una propietat de configuració concreta.
  - *GetProperties(): Map*: obté el mapa amb totes les propietats de configuració.

i18n: gestiona i centralitza tots els missatges de l'aplicació i el *java.util.Locale* de la petició actual, la resta de llibreries s'han de sincronitzar amb ell. Es pot accedir a un missatge per clau programàticament o com si es tractés un *java.util.ResourceBundle* mitjançant la classe *Messages*.

- Bundle: encapsula la informació per accedir a un conjunt de fitxers de missatges, cada un d'ells tradueix les claus a un idioma diferent. Conté:
  - *path* (String, obligatori): ruta al fitxer de missatges.
- I18NComponent: Carrega els missatges de tots els bundles registrats en el context d'Spring. Té la informació del locale actual i és capaç de proporcionar el missatge en funció d'una clau i de l'idioma actual.

  Atributs:
  - *bundles* (Llistat de Bundle) llistat amb tots els bundles registrats, és carregat per el context d'spring.
  - *defaultLocale* (*java.util.Locale*) es configura al carregar l'objecte indicant el locale per defecte.
  - *currentLocale* (*java.lang.ThreadLocal* de *java.util.Locale*) emmagatzema a nivell de fil d'execució quin és el locale actual.

  Mètodes:
  - *setDefaultLocale(*java.util.Locale *locale)*: inicialitza la localització per defecte.
  - *setCurrentLocale(*java.util.Locale *locale)*: inicialitza la localització per a la petició actual.
  - *getCurrentLocale() :* java.util.Locale: obté la localització per a la petició actual.
  - *getMessage(*String *key,* Object[] *parameters) :* String: obté el missatge formatat per la clau especificada en la localització actual.

logging: centralitza la gestió de traces, permet una configuració inicial, així com canvis en calent d'aquesta.

- LoggingService: s'encarrega de configurar les traces partint d'un fitxer de configuració logback, permet consultar in canviar la configuració en calent.

  Atributs:
  - *configurationFile* (String, obligatori) ruta al fitxer de configuració logback.

  Mètodes:
  - *getLoggers(): llistat d'* String: Obté de la *API* de logback el llistat de loggers configurats.
  - *getLevel(*String *logger) : String*: Obté el nivell per a un logger.
  - *setLevel(*String *logger,* String *level)*: canvia el nivell per a un logger determinat.

3.3. Mòdul test

El mòdul test és un conjunt de dependències i utilitats per tal de poder provar les aplicacions i el propi *framework*. No es tracta d'un mòdul pensat per ser desplegat junt amb l'aplicació, per tant s'importarà en els projecte indicant scope=test. Les utilitats que proporciona estan basades en

spring-test. A més d'ocupar-se de la càrrega del contenidor spring, i aportar dependències útils per a provar com ara el JUnit, spring-test ... tractarà d'aportar implementacions mock que facilitin el desenvolupament de test.

a) Diagrama de Classes

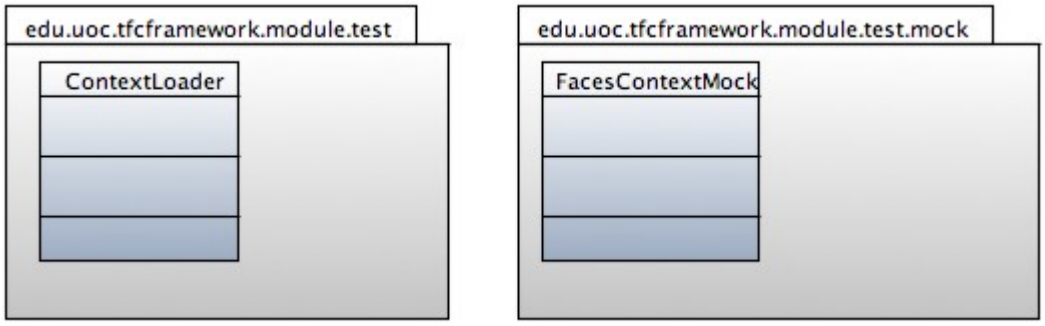

Figura 14: Diagrama de classes mòdul test

- ContextLoader: és una implementació de *org.springframework.test.context.ContextLoader* per tal de gestionar la càrrega del context delegant la major part de feina al mòdul boot.

  Mètodes:

  - *loadContext(*String[] *paths)* : *ApplicationContext*: Carrega del contexts d'acord amb les rutes a fitxers de configuració.

- FacesContextMock: és una implementació de *javax.faces.context.FacesContext* per tal de poder fer tests de la capa de presentació. Tan sols implementarà els mètodes necessaris per a tenir accés al context des de la capa de presentació.

  Mètodes sobrescrits:

  - *getExternalContext() : ExternalContext* : Obté el context inicialitzant la request, session amb les implementacions *Mock* que proporciona spring-test.
  - *addMessage(*String *clientId,* FacesMessage *message)* : Afegeix un missatge en el facesContext i el vincula a un identificador de client.
  - *getMessages(*String *clientId) : Listat de FacesMessage* : Obté els missatges vinculats a un identificador de client.
  - *getMessages() : Listat de FacesMessage* : Obté els missatges globals.
  - *setCurrentInstance(*FacesContext *context)*: inicialitza la instancia actual.

3.4. Mòdul web

El mòdul proporciona les dependències necessàries per al poder arrencar una aplicació web, integra el mòdul boot en l'aplicació, s'ocupa de mostrar els missatges correctament localitzats,

