Llistats Web

Memòria del Projecte de Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Oriol Barcelona Palau
i dirigit per
Antonio M. López Peña
Bellaterra, 29 de gener de 2010

Índex

Capítol 1

Introducció

1.1 Motivació

En les empreses actuals és una pràctica molt comuna utilitzar diferents tecnologies com a motors de base de dades. El mercat és molt ampli i per tal que la producció tingui un bon rendiment cal utilitzar sempre el motor més adequat. El nostre client, té diverses bases de dades de diferents proveïdors instal·lades, que cadascuna dóna suport a diferents aplicacions: comptabilitat, gestió d'assegurances, control del personal, etc.

Una de les pràctiques habituals és obtenir llistats de les bases de dades. S'entén per llistat un conjunt d'informació ordenat per files. Cada fila representa un individu dins del llistat, amb la seva informació ordenada per columnes que són comunes per a tots els individus. Aquestes columnes normalment tenen unes capçaleres on s'especifica el que representa la informació.

Per tal de generar aquests llistats, s'utilitzen procediments emmagatzemats a la base de dades encarregats de fer les consultes adients per tal d'extreure la informació. Cada procediment és propi de la base de dades i té el seu llenguatge per programar-lo. Per exemple: Oracle i Postgres utilitzen l'anomenat PL/SQL, MySQL té el seu propi MySQL stored Procedure Language, entre d'altres exemples.

Els usuaris no saben comunicar-se i tractar amb bases de dades i per tant el Departament de Sistemes és l'encarregat de generar els informes. L'usuari s'hi ha de posar en contacte i fer la petició del llistat. Els treballadors del Departament de Sistemes tenen unes hores durant la setmana dedicades a la generació d'informes i al manteniment dels procediments. També durant aquest temps han de donar suport als usuaris que fan les peticions. Una vegada han generat el llistat, s'han de posar en contacte amb l'usuari i enviar-li el document corresponent. A més, si l'usuari demana llistats de bases de dades diferents, la complexitat a l'hora de generar els llistats és més gran ja que l'encarregat de la generació ha de saber com tractar cada tipus de bases de dades. Una vegada l'usuari ha rebut els llistats, moltes vegades cal generar una documentació associada al llistat. Actualment aquest procés es fa

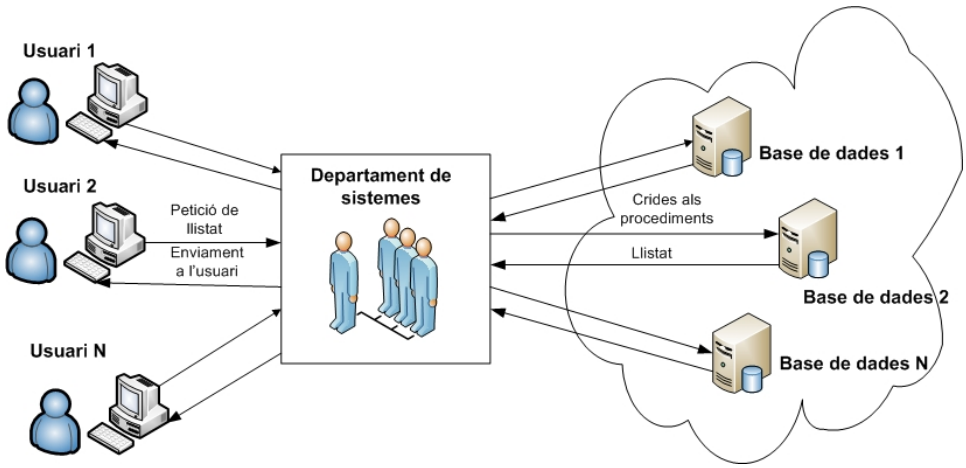

Figura 1.1: Mètode de treball abans de l'entrada en funcionament de l'aplicació

manualment mitjançant plantilles Word i Excel. El mètode de treball es pot veure a la figura 1.1.

Els problemes que han sorgit d'aquest sistema són els següents:

- El sistema de generació de llistats és lent i requereix que diverses persones estiguin implicades en la generació.
- Les demandes per part dels usuaris han augmentat i el Departament de Sistemes es troba que no té prou hores assignades per tal de cubrir la creació de llistats.
- Els treballadors del Departament de Sistemes han d'estar formats en diverses bases de dades per tal de poder-hi treballar.
- El format de sortida dels llistats no és ni atractiu ni agradable.

Vistos tots aquests problemes, s'arriba a la conclusió que, en un entorn tan disgregat, cal una manera de poder generar els llistats independentment de les tecnologies que hi hagi al darrere. La idea és que la solució sigui una interfície que comuniqui l'usuari amb els processos emmagatzemats a les diferents bases de dades. A més s'ha d'aconseguir que el procés no involucri al Departament de Sistemes, sino que el propi usuari sigui capaç de fer-ho. Per tant, la solució ha de ser intuitiva i amigable ja que l'usuari no ha de tenir coneixements sobre les tecnologies.

Cal fer la distinció entre usuari i client. Un usuari és qualsevol persona que utilitza l'aplicació en producció. El client és l'empresa que ha contractat la creació de l'aplicació i que, per tant, ha de donar el vistiplau als requeriments de l'aplicació.

A l'aplicació l'anomenem 'Llistats Web' (o LWB). La paraula 'Llistats' prové de la principal funcionalitat que ha de tenir, la generació de llistats. La paraula 'Web' prové de com s'accedeix a l'aplicació: via el navegador d'Internet. Per tant

s'ha intentat buscar un nom força representatiu de les característiques de l'aplicació.