a) Diagrama de Classes

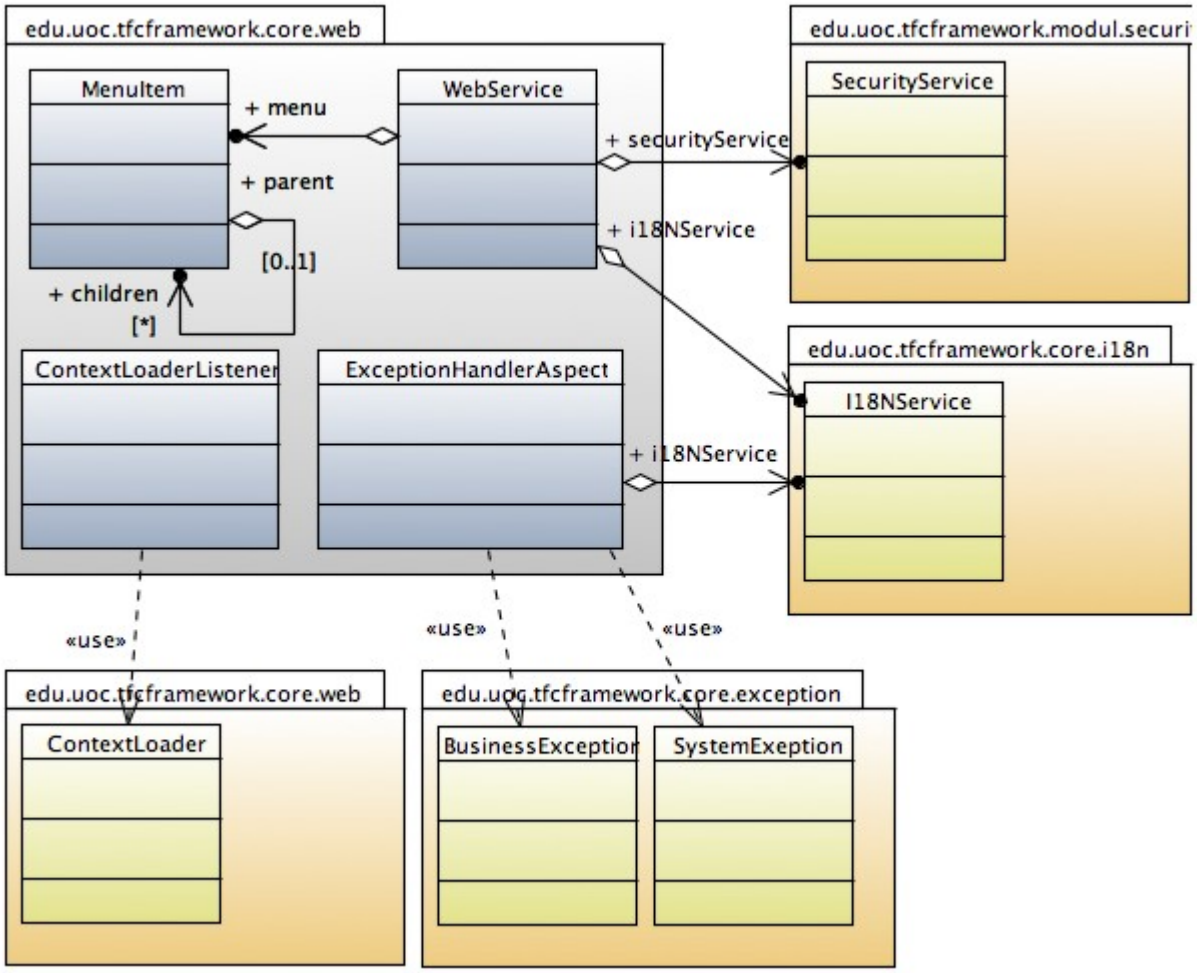

Figura 15: Diagrama de classes mòdul web

- WebService: centralitza l'accés al *framework* des de les pantalles i emmagatzema configuració l'accés a les plantilles i recursos o la configuració de menú. Implementa accions com el canvi d'idioma.

  Atributs:

  - *menu* (MenuItem, obligatori) : Node base del menú amb la jerarquia carregada d'acord amb els perfils de l'usuari.
  - *resourcePath* (String, obligatori) : ruta interna a les plantilles i les imatges i fitxers d'estils.
  - *securityService* (SecurityService, obligatori) : ens proporciona informació sobre l'usuari actual i els rols.
  - *i18nService* (I18NService, obligatori) : ens proporciona informació sobre la localització actual i ens permet canviar-la.

  Mètodes:

  - *buildMenu()*: inicialitza la configuració del menú d'acord amb el context de seguretat que obté del SecurityComponent i l'emmagatzema a sessió.

- *loadTemplate(*String *name) : URL*: obté la URL interna d'una plantilla d'acord amb la ruta resourcePath, per tal que pugui ser carregada des de la vista.
- *loadResource(*String *name) : URL*: obté la URL interna d'una recurs plantilla d'acord amb la ruta resourcePath, per tal que pugui ser carregada des de la vista.
- *changueLocale(*Locale *locale)*: canvia la localització de la petició en I18NService i emmagatzema la informació a sessió per a futures peticions. Cal que sincronitzi la localització de JSF i d'spring, per tal que tot el vegi amb el mateix idioma.
- *getLocale() :* Locale: obté el localització de sessió i si no existeix la obté de I18NService. Guarda el resultat a sessió per tal d'agilitzar futures peticions. Cal que sincronitzi la localització de JSF i d'spring, per tal que tot el vegi amb el mateix idioma.

- MenuItem: emmagatzema la informació d'un menú, tant la configuració com cada un dels menús per usuari que es construeixen tenint en compte els rols i es guarden l la sessió.

  Atributs:

  - *children* (Llistat de MenuItem, opcional) : Fills d'aquest node.
  - *roles* (String[], opcional) : llistat de rols que tenen accés a aquest punt de menú.
  - *label* (String, obligatori) : clau de recursos registrada prèviament en el I18NService amb l'etiqueta que es mostrarà a l'usuari.
  - *action* (String, opcional) : normalment present en els nodes terminals (fulles de l'arbre) és una ruta a una pàgina de l'aplicació.

- ContextLoaderListener: estén *org.springframework.web.context.ContextLoaderListener* i s'encarrega vincula la càrrega al mòdul boot.

  Mètodes:

  - *customizeContext(*ServletContext *servletContext,* ConfigurableWebApplicationContext *applicationContext)*: inicialitza applicationContext, delegant a module-boot.

- ExceptionHandlerAspect: està anotada com a aspecte (@Aspect) captura les excepcions en les classes de tipus @Action i emmagatzema el missatge de l'excepció de *framework* traduïts a l'idioma de l'usuari.

  Mètodes:

  - *handleException (*Exception *exception)* : anotat com a @AfterThrowing i fent referència al *pointcut* destinat als @Action, tradueix la excepció, la guarda en el FacesContext i imprimeix una traça, mitjançant el module-core.

3.5. Mòdul persistence

El mòdul proporciona les dependències necessàries per al poder accedir a base de dades, també proporciona el mecanisme per a que altres mòduls puguin afegir mapejos (per exemple les taules de seguretat) de manera transparent al desenvolupador. Proporciona també una implementació bàsica de *DAO* que pot ser estesa o utilitzada directament.

a) Diagrama de Classes

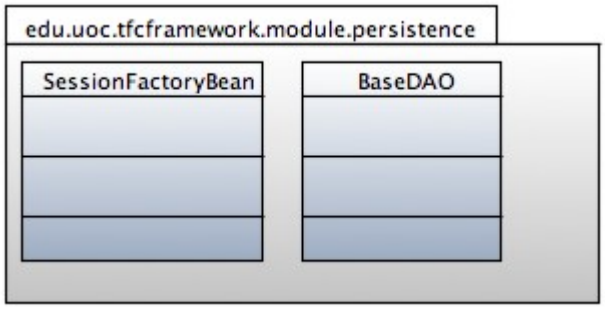

Figura 16: Diagrama de classes mòdul persistence

- SessionFactoryBean: estén *org.springframework.orm.hibernate4.LocalSessionFactoryBean* per tal d'afegir possibles fitxers de mapeig que empaquetats en altres mòduls.

  Mètodes:

  - *afetrPropertiesSet()*: Sobreescriu el mètode d'inicialització per tal de realitzar l'escaneig en el *classpath* de fitxers ubicats a META-INF/tfc/database/*.hbm.xml per tal d'afegir-los al llistat de mappingLocations.

- BaseDAO: implementa les funciona més habituals d'accés a base de dades. La API és molt semblant a la de JPA.

  Mètodes:

  - *clear()* : Neteja la memòria cau de primer nivell. Totes les instàncies queden desvinculades de la sessió hibernate.
  - *contains(*Object *entity) :* Boolean: Comprova si l'entitat està vinculada a la sessió.
  - *query(*String, Map *params) : List*: Executa la consulta especificada en hql amb els paràmetres epecificats.
  - *detach(*Object *entity)*: Desvincula l'instancia de al sessió eliminant-la de la memória cau de primer nivell.
  - *find(*Class *entityClass,* Object *primaryKey) :* Object: Recupera per clau primària.
  - *find(*Class *entityClass,* Object *primaryKey,* LockModeType *lockMode) :* Object: Recupera per clau primària i indica un bloqueig de tipus "CAP", "LECTURA", "ESCRIPTURA".
  - *flush()*: Bolca els canvis a la base de dades.
  - *getSessionFactory() :* org.hibernate.SessionFactory: Obté la sessió d'hibernate vinculada al DAO.
  - *lock(*Object *entity,* LockModeType *lockMode)*: Especifica un bloqueig a una entitat.
  - *merge(*Object *entity)*: Modifica la instància que hi ha en el context amb els valors de l'entitat passada.
  - *persist(*Object *entity)*: Torna la instància persistent.

- *refresh(*Object *entity)*: Actualitza la instància actual amb els canvis de base de dades.
- *refresh(*Object *entity, LockModeType lockMode)*: Actualitza la instància actual amb els canvis de base de dades. Indicant un bloqueig.
- *remove(*Object *entity)*: Elimina la entitat.