1.2 Objectius

Analitzada la problemàtica de la situació actual l'objectiu global del projecte és automatitzar tot el procés de generació de llistats. Per assolir aquest objectiu, cal marcar-se una sèrie de tasques per tal de poder obtenir una solució. Les tasques que es durant a terme durant el projecte seran:

- Estudiar com s'emmagatzemen, a les bases de dades del client, els procediments encarregats de la generació d'informes i trobar una manera unitària de cridar-los.
- Implementar una solució que permeti la creació de llistats contra bases de dades de diferents proveïdors.
- Treballar en l'obtenció dels llistats per tal que es puguin exportar en els formats: TXT, Excel97 i OpenXML. En qualsevol cas, el programari sempre ha de permetre ampliar els formats d'exportació.
- Permetre obtenir la documentació associada a un llistat mitjançant la comunicació amb el servei de generació documental.
- Dotar de seguretat l'aplicació, tant pel que fa a l'autenticació com en el control d'accés a determinats recursos.

1.3 Estructura de la memòria

En el capítol 2 es planteja com s'ha dissenyat l'aplicació, quines tecnologies s'han utilitzat i el seu perquè. L'estructuració del projecte en diferents subprojectes també forma part d'aquest capítol. Finalment es comenta la metodologia que s'ha seguit durant el desenvolupament de l'aplicació i l'entorn de desenvolupament del qual s'ha disposat per dur a terme el projecte.

En el capítol 3 es comenten els detalls de la implementació i el desplegament del l'aplicació. La implementació inclourà: com s'ha dissenyat i integrat el model de dades amb l'aplicació, el disseny i l'explicació d'algunes de les pantalles de l'aplicació client i el funcionament complet del servidor. El funcionament del servidor centra la majoria d'aquest capítol ja que és el motor principal de l'aplicació. En aquest capítol també hi ha una breu introducció als proveïdors de servidors que s'han utilitzat i la seva raó. A més, s'hi troben detallats els diferents entorns en què es trobarà l'aplicació.

En el capítol 4 es tracten els resultats de la posada en marxa de l'aplicació. Des del moment en què el client valida l'aplicació fins a l'impacte que ha tingut el pro-

jecte en termes econòmics i temporals. A més, s'hi poden trobar les estadístiques del funcionament del programari.

Finalment, al capítol 5 es troben les conclusions extretes després de la finalització del projecte i les possibles vies de continuació.

Capítol 2

Disseny de la solució

Durant l'etapa de disseny s'han hagut de pendre les decisions següents:

- Quina arquitectura del software ha de tenir la solució, és a dir, quines són les responsabilitats de cada mòdul del nostre programa i com s'han de comunicar entre ells.
- Quins patrons i models han de seguir els moduls a desenvolupar.
- Com dotar de seguretat l'aplicació.
- Com s'organitzarà en diferents projectes per tal de seguir adeqüadament tant l'arquitectura com els patrons i models.

A continuació s'analitza la presa de decisions i es fa una breu introducció a les tecnologies utilitzades.

2.1 Arquitectura del software

La solució adoptada ha estat utilitzar l'arquitectura del software centrada en un client i un servidor. La principal funcionalitat del client és mostrar a l'usuari la interfície gràfica, realitzar peticions i veure'n l'estat. La funcionalitat del servidor és comprovar si hi ha noves peticions i executar-les, si pertoca. D'aquesta manera s'aconsegueix separar dues funcionalitats molt clares. Es podria haver-ho implementat tot en un sol servidor però el resultat hauria estat una aplicació molt pesada, poc eficient i difícil de mantenir.

Malgrat aquest raonament, el servidor també té una petita interfície gràfica destinada a l'administració, només disponible per als usuaris pertinents.

Cal remarcar que la solució adoptada està inspirada en un paradigma client-servidor. Malgrat això, el client no es comunica en cap dels casos amb el servidor per tal de fer-li peticions. En ambdós casos, la via de comunicació és mitjançant la base de dades. Això afavoreix el rendiment de l'aplicació i la independència dels dos components. Encara que el servidor estigui caigut es poden posar en cua

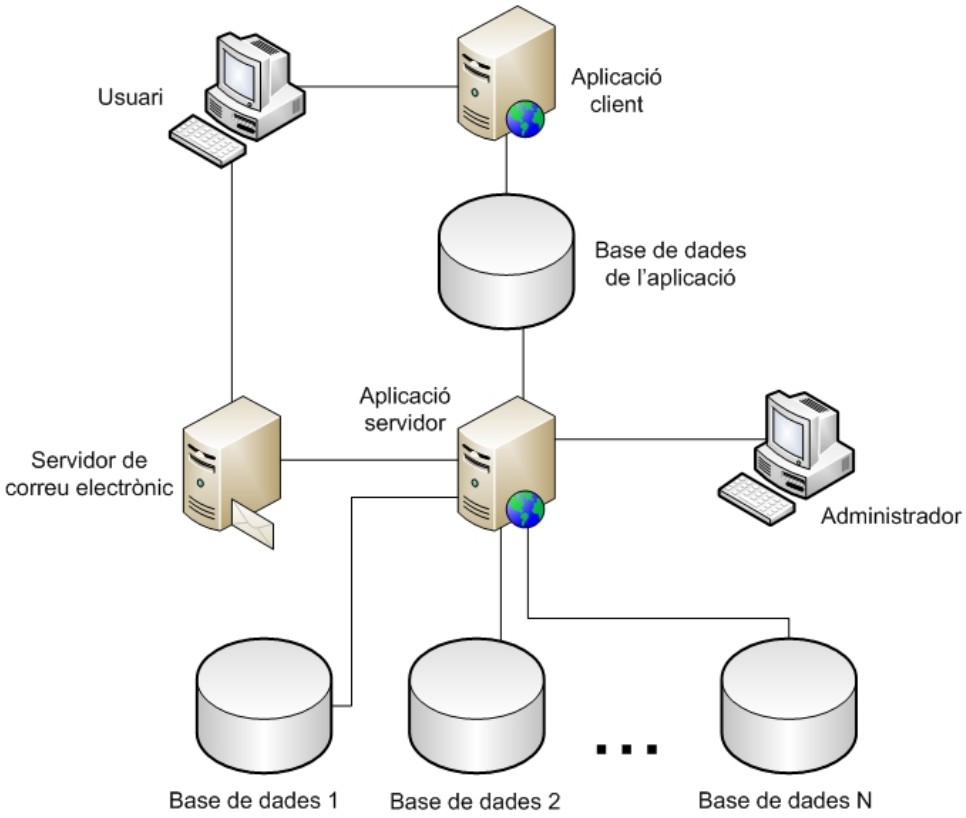

Figura 2.1: Interacció entre components de LWB

peticions, i a l'inrevés, quan el client no funciona es poden atendre les peticions pendents i fins i tot generar-ne de noves introduint nous registres a la base de dades.

Aquesta arquitectura del software està representada en la figura 2.1.

2.2 Aplicació Web

Es va decidir implementar la solució mitjançant una aplicació web en comptes d'una aplicació tradicional. El principal avantatge que proporciona realitzar una aplicació web és que ofereix un punt centralitzat de configuració i d'instal·lació. Les aplicacions instal·lades al sistema operatiu requereixen una posada en marxa per part dels tècnics i un posterior seguiment en cas d'alguna incidència. En el cas d'una aplicació web, els paquets són instal·lats en un únic servidor i, per tant, es deixen de banda les particularitats de cada ordinador personal. Altres avantatges són la portabilitat i el consum poc elevat de la màquina. Una aplicació web només requereix, per part de l'usuari, un navegador compatible amb l'aplicació, el qual ve pràcticament instal·lat en qualsevol sistema operatiu.

Per tal de dur a terme la implementació s'han considerat bàsicament dos llen-

guatges: Java i CShell. La selecció d'aquests dos llenguatges ha estat principalment perquè són orientats a objectes i per les seves potents APIs. Aquests dos punts permeten, als desenvolupadors, extreure'ns de les particularitats del llenguatge de programació en si i centrar-nos a dur a terme el projecte. Finalment s'ha optat per dur a terme la implementació amb Java ja que la documentació sobre la creació d'aplicacions web i les eines que ofereix són molt extenses. A més, és un llenguatge de programació més conegut pel client i pels mateixos desenvolupadors, de tal manera que s'evita qualsevol tipus de procés de formació.

Per al desenvolupament d'una aplicació Java cal tenir instal·lat i configurat el JDK (Java Development Kit). El JDK és un conjunt de software distribuït per Sun que conté totes les eines necessàries per tal de desenvolupar: compiladors, llibreries, *deployers*, JRE (Java Runtime Environment), etc. Cal especificar que s'ha utilitzat la versió del JDK 1.5.0_19 per mantenir la compatibilitat amb d'altres aplicacions desenvolupades per aquesta versió.

Com no podia ser de cap més manera tractant-se d'una aplicació web, com a model de treball s'ha adoptat el patró MVC (Model Vista Controlador). El patró MVC és un model d'arquitectura del software que separa les dades d'una aplicació, la interfície d'usuari i la lògica de control en tres components diferents.

El patró MVC el va descriure per primera vegada Trygve Reenskaug, un treballador de Smallpark en els laboratoris d'investigació de Xerox. La implementació original es pot trobar en el document "Applications Programming in Smalltalk-80: How to use Model-View-Controller"[1] i s'utilitza freqüentment per la creació d'aplicacions web.

Els tres components del model es defineixen de la manera següent:

- Model. Representació específica de la informació amb la qual el sistema opera. És important destacar que la lògica de dades no només facilita l'accés a aquestes sinó que també assegura la integritat de les dades i permet derivarne de noves. La lògica s'obté a partir del model de dades de l'aplicació.

- Vista. Ofereix les eines necessàries per mostrar la interfície de l'aplicació per tal que l'usuari pugui interactuar-hi.

- Controlador. S'encarrega del control de flux de l'aplicació. Segons les accions de l'usuari a la vista es realitzaran les accions necessàries al model i s'obtindrà la informació necessària per tal de mostrar-la a la vista.

L' esquema general es pot veure a la figura 3.6

Els beneficis d'utilitzar aquest patró com a arquitectura de treball són diversos. Primer de tot afavoreix clarament el treball en equip i l'organització del projecte. La separació de tasques entre capes permet desenvolupar paral·lelament, sempre i quant es defineixin, unes interfícies que mantinguin la mateixa estructura. Aquestes interfícies permeten la integració de diferents tecnologies en cada capa del model, de tal manera que es poden buscar solucions òptimes per cada lògica. L'ús