3.6. Mòdul security

El mòdul proporciona les dependències necessàries per assegurar una aplicació. Mitjançant un conjunt de taules es poden definir els usuaris i rols, es proporcionen els mapejos per a aquestes taules.
Mòdul basat en spring-security, que s'integra amb la cap de presentació mitjançant una llibreria de *tags* que permet mostrar porcions de la vista en funció dels rols d'usuari i amb la capa de negoci mitjançant unes anotacions que permeten autoritzar la invocació de mètodes de negoci (utilitzant la Domain Object Security (ACLs) d'spring-security)

a) Diagrama de Classes

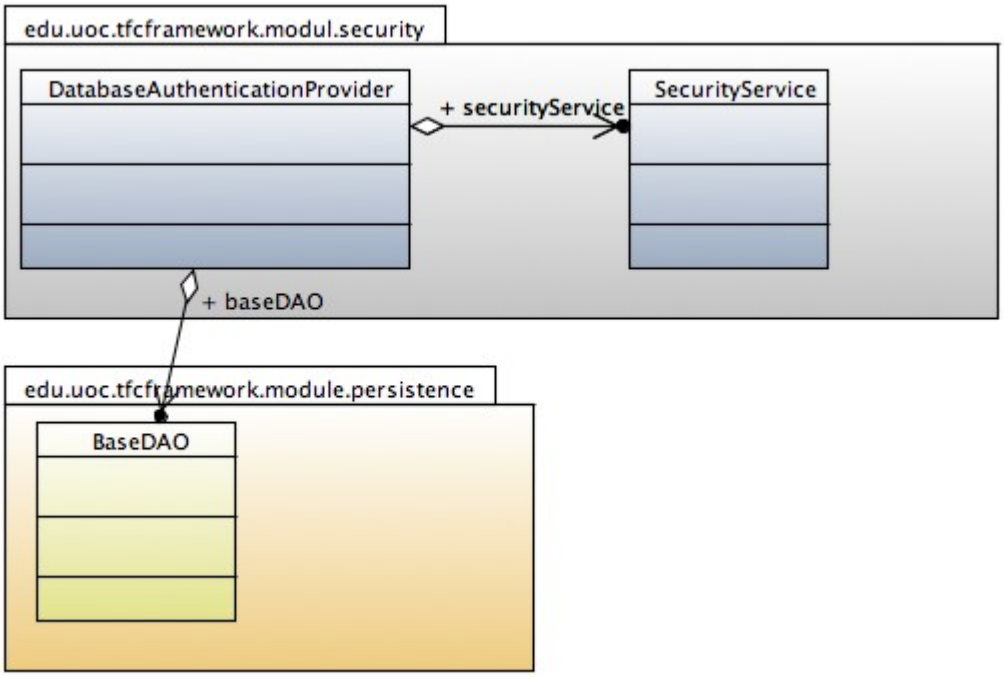

Figura 17: Diagrama de classes mòdul security

- DatabaseAuthenticationProvider: implementa *org.springframework.security.authenticationAuthenticationProvider* per tal de permetre l'autenticació contra les taules de seguretat. Es configura mitjançant el *ProviderManager* d'spring-security, per tal que intercepti les peticions d'autenticació.

  Propietats:

  - *securityService*(SecurityService, obligatori): l'utilitza per accedir al contexte de seguretat.
  - *baseDAO(*BaseDAO*, obligatori)*: Utilitza per recuperar la informació de seguretat autenticar l'usuari i carregar-ne els rols.

Mètodes:

- *authenticate(*Authentication *authentication) : Authentication*: Rep una la informació d'autenticació i la valida contra base de dades, en cas que les credencial proporcionades siguin correctes, recupera els rols d'usuari i actualitza el context de seguretat utilitzant SecurityService.

- SecurityService: Encapsula els accessos més comuns al context de seguretat.

  Mètodes:

  - *updateSecurityContext(*Authentication *authentication)*: registra una nova autenticació en el context de seguretat.
  - *getCurrentUser(): *User: obté l'usuari actual.
  - *IsUserInRole(*String *rol): *Boolean: determina si l'usuari actual té assignat algun rol.

Aquest mòdul contarà també d'una sèrie de taules i les seves corresponents classes de mapeig.

b) Diagrama de ER

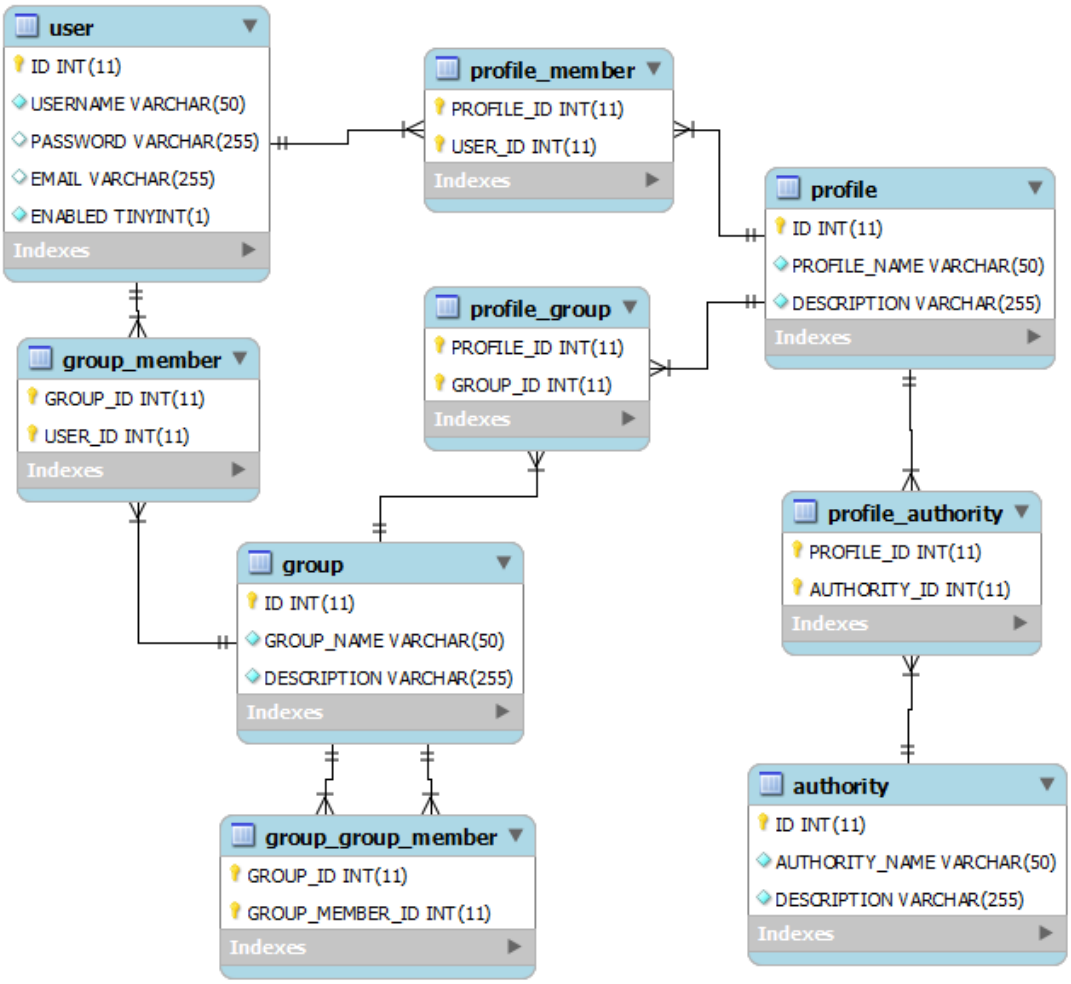

Figura 18: Diagrama ER mòdul security

USER: taula que conté els usuaris que tindran accés a l'aplicació.