[1] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html

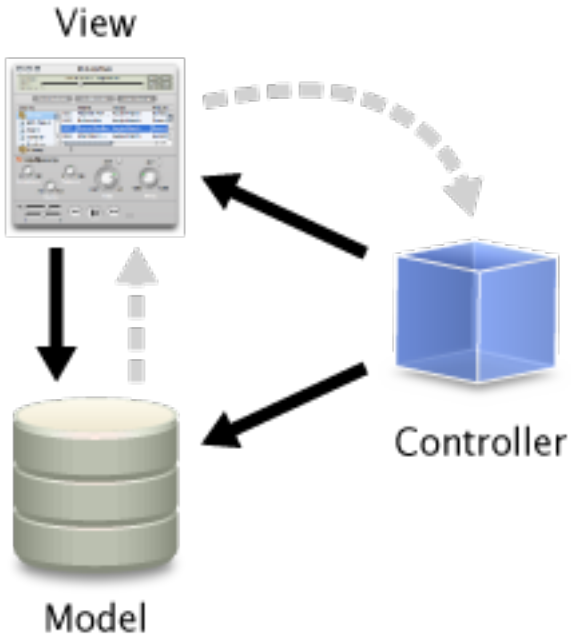

Figura 2.2: Esquema general del patró Model Vista Controlador

d'interfícies també és un altre punt a favor ja que permet establir estereotips de treball de tal manera que la implementació sigui independent a les definicions de les classes. Finalment una de les característiques més importants és que és molt escalable. Es poden modificar les tecnologies aplicades a cada capa i permet ampliar fàcilment qualsevol dels components, sense que els altres es puguin veure afectats en qualsevol dels casos. Una de les propietats que demostra com n'és d'escalable, el model MVC és que permet que dins de cada component hi pugui haver un altre triplet de model-vista-controlador i així recursivament.

A continuació passarem a analitzar les tecnologies utilitzades per cada component i les seves raons.

Model

La representació lògica del model de dades s'ha dut a terme mitjançant Hibernate [7]. Hibernate és una llibreria ORM (Object Relational Mapping) que facilita el mapatge d'atributs entre una base de dades relacional tradicional i el model d'objectes d'una aplicació. Aquesta relació s'estableix mitjançant la definició de les relacions en el codi de l'aplicació, ja sigui mitjançant anotacions o amb fitxers XML.

Hibernate permet desenvolupar classes persistents seguint el llenguatge orientat a objectes. D'aquesta manera, s'aconsegueix que la consulta i modificació de la base de dades sigui molt més dinàmica i flexible. També permet realitzar consultes

Tecnologia	*Problema*
Hibernate	Capa de persistència
JSF	Interfície gràfica
Richfaces	Llibreries de tags
Active Widgets	Taules de presentació
Facelets	Adaptació de JSP a XHTML
Spring Webflow	Controlador de flux

Figura 2.3: Taula resum de tecnologies

SQL amb la seva pròpia extensió anomenada HSQL (Hibernate SQL).

Vista

La vista o interfície gràfica està construïda amb diverses tecnologies: Java Server Faces (JSF) [11], Facelets[8], Richfaces[10] i ActiveWidgets[13].

JSF és una tecnologia pionera per a la creació d'interfícies gràfiques. La programació de la interfície es fa mitjançant components i està basada en esdeveniments. El que permet JSF és treballar d'una manera molt intuïtiva sobre la capa de presentació de com es volen estructurar les pàgines i els esdeveniments que s'han de llançar quan l'usuari realitza una determinada acció.

Per tal de poder reutilitzar components ja creats, s'utilitzen les llibreries de tags anomenades Richfaces. Aquestes llibreries proporcionen els components més comuns, com poden ser botons, llistes desplegables, formularis, etc. Una de les virtuts d'utilitzar Richfaces és que la majoria dels seus components utilitzen Ajax i, per tant, la plana web adopta un dinamisme molt interessant. Malgrat que també proporcionen taules per mostrar continguts, s'ha utilitzat una altra llibreria per crear-les anomenada ActiveWidgets. La raó per la qual ha estat així són qüestions d'estètica i velocitat d'implementació: els desenvolupadors estem més acostumats a treballar amb aquesta llibreria ja que hi hem treballat en d'altres projectes. El format de treball de JSF és jsp i això suposa alguns inconvenients. Per tal de treballar amb d'altres formats es va optar per utilitzar Facelets, principalment per tal de passar de planes jsp a xhtml. D'aquesta manera, quan la plana sol·licitada s'ha de renderitzar per mostrar-la al usuari es tracta com un conjunt, es crea un arbre de la pàgina i s'avalua sencer.

Cal destacar que la integració de JSF, Facelets i Richfaces és una tècnica bastant pionera; en conseüència, no es disposa de gaire material de referència.

Controlador

El controlador s'ha implementant mitjançant Spring Web Flow [4]. Spring Web Flow és un mòdul del framework Spring amb la finalitat de definir i gestionar els fluxos de pàgines dins d'una aplicació web. El flux de pàgines d'una aplicació web consisteix en la seqüència de pàgines per on passa l'aplicació en funció de la

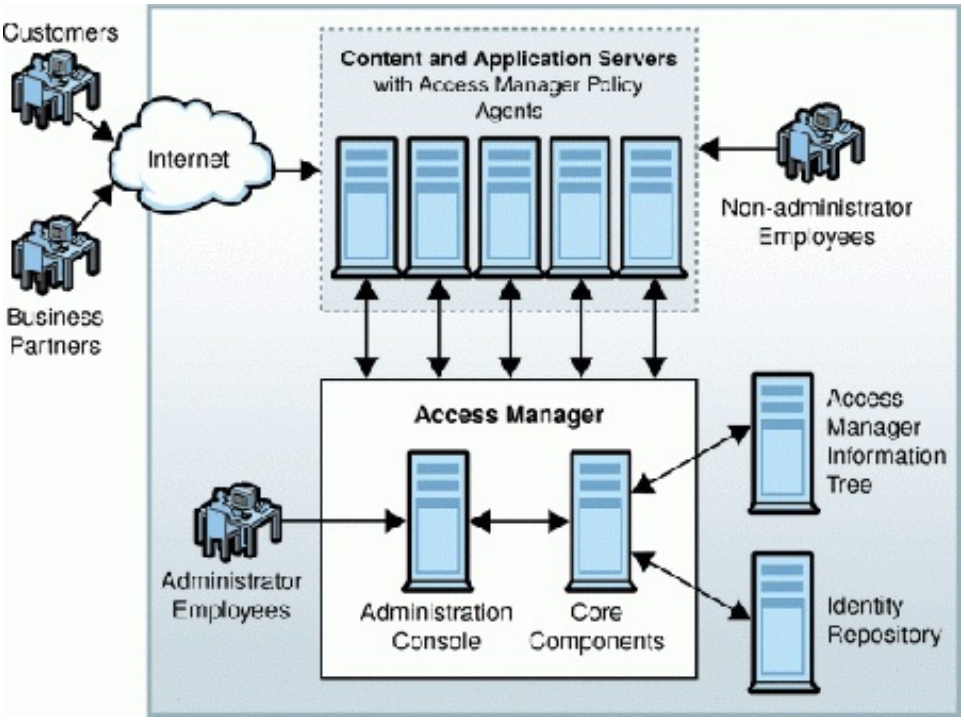

Figura 2.4: Arquitectura Access Manager 7.1

interacció amb l'usuari. Segons les opcions que escull l'usuari, resultats d'operacions de procés, accions realitzades,..., l'aplicació seguirà una ruta de pàgines o una altra. La definició d'aquestes rutes, anomenades flows, es duu a terme mitjançant fitxers XML, que tenen unes anotacions pròpies. Aquestes anotacions permeten definir diferents estats per tal de controlar el flux d'execució: estats inicials, de vista, de decisió, de control, entre d'altres.

2.3 Seguretat

Cal dotar de seguretat a l'aplicatiu LWB en dos àmbits: autenticar els usuaris a l'aplicatiu i protegir determinats recursos. Per assolir aquests dos objectius s'ha decidit utilitzar Access Manager 7.1 [12].

Acces Manager és un producte desenvolupat per Sun Microsystems, que proporciona serveis d'autenticació Single Sign On (SSO) d'una manera transparent per a entorns distribuits. Una de les seves principals virtuts és que permet utilitzar múltiples repositoris d'usuaris, distribuir-los en diferents regnes i administrar-los desde un punt centralitzat. Una arquitectura aprofitant-se dels serveis d'Access Manager es pot veure a la figura 2.4

Per realitzar l'autenticació des de l'aplicació cal instal·lar al servidor un agent d'Access Manager, que s'encarrega de gestionar amb el servidor d'Access Ma-

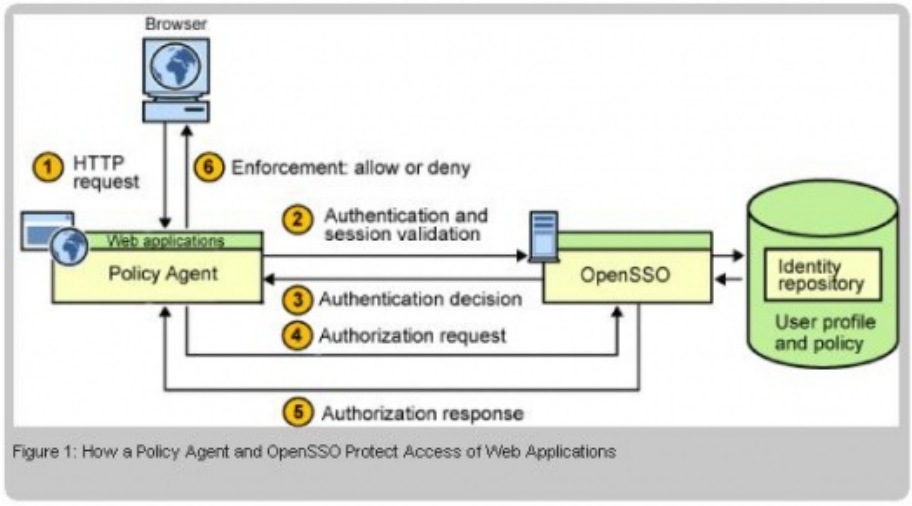

Figura 2.5: Arquitectura Policy Agent

nager l'autenticació de l'usuari sense que calgui afegir moduls d'autenticació a l'aplicació. Aquest agent permet aprofitar-se d'una arquitectura SSO. Cal destacar que també hi ha la opció de realitzar polítiques d'accessos a recursos protegits, segons objectes que tinguis assignats les identitats. El flux que segueix l'agent quan ha d'autenticar un usuari es pot comprovar a la figura 2.5.

2.4 Estructura del projecte

Per tal de seguir d'una manera estructurada el model MVC s'han creat les estructures de projectes següents:

- CLIENT
  - Data
  - Service
  - UI
- SERVIDOR
  - Data
  - Service
  - Server
  - Server.UI
  - Webservice.client
  - Webservice.servidor
- RELEASES

Projectes comuns

Tot i que les aplicacions client i servidor han estat dissenyades per a funcions diferents, comparteixen els projectes de Data i Service. L'explicació de per què comparteixen projectes és ben senzilla: com que la comunicació entre client i servidor es duu a terme mitjançant la base de dades, les dues han de tenir les classes i mètodes adients per accedir-hi i tractar les dades adequadament. Aquestes "eines"són compartides ja que sovint són funcions molt comunes, com obtenir les peticions en curs, els llistats d'una aplicació, els paràmetres d'un llistat, etc.