Camps:

- **id** (PK): identificador intern de l'usuari.
- **username**: nombre d'usuari.
- **password**: clau de l'usuari.
- **email**: Correu electrònic de l'usuari.
- **enabled**: Indica si l'usuari està actiu o no.

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	INT(11)	☑	☑	☐	☐	☐	☐	☑	
USERNAME	VARCHAR(50)	☐	☑	☑	☐	☐	☐	☐	
PASSWORD	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	NULL
EMAIL	VARCHAR(255)	☐	☐	☐	☐	☐	☐	☐	NULL
ENABLED	TINYINT(1)	☐	☑	☐	☐	☐	☐	☐	

GROUP: taula que conté els grups que tindran accés a l'aplicació.
Camps:

- **id** (PK): identificador interno del grup.
- **group_name**: nom del grup.
- description: descripció del grup.

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
ID	INT(11)	☑	☑	☐	☐	☐	☐	☑	
GROUP_NAME	VARCHAR(50)	☐	☑	☐	☐	☐	☐	☐	
DESCRIPTION	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	

PROFILE: **taula** que conté els perfiles als que pot pertànyer cada usuari de la aplicació. Cada perfil agruparà diversos permisos d'accés (*authorities*) als recursos de l'aplicació.
Camps:

- **id** (PK): identificador intern del perfil.
- **profile_name**: nom del perfil.
- **description**: descripció del perfil.

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(4)	☑	☑	☐	☐	☑	☐	☑	
profile_name	VARCHAR(50)	☐	☑	☑	☐	☐	☐	☐	
description	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	

AUTHORITY: tabla que conté els permisos d'accessos als recursos de l'aplicació.
Camps:
id (PK): identificador interno del permís.
authority_name: nom del permís.
description: descripció del permís.

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(4)	☑	☑	☐	☐	☑	☐	☑	
authority_name	VARCHAR(50)	☐	☑	☑	☐	☐	☐	☐	
description	VARCHAR(255)	☐	☑	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	

3.7. Mòdul reporting

El mòdul proporciona les dependències necessàries i una API per a generar generar informes.

També s'ocupa de la integració amb el servei Web per tal de publicar llistats i publicar informes propis.

a) Diagrama de Classes

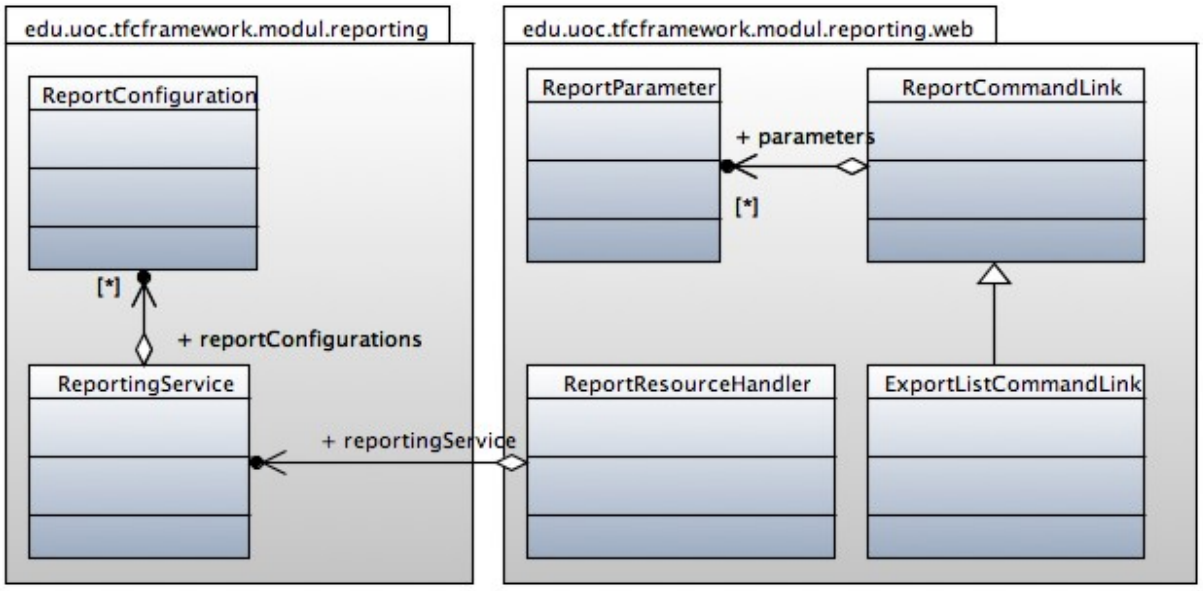

Figura 19: Diagrama de classes mòdul reporting

- ReportConfiguration: Encapsula la informació per a generar un informe.

  Propietats:

  - *templatePath*(String, obligatori): ruta a la plantilla en format JRXML.
  - *reportParameters*(Mapa, opcional): paràmetres que es passaran a la plantilla a l'hora de generar.
  - *configurationParameters*(Mapa, opcional): paràmetres per configurar el motor de plantilles (Son paràmetres del JasperReports).

- ReportingService: Proporciona mètodes per a poder generar els reports que estan configurats.

  Propietats:

  - *reportConfigurations*(Mapa de String i ReportConfiguration, obligatori): emmagatzema les configuracions d'informe i els seus identificadors.

  Mètodes:

  - *initialitze()*: cridat quant s'aixeca la instància (anotat com a @PostConstruct), carrega les configuracions que hi han en el contenidor d'objectes de l'aplicació.
  - *generateReport(*String *reportId,* String *format,* OutputStream *out,* Map *reportParameters,* Map *configurationParameters)* : Genera l'informe tenint en compte el format especificat, injectant els reportParameters (barrejant-los amb els que ja hi han configurats) al motor de plantilles i configurant aquest motor (JasperReports) amb els paràmetres de configurationParameters (barrejant-los amb els que ja hi han configurats).

- ReportResourceHandler: Hereta de *javax.faces.application.ResourceHandler* i registrat com

a llibreria "*reporting*" i s'ocupa de publicar els reports.

Propietats:

- *reportingComponent*(ReportingComponent, obligatori): utilitzat per generar els reports.

Mètodes:

- *createResource(*String *resourceName*, String *libraryName)* : *javax.faces.application.Resource*: Cridat per JSF genera el report amb identificador "resourceName" i amb els paràmetres de la *request*.

- ReportCommandLink: Hereta de *javax.faces.component.UIComponentBase* i genera un enllaç a un report.

  Propietats:

  - *reportId(*String*, obligatori) :* Identificador de l'informe.
  - *paràmetres(*ReportParameter, obligatori*)* : Llistat amb els paràmetres.

  Mètodes:

  - *encodeBegin(*javax.faces.context.FacesContext *context)* : Genera l'enllaç capaç d'enviar els payàmetres configurats.

- ReportParameter: Hereta de *javax.faces.component.UIComponentBase* no renderitza res però guarda el no i valor d'un paràmetre.

  Propietats:

  - *name(*String, obligatori*)* : nom del paràmetre.
  - *value(*Object*, obligatori) : valor del paràmetre.*
  -

- ExportListCommandLink: Hereta de ReportCommandLink i està vinculat a un *javax.faces.component.HtmlDataTable* de JSF, d'on n'extreu les dades i la configuració per al report.

  Propietats:

  - *dataTable(*javax.faces.component.HtmlDataTable, obligatori*)* : taula a la que està vinculat.

  Mètodes:

  - *encodeBegin(*javax.faces.context.FacesContext *context)* : Genera l'enllaç per tal que es generi el report.

3.8. Mòdul webservices

Es tracta d'un mòdul de llibreries basat en CXF que permet publicar i consumir serveis web, proporciona també integració amb el mòdul de seguretat permetent autenticar mitjançant UserNameToken contra el component de seguretat i que això actualitzi el context de seguretat per a aquest petició. ÉS adir que a la capa de negoci apliquin les restriccions per rol.

a) Diagrama de Classes

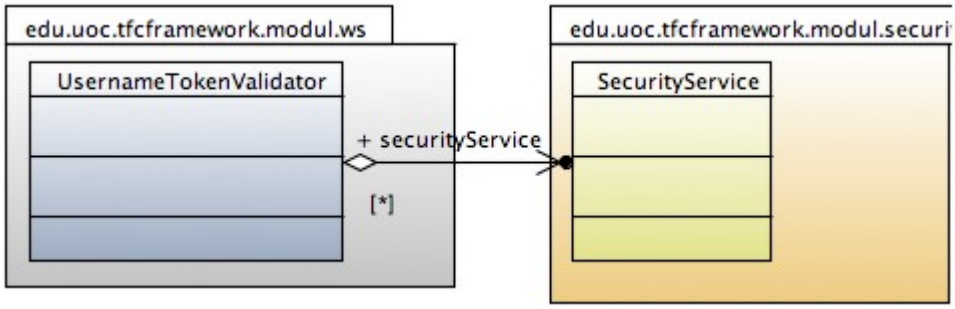

Figura 20: Diagrama de classes mòdul Web Services

- UsernameTokenValidator: Hereta de *org.apache.ws.security.validate.Validator* per tal de validar les credencials utilitzant el component de seguretat.

  Mètodes:

  - *validate(*Credential *credential,* RequestData *data) : Credential* : valida les credencials amb el servei de seguretat. En cas d'error llença una WSSecurityException, es cas que l'autenticació sigui correcte actualitza el context de seguretat de SecurityService

4. Aplicació demo

Per tal de demostrar el bon funcionament del framework s'ha creat una aplicació *demo* creada a partir de l'estructura i bones pràctiques recollides en aquesta memòria, es tracta d'un manteniment de clients, a més incorpora un informe on s'agrupa els clients per número de contracte. A continuació s'explica amb detall com està construïda aquesta aplicació.

4.1. Mòdul common:

Ja s'ha explicat les funcions del mòdul comú. És on es troben les interfícies de negoci utilitzades per les altres capes i els objectes POJO que es passaran entre capes.

a) Estructura:

En aquest cas , el mòdul implementa la interfície CustomerBO i els POJOS Contract i Customer.

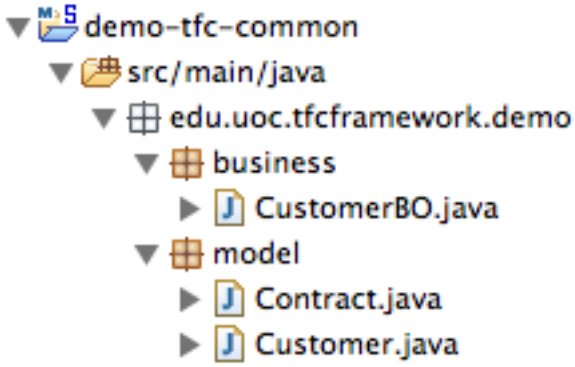

Figura 21: Demo - estructura mòdul common

4.2. Mòdul business:

El mòdul de negoci implementa tots els mètodes de negoci de l'aplicació i conté la major part de configuració i testos. Té com a única dependència el mòdul *common*.

a) Estructura:

Aquest mòdul implementa tota la lògica de negoci, l'accés a dades, construcció d'informes i conté bona part de la configuració (font de dades, seguretat, missatges de negoci, traces ...) també conté la implementació dels testos unitaris de la capa de negoci. La següent captura il·lustra la seva estructura:

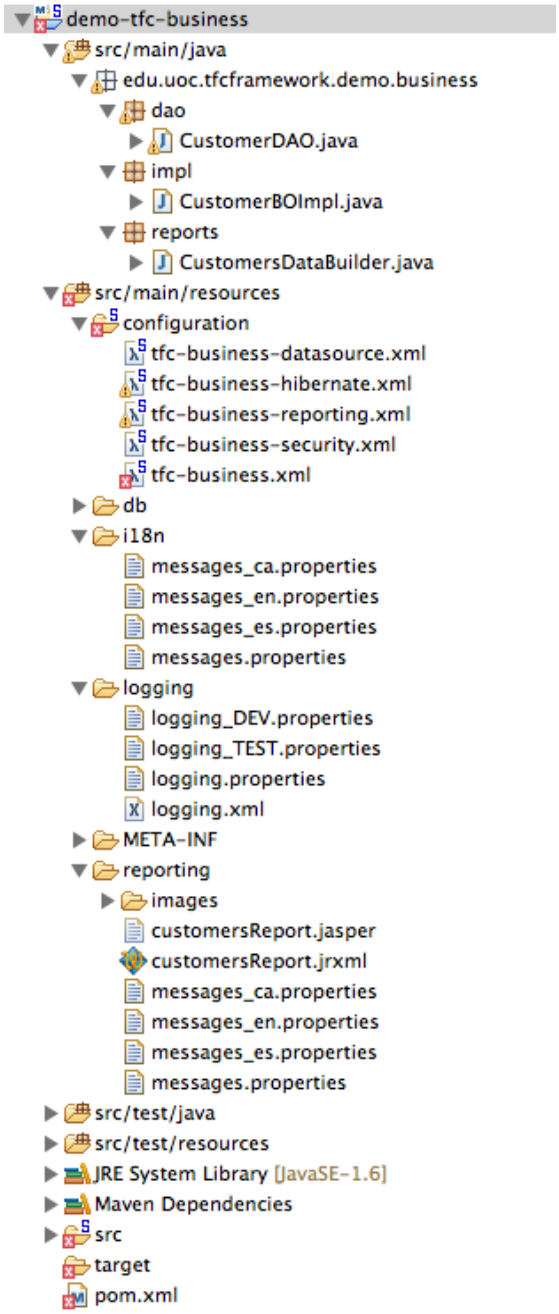

Figura 22: Demo - estructura mòdul business

b) Implementació:

Data access object

Implementació dels objectes d'accés a dades està basada en JPA, utilitza el *javax.persistence.EntityManager*, que és injectat automàticament amb l'anotació *javax.persistence.PersistenceContext*. Aquí tenim un exemple del codi d'un *DAO*.

```
@Dao
public class CustomerDAO {
	private @PersistenceContext EntityManager entityManager;
	private @Log Logger log;
	/*
	 * (non-Javadoc)
	 * @see
edu.uoc.tfcframework.demo.business.CustomerBO#load(java.lang.Integer)
	 */
	public Customer load(Integer id) {
		return entityManager.find(Customer.class, id);
	}
...
```

Business Object

Els objectes de negoci són transaccionals, comencen i acaben una transacció per si mateixos, tot i que es pot imbricar crides de negoci dins altres crides de negoci més generals. A continuació un exemple de codi de negoci on veiem com es fa ús d'un objecte d'accés a dades.

```
@BusinessObject
public class CustomerBOImpl implements CustomerBO {
	private @Inject CustomerDAO customerDAO;
	private @Log Logger log;
	/*
	 * (non-Javadoc)
	 * @see
edu.uoc.tfcframework.demo.business.CustomerBO#load(java.lang.Integer)
	 */
	@Transactional(readOnly=true)
	public Customer load(Integer id) {
		log.debug("load customer");
		return customerDAO.load(id);
	}
```