- Projecte Data. És l'encarregat d'implementar l'accés a la capa de persistència (base de dades).
- Projecte service. La idea d'aquest projecte, com el seu nom indica, és oferir serveis a les altres interfícies. Principalment té dues funcionalitats. La primera és establir d'una interfície entre els altres projectes i el projecte Data. D'aquesta manera es poden agrupar les funcions més comunes que s'utilitzaran en l'aplicació en un sol projecte, per exemple: recuperar totes les aplicacions d'un usuari,obtenir els llistats i paràmetres d'una aplicació o afegir una petició a la base de dades. L'altra funcionalitat és oferir servei a tots els altres projectes. S'entén, per oferir servei, agrupar totes les classes que no són pròpies del projecte en què s'han de tractar però que hi són necessàries. Per exemple aquí estaran ubicades les classes per determinar l'estat d'una petició o per obtenir la versió d'un format en fitxer Excel , entre d'altres.
- Projecte releases. Aquest projecte conté tot les eines necessàries per al desenvolupador, per treballar en el projecte i generar els paquets de l'aplicació. S'hi poden distingir diferents carpetes:
  - Lib: carpeta que conté les llibreries que li són necessàries a l'aplicació per funcionar correctament. En aquest cas hi tenim dos subcarpetes, client i servidor, on estan ubicades les llibreries corresponents.
  - Logs: cada aplicació disposa del seu fitxer de logs pertinent. De la mateixa manera no serà la mateixa configuració de logging per a desenvolupament que per a producció. Aquesta carpeta conté els fitxers de propietats necessaris per al nostre sistema de logging.
  - Scripts: on hi ha ubicats els dos scripts per generar els paquets de les aplicacions. Aquests scripts estan implementats mitjançant Apache Ant i és una manera molt eficaç d'evitar feines repetitives.

Projectes servidor

Deixant de banda dels projectes comuns, els projectes que contindrà l'aplicació servidor seran:

- Projecte server. Com el seu nom indica conté tota la lògica del servidor que genera les peticions. La lògica està implementada en Java, utilitzant Quartz. Quartz es un projecte Open Source que permet planificar tasques (anomenades Jobs) donats uns paràmatres de temps.
- Projecte Server.UI. Aquest projecte conté la interfície gràfica de la consola d'administració del servidor i els controladors necessaris. Aquesta consola està implementada mitjançant les mateixes tecnologies que la interfície gràfica del client.
- Projecte Webservice.Client. És on hi ha les classes relacionades amb la comunicació mitjançant el webservice del projecte generador de documentació.
- Projecte Webservice. Servidor. És on hi ha la implementació del nostre webservice per tal de rebre la documentació associada.

Projecte client

En el cas del client, només tindrà un projecte propi. Serà l'anomenat UI (User Interface), que contindrà les planes web i els controladors necessaris per aquestes planes.

2.5 Entorn de desenvolupament

Les eines utilitzades per al desenvolupament són comunes per a tots els treballadors del projecte. S'han buscat eines de referència del mercat, estables i que facilitin al desenvolupador les tasques del projecte. A banda dels servidors que cal tenir instal·lats per a què el desenvolupador pugui fer proves en local (explicat en 3.8), les eines de més pes dins del projecte han estat: Eclipse Platform per a Java, Oracle SQL Developer i Microsoft Visual Sourcesafe.

Eclipse Platform

Eclipse Platform en la seva versió Java és un entorn de treball (IDE) per als programadors Java que proporciona eines per al desenvolupament d'aplicacions. A més, se li poden incorporar plugins desenvolupats per la comunitat que faciliten encara més la feina. Concretament, per aquest projecte s'ha utilitzat els plugins de Visual Source Safe i Ant. El plugin de Visual Source Safe consegueix integrar Eclipse amb el repositori, de tal manera que no cal tenir oberts els dos entorns. Des d'Eclipse es poden realitzar la majoria de tasques des de als menús de l'aplicació. El plugin per a Ant permet desenvolupar i executar els scripts Ant desde l'entorn IDE. És una manera fàcil de treballar que evita tasques repetitives.

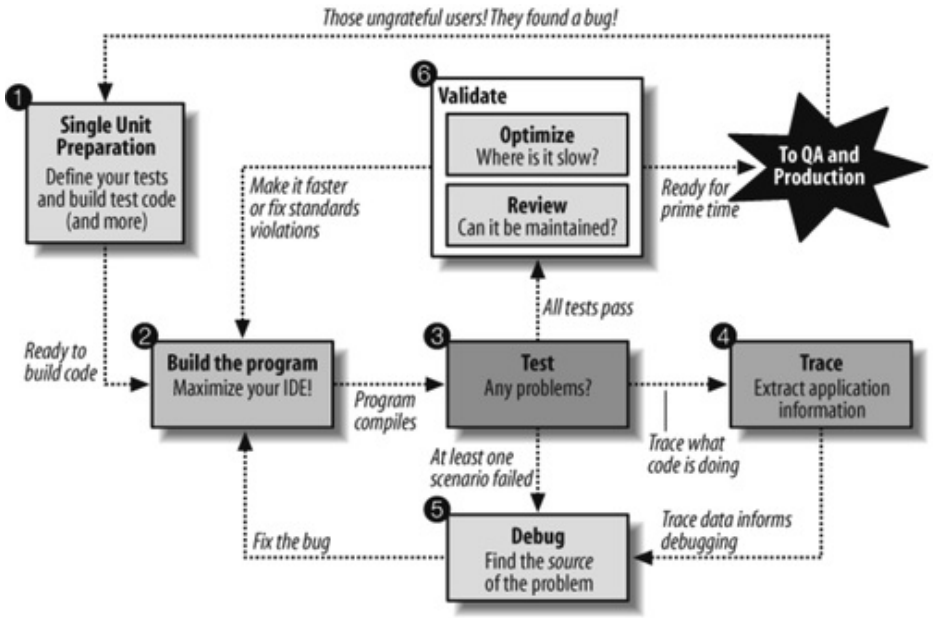

Figura 2.6: Cicle de vida de desenvolupament

Microsoft Visual SourceSafe

Visual Sourcesafe és el repositori de documents que s'ha utilitzat per al projecte. Aquest repositori permet treballar en grup a les persones que treballen en el projecte. D'aquesta manera es garanteix que el desenvolupador sempre treballa amb l'última versió del projecte i que mai no hi haurà diverses persones treballant en el mateix mòdul.