Report Data Builder

Els informes que es dissenyen a mida necessiten una classe que s'encarregui de proporcionar la font de dades, per això cal que implementi el mètode *getReportData* tal i com es veure en aquest codi.

```
@Component
public class CustomersDataBuilder implements ReportDataBuilder {
```

```
	private @Inject CustomerBO customerBO;
	/*
	 * (non-Javadoc)
	 * @see
edu.uoc.tfcframework.module.reporting.model.ReportDataBuilder#getReportD
ata(java.util.Map)
	 */
	public ReportData getReportData(Map<String, Object> params) {
		return new ReportData(customerBO.findAll(), params);
	}
}
```

c) Configuració:

El mòdul consta d'una sèrie de fitxers de configuració en format spring, que son carregats automàticament, cal que estiguin a al carpeta configuration i el seu nom comenci per tfc-. Com a exemple veim com es configura la càrrega dels components del mòdul, les traces i els missatges de negoci.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:context="http://www.springframework.org/schema/context"
	xmlns:tfc-core="http://www.uoc.edu/schema/tfcframework/core"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd
		http://www.uoc.edu/schema/tfcframework/core
http://www.uoc.edu/schema/tfcframework/core/namespace.xsd
	">

	<!-- Scan for business components -->
    <context:component-scan base-
package="edu.uoc.tfcframework.demo.business" />

	<!-- Logging configuration -->
	<tfc-core:configureLogging
configuration="classpath:logging/logging.xml" />
	<tfc-core:configurationFile
path="classpath:logging/logging.properties" />

	<!-- I18N configuration -->
	<tfc-core:loadResourceBundle baseName="i18n.messages" />
</beans>
```

4.3. Mòdul web:

El mòdul web implementa les capes de vista i controlador, apart també incorpora la seguretat a nivell de URL i la configuració del menú.

a) Estructura:

L'estructura és similar a la resta de mòduls, amb una carpeta pel codi font "src/main/java" i una carpeta per la configuració "src/main/resources", a més també incorpora una carpeta amb els recursos disponibles dins la web "src/main/webapp".

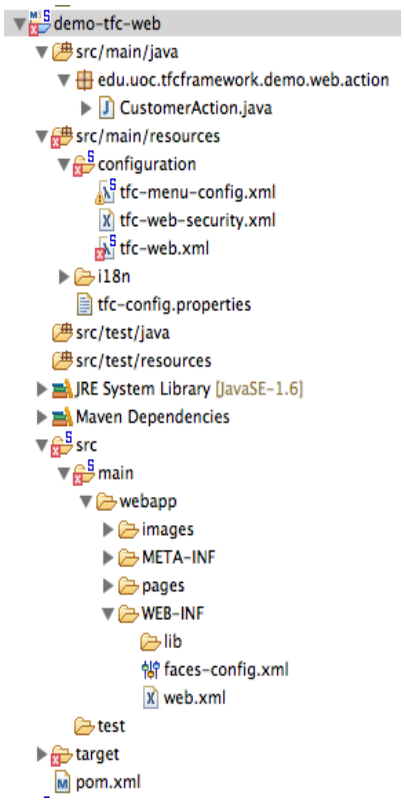

Figura 23: Demo - estructura mòdul web

b) Implementació:

Per tal d'implementar la capa de control, es programa una classe de tipus Action que servint-se de la capa de negoci, proporciona tota mena de funcions i objectes necessaris a la vista. Per exemple:

```
@Action
@Scope(WebApplicationContext.SCOPE_SESSION)
public class CustomerAction {
	private @Inject CustomerBO customerBO;
	private Customer filter;
	private DataTableHelper<Customer> customers = new
DataTableHelper<Customer>();
	@PostConstruct
	public void init() {
```

```
            filter = new Customer();
            customers.setData(new GenericDataModel<Customer>() {
                  @Override
                  public List<Customer> find(int firstRow, int
numberOfRows,
                              Order... orders) {
                        return customerBO
                                    .find(filter, firstRow, numberOfRows,
orders);
                  }
                  @Override
                  public int rowCount() {
                        return customerBO.count(filter);
                  }
                  @Override @EntityId("id")
                  public Customer getObjectById(Object id) {
                        return customerBO.load((Integer)id);
                  }
            });
      }

      public String find() {
            return "/pages/customerManagement";
      }
```

c) Configuració:

En el mòdul web es configuren bàsicament 2 aspectes de l'aplicació:

Seguretat:

Es defineix el formulari d'entrada, la URL de sortida i els rols per a cada patró de URLs:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:security="http://www.springframework.org/schema/security"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
    http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd
 ">

      <security:http
            realm="TFC Framework Application"
            access-decision-manager-ref="accessDecisionManager">
```

```
          <!-- local DB login & logout -->
          <security:intercept-url
pattern="/tfc:/module/security/login*"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:form-login
               login-processing-
url="#{ securityService.loginProcessingUrl }"
               login-page="/tfc:/module/security/login.jsf"
               authentication-failure-
url="/tfc:/module/security/login.jsf"
               always-use-default-target="true"
               default-target-url="/pages/welcome.jsf" />
          <security:logout
               logout-url="/logout"
               logout-success-url="/tfc:/module/security/login.jsf"
               invalidate-session="true" />
          <!-- end local DB login & logout -->

          <!-- non secure URLs -->
          <security:intercept-url pattern="/styles/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url pattern="/images/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url pattern="/services/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url pattern="/styles/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url pattern="/css/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url pattern="/javax.faces.resource/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />
          <security:intercept-url
pattern="/org.richfaces.resources/javax.faces.resource/**"
access="IS_AUTHENTICATED_ANONYMOUSLY" />

          <!-- role-based secure URLs -->
          <security:intercept-url
pattern="/tfc:/module/security/management/**" access="web-admin" />
          <security:intercept-url pattern="/**" access="web-basic" />

     </security:http>

</beans>
```

Menu:

La configuració del menú s'ocupa de definir les opcions que hi hauran disponibles en el menú de

l'esquerra de cada aplicació. Així com els rols que tindran accés a cada una d'elles. També pot incloure referències a menús empaquetats en altres mòduls, com ara la referència al menú del mòdul de seguretat.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
     Menu configuration
-->
<menu
     xmlns="http://www.uoc.edu/schema/tfcframework/module/web"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.uoc.edu/schema/tfcframework/module/web
http://www.uoc.edu/schema/tfcframework/module/web/namespace.xsd"
     id="mainMenu" label="menu.root">
     <menuItem
          action="/pages/welcome"
          label="menu.welcome"
          authorities="web-basic,web-admin"/>
     <menuItem
          action="/pages/customerManagement"
          label="menu.customerManagement"
          authorities="web-basic,web-admin"/>

     <menuItem
          label="menu.admin"
          authorities="web-admin">
          <menuItem ref="tfc-module-security-management-menu" />
     </menuItem>
</menu>
```

Aquest configuració acaba generant el menú que veiem a continuació:

Figura 24: Demo - menú mòdul web

4.4. Mòdul serveis web:

El mòdul de serveis web té la implementació i configuració per tal de publicar serveis que puguin ser consumits per altres aplicacions, té com a dependència el mòdul common, d'aquest manera pot fer ús de les interfícies de negoci.

a) Estructura:

El mòdul conté tant la implementació, com al configuració i el contracte (wsdl) dels serveis.

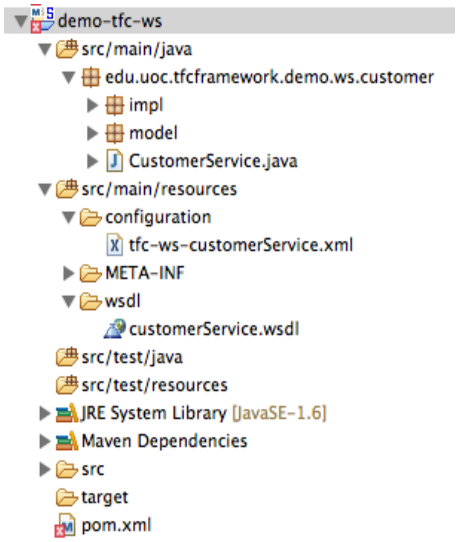

Figura 25: Demo - estructura mòdul serveis web

b) Implementació

La implementació dels serveis està basada en JAX-WS, també es permet (igual que a la resta de classes del framework) implementar una seguretat declarativa amb les anotacions d'*spring security*.