Oracle SQL Developer

SQL Developer és un entorn de treball gràfic per a bases de dades Oracle. Ha estat una eina bàsica per al desenvolupament, ja que sovint les consultes o modificacions a la base de dades són poc intuïtives si s'intenten de fer per línia de comandes. A més, permet gestionar diverses connexions a diferents bases de dades a la vegada, de manera que agilitza el treball amb els diferents entorns de treball.

2.6 Metodologia de desenvolupament

Durant el desenvolupament de l'aplicació s'ha anat fent proves de cada mòdul de tal manera que se'n garanteix el funcionament individual. En general, s'ha seguit un procés iteratiu incremental que es pot veure gràficament en la figura 2.6

El procés seguit ha estat el següent:

1. El desenvolupador planteja i raona la solució per complir les especificacions dels documents de requeriments.

2. Es creen les definicions dels moduls: crear les classes, definir-ne els mètodes, els paràmetres i els tipus de retorn, etc. Tot això sense implementar-ho, simplement defineix i s'implementa el mínim perquè passi el procés de compilació.

3. Una vegada definit el mòdul, es crea el joc de proves corresponent per a la validació del mòdul. En aquest punt es pot utilitzar alguna tecnologia que faciliti aquesta tasca com, per exemple, Junit [9]. En el nostre model no hem utilitzat cap eina sinó que s'ha implementat manualment.

4. S'implementa el mòdul seguint les especificacions dels documents.

5. Es passa el joc de proves. En cas que el resultat sigui correcte es passa a desenvolupar un altre mòdul. Si fos erroni, es busca l'error i es torna al punt 3.

Em cas de que hi hagi un canvi en els requeriments, cal tornar a començar pel primer punt revisant totes les tasques. Cada vegada que es desenvolupa un mòdul nou o se'n modifica un, cal passar els jocs de proves de tots els mòduls per assegurarnos que el canvi no ha suposat cap problema i que l'aplicació es manté estable. Una vegada s'entrega un conjunt de mòduls al client per què el validi, en cas que es trobi algun bug cal tornar al punt 3 i afegir la casuistica adient al joc de proves.

Aquest mètode es pot veure a la figura 2.6.

Capítol 3

Implementació i desplegament

En aquest capítol es comentaran els detalls d'implementació de l'aplicació. La implementació de l'aplicació es pot dividir en tres apartats: el model de dades, l'aplicació client i l'aplicació servidora. Dotar de seguretat l'aplicació es pot tractar com un apartat a banda, tenint en compte que es configura de manera similar al client i al servidor. Finalment, també es definirà l'arquitectura de servidors adoptada i tot el conjunt d'entorns que s'han construït.

3.1 Model de dades

El model de dades és el punt de partida de l'aplicació. Cal dissenyar un model consistent i eficaç que permeti emmagatzemar la informació de l'aplicació. En aquest punt se'n distingeixen dos apartats: el disseny de la base de dades i com aquest disseny s'ha integrat amb l'aplicació mitjançant hibernate. Cal destacar que tant l'aplicació client com la servidora comparteixen el mòdul de la capa d'accés a dades. La causa d'això és que el seu mitjà de comunicació és la base de dades i, per tant, necessiten de les mateixes funcions per interactuar i obtenir la informació.

3.1.1 Disseny

El disseny del model de dades es pot veure en la figura 3.1

A continuació es detallen les entitats i una breu explicació de la seva utilitat:

- Llistat: encarregada de contenir les dades corresponents al llistat. En ella s'hi trobaran atributs com el nom, la descripció i la prioritat. També s'hi troben indicats els noms de les taules on es guarden els resultats de l'execució del procediment.
- Paràmetres del llistat: és on es guarda la descripció dels paràmetres del llistat, per exemple, el tipus de cada paràmetre, el seu nom, el seu format, si és obligatori, etc.

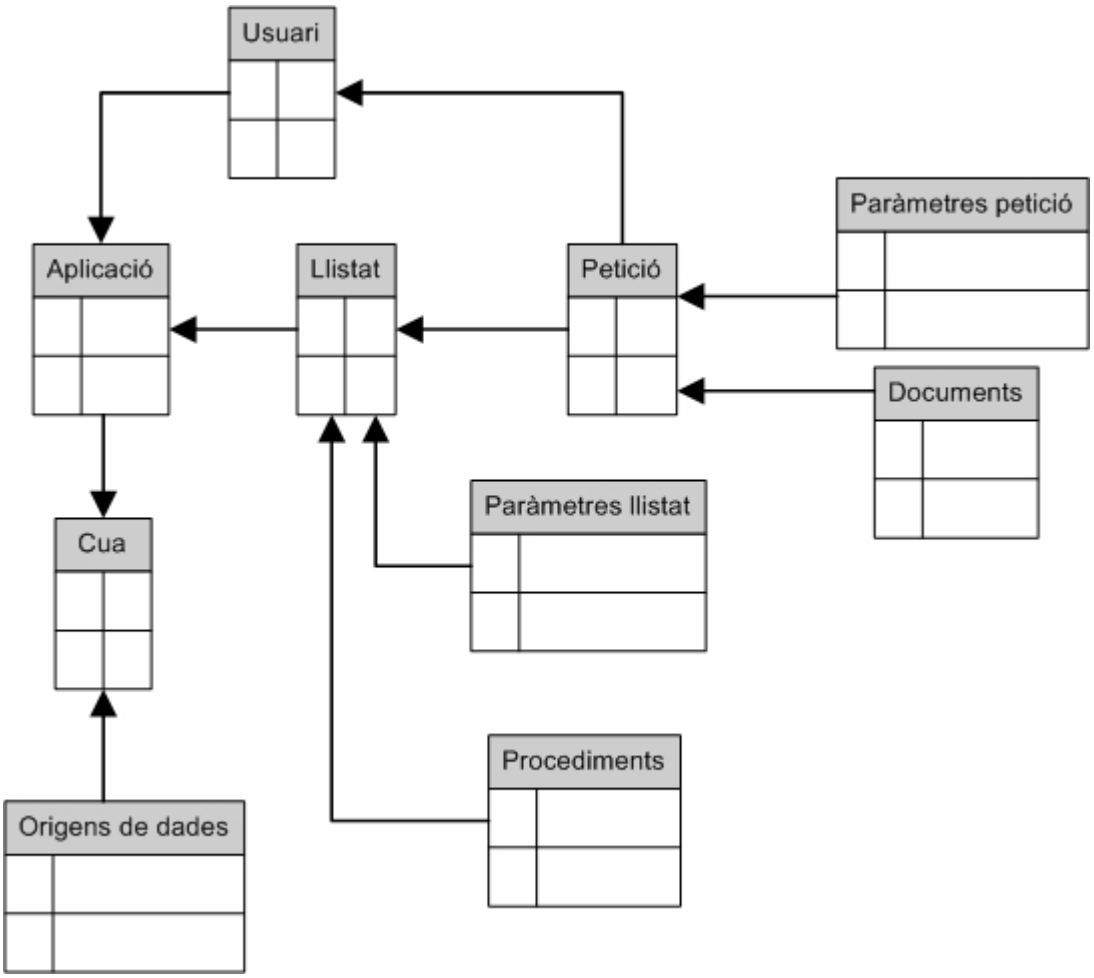

Figura 3.1: Diagrama entitat relació de LWB

- Procediment: és on es guardaran els procediments que executaran els llistats. Cada procediment guarda la seva sortida en una taula diferent, per tant, per cada procediment associat al llistat hi haurà una taula indicada a l'entitat llistat. Hi ha la restricció que poden haver-hi només 15 procediments com a màxim associats a un llistat.

- Petició: emmagatzema les dades de la petició feta per l'usuari. Alguns atributs són: el format d'exportació, la prioritat i el seu estat. També conté els atributs de dates de control: quan s'ha relitzat la petició, quan s'ha començat a executar, quan ha finalitzat, la data d'anulació i quan s'ha notificat a l'usuari.

- Paràmetres de la petició: conté els valors dels paràmetres de la petició realitzada.

- Documents: és on es guarden les rutes dels documents que s'han generat després del procés d'exportació.

- Usuari: conté les dades de l'usuari, per exemple el seu nom, cognoms, codi d'usuari, etc. També conté l'adreça electrònica on s'han d'enviar les notificacions de LWB.

- Aplicació: és on es guarda el nom de l'aplicació i les seves dades. Una aplicació és un conjunt de llistats, s'utilitzen per donar accés als usuaris als llistats.

- Cua: s'emmagatzemen les dades de la cua. Cada cua té associades un conjunt d'aplicacions de tal manera que les peticions de diverses aplicacions s'executen sempre per la mateixa cua. És imprescindible que els procediments de la petició estiguin definits en l'origen de dades de la cua.

- Origen de dades: conté les dades de connexió de la cua, com per exemple, les direccions ip dels servidors de bases de dades, els ports, els identificadors SID i l'usuari i password per realitzar la connexió.

3.1.2 Integració

La integració del model de dades amb la nostra aplicació es duu a terme mitjançant Hibernate. Tal i com hem explicat anteriorment, aquesta implementació resideix dins el projecte Data i s'hi poden distingir els elements següents:

- Les classes que representen les entitats de la base de dades. Per a cada entitat hi haurà una classe, excepte en el cas que l'entitat a la base de dades tingui una id composta, on es generarà una classe a part per representar aquesta id. Segons ens dicta Hibernate, cada classe contindrà els atributs membres com a privats i amb les seves corresponents funcions d'accés (getters i setters). Cal destacar que, per tal de mantenir les classes on Hibernate fa el mapatge

dels objectes, es creen dues classes: una d'abstracta que conté exactament els atributs que conté la taula de la base de dades, i una que implementa aquesta classe per si es volen afegir nous atributs o mètodes. Es considera una bona pràctica fer-ho així ja que el desenvolupador sempre haurà de mantenir el tipus dels atributs de la classe abstracta i no crearà conflictes amb Hibernate.

- "Mappings"d'Hibernate. Són els mapatges que utilitza Hibernate per relacionar cada entitat de la base de dades amb les classes definides anteriorment. Cada mapatge entre una taula i una classe conté els seus noms respectius. A continuació hi ha la relació entre cada columna de la taula i el seu corresponent atribut, especificant el tipus de dades de Java. Cal destacar que Hibernate permet el mapatge d'atributs no només de tipus base de Java (Integer, String, Double,..) sinó també de classes definides per l'usuari. Això ens proporciona una manera molt còmoda de treballar i molt intuïtiva. Per exemple, suposem que definim el mapatge de la classe TAplicacionData. Com sabem. cada aplicació conte tot un seguit de llistats. Es pot definir el mapatge de tal manera que hibernate recupera i omple els objectes del tipus llistat directament i els col·loca dins la llista corresponent.

- Objectes DAO (Data Access Objectes). El patró DAO s'utilitza per encapsular tots els accessos a la font de dades, de manera que s'afegeix una altre capa