```
@WebService(

endpointInterface="edu.uoc.tfcframework.demo.ws.customer.CustomerService
",
	targetNamespace=CustomerService.TARGET_NAMESPACE,
	serviceName="CustomerServiceService",
	portName="CustomerServiceServicePort",
	name="CustomerServiceServicePortType"
)
public class CustomerServiceImpl implements CustomerService {
	private @Inject CustomerBO customerBO;
	/*
	 * (non-Javadoc)
	 * @see
edu.uoc.tfcframework.demo.ws.customer.CustomerService#loadCustomer(java.
lang.Integer)
	 */
	@PreAuthorize("hasRole('web-basic')")
	public Customer loadCustomer(Integer pojoId) throws
ServiceException {
		return adapt(customerBO.load(pojoId));
	}
```

c) Configuració

La configuració està basada en CXF i permet particularitzar els detalls de publicació del servei.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
	Application context definition for WebService services.
-->
<beans xmlns="http://www.springframework.org/schema/beans"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xmlns:jaxws="http://cxf.apache.org/jaxws"
	xsi:schemaLocation="
		http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
		http://cxf.apache.org/jaxws
http://cxf.apache.org/schemas/jaxws.xsd
	">

	<!-- Servei customer -->
	<!-- definició del bean que implementa el customerService -->
```

```
	<bean id="customerServiceImpl"


class="edu.uoc.tfcframework.demo.ws.customer.impl.CustomerServiceImpl" /
>

	<!--
		definició de l'endpoint que fa referència al bean
		#customerServiceImpl i es publica a la url /customerService
	-->
	<jaxws:endpoint
		implementor="#customerServiceImpl"
		address="/customerService"
		wsdlLocation="classpath:wsdl/customerService.wsdl"
	>
		<jaxws:properties>
			<entry key="schema-validation-enabled" value="true" />
			<entry key="ws-security.ut.validator">
				<bean
class="edu.uoc.tfcframework.module.ws.authentication.UsernameTokenValida
tor" />
			</entry>
		</jaxws:properties>
	</jaxws:endpoint>
</beans>
```

4.5. Cas d'us gestió de clients:

L'aplicació demo implementa el cas d'ús de gestió de clients, per tal d'accedir-hi cal en primer lloc identificar-se:

Figura 26: Demo - Cas d'ús clients - login

Veiem que hi ha un punt de menú "Gestió de clients" que ens dona accés al menteniment.

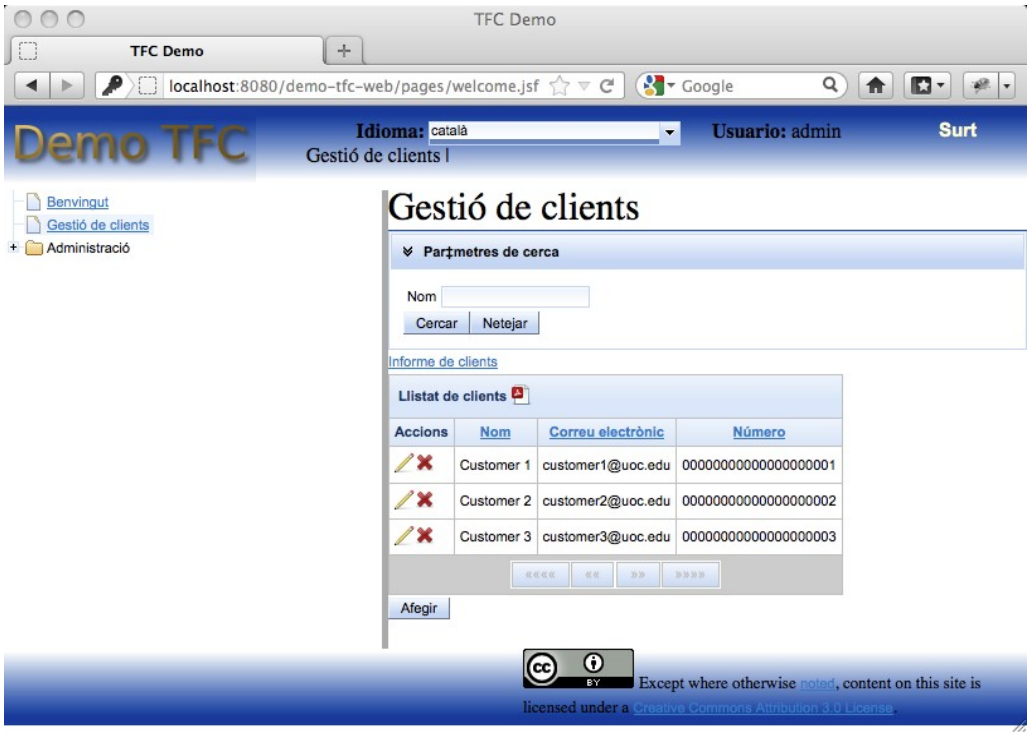

Figura 27: Demo - Cas d'ús clients - llistat

La pantalla ens ofereix un llistat (que podem ordenar, filtrar o exportar a *pdf*), també hi ha un enllaç a l'informe de clients (un altre *pdf* que agrupa els clients per contracte) i una sèrie d'accions que ens permeten modificar, afegir o esborrar clients. La pantalla per a crear o modificar clients és un diàleg modal, tal i com es veu a la següent captura.

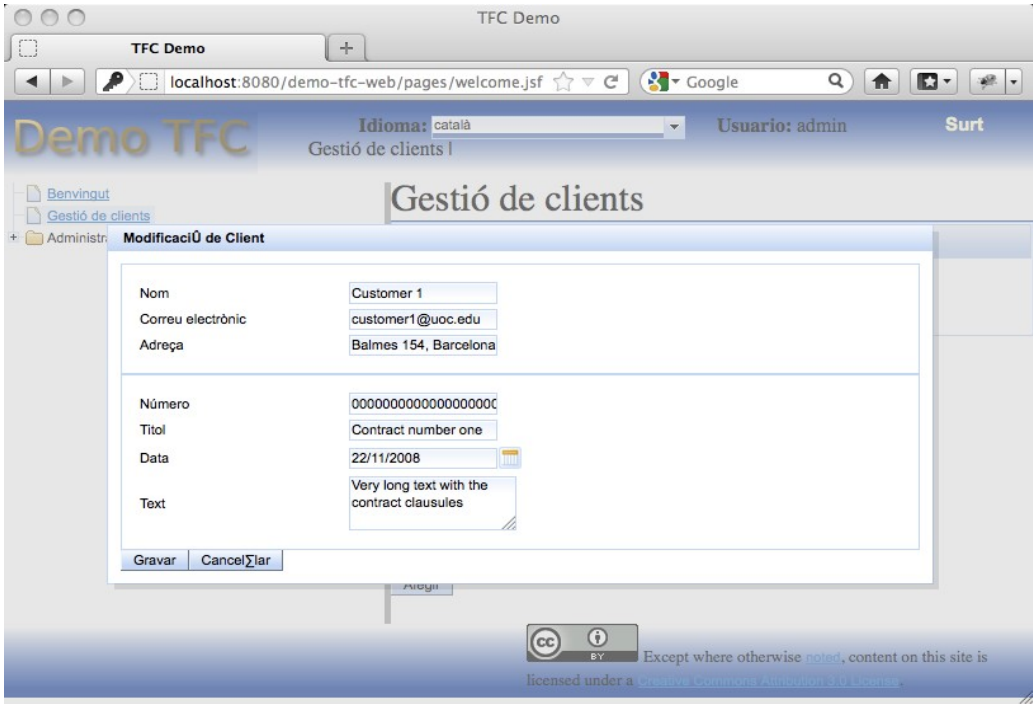

Figura 28: Demo - Cas d'ús clients - edició

A l'hora d'esborrar un client, l'aplicació mostra un diàleg de confirmació:

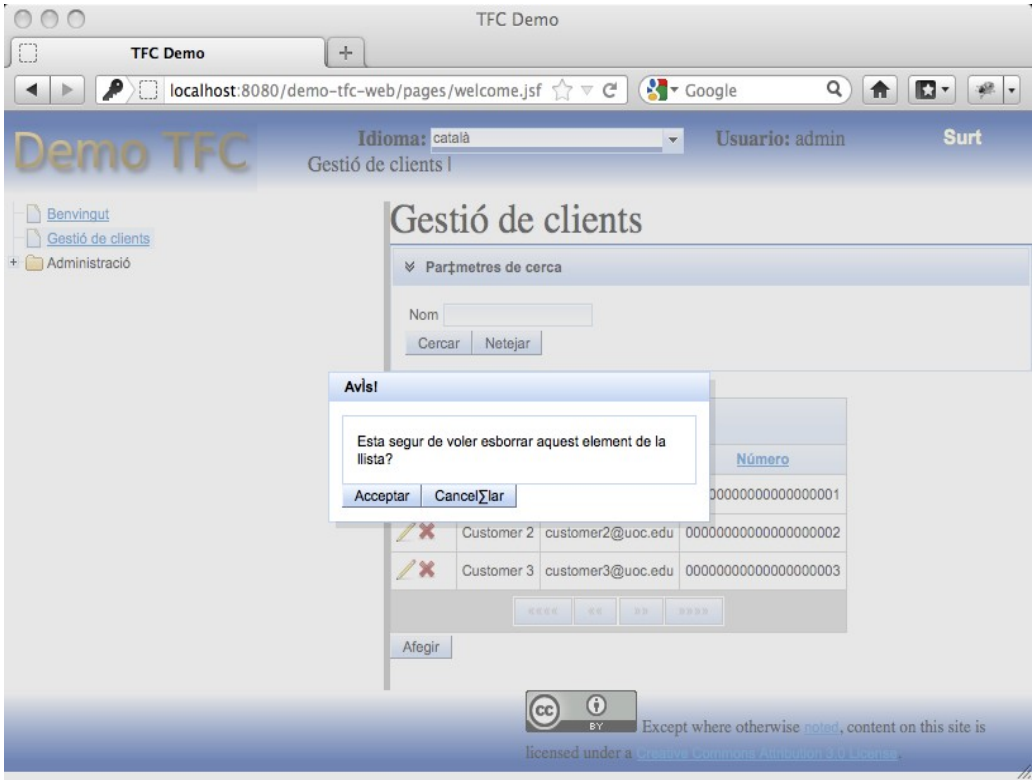

Figura 29: Demo - Cas d'ús clients - esborra

5. Conclusions

En la justificació i contexts del framework veiem els riscos que podia córrer una organització a l'hora de desenvolupar programari a mida, si cada equip de desenvolupament té llibertat per decidir lliurement les tecnologies a utilitzar el més provable és que tot plegat acabi amb una dispersió tecnològica. És a dir, projectes tan diferents que es faci difícil integrar-los, reaprofitar codi, reaprofitar coneixement, mantenir-los, unificar-ne l'aspecte, cooperar entre equips...

El TFC ha definit solucions a una sèrie de problemes (seguretat, persistència, transaccionalitat, configuració, traces, multi-idioma, capa de presentació, publicació de serveis, generació d'informes, tests unitaris ...) tot plegat amb tecnologies obertes i acceptades per el mercat. A més proporciona un seguit d'eines com els arquetipus amb una doble funció accelerar l'arranc de la fase de codificació i uniformitzar de manera natural la estructura dels projectes.

Tot això proporciona una sèrie de avantatges a l'hora de poder plantejar-se canvis d'imatge corporatius, canvis de servidor d'aplicacions a nivell global, corregir errors d'arquitectura, proporcionar components visuals, proporcionar connectors a components nous que vagin apareixent ja que amb un conjunt d'aplicacions unificades només cal fer aquesta feina una vegada.

Tot això, però, només és un punt de partida, un conjunt de serveis bàsics, una manera d'empaquetar, de definir la configuració, de programar la capa de presentació... a partir d'aquí es poden anar introduint nous mòduls, nous elements a l'arquitectura, nous connectors per tal que les aplicacions desenvolupades amb el framework es puguin comunicar fàcilment amb la resta de sistemes implantats en l'organització.

6. Glossari

Context d'spring: Contenidor d'objectes que en gestiona el cicle de vida i la injecció de dependències mitjançant IoC.

IoC: paradigma on l'objecte que requereix un altre no s'ocupa de la seva creació, ho delega en el contenidor.

Gestió de dependències: el projecte només declara les llibreries que vol utilitzar i una eina (en el nostre cas *maven*) s'ocupa de resoldre les dependències transitives.

Servei web: part del negoci de l'aplicació que es publica per tal que pugui ser utilitzat per altres aplicacions. En el nostre cas es fa mitjançant JAXWS (l'estàndard *java*) i la implementació *CXF*.

Component visual: Peça de programari que pot ser incrustada a la vista i proporciona una funcionalitat concreta, en el nostre cas cal programar-lo com a component JSF.

Estereotip: Element que indica el rol d'un component de programari, el mecanisme és amb una anotació.

AOP: paradigma de programació que permet afegir una funcionalitat a un programa de manera transversal, sense haver de retocar el codi que ja hi ha fet.

Profile: perfils que s'activen en funció de l'entorn i permeten variar-ne la configuració amb l'objectiu que el mateix paquet es pugi comportar de manera diferent en funció de l'entorn on es troba. És una funcionalitat que proporciona *Spring*.

7. Bibliografia

Rod Johnson. Reference Documentation. [en línia].

http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/ [data de consulta: 17/06/2012].

Rod Johnson. Spring Extensible XML authoring. [en línia]. http://static.springsource.org/spring/docs/3.1.x/spring-framework-reference/html/extensible-xml.html [data de consulta: 17/06/2012].

Brian Leathem, Lukas Fryc i Sean Rogers. Developer Guide - Develop applications using RichFaces 4. [en línia]. http://docs.jboss.org/richfaces/latest_4_2_X/Developer_Guide/en-US/html/ [data de consulta: 17/06/2012].

Oracle. The Java EE 6 Tutorial - JavaServer Faces Technology. [en línia]. http://docs.oracle.com/javaee/6/tutorial/doc/bnaph.html [data de consulta: 17/06/2012].

Oracle. The Java EE 6 Tutorial - Composite Components: Advanced Topics and Example. [en línia]. http://docs.oracle.com/javaee/6/tutorial/doc/gkhxa.html [data de consulta: 17/06/2012].

L'equip d'Hibernate. Hibernate Developer Guide. [en línia]. http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html_single/ [data de consulta: 17/06/2012].

JasperReports. JasperReports Tutorial. [en línia]. http://jasperforge.org/uploads/publish/jasperreportswebsite/trunk/tutorial.html [data de consulta: 17/06/2012].

L'equip CXF. CXF User's Guide. [en línia]. http://cxf.apache.org/docs/index.html [data de consulta: 17/06/2012].

L'equip Maven. Introduction to the POM. [en línia]. http://maven.apache.org/guides/introduction/introduction-to-the-pom.html [data de consulta: 17/06/2012].

L'equip Maven. Maven Archetype Plugin. [en línia]. http://maven.apache.org/archetype/maven-archetype-plugin/ [data de consulta: 17/06/2012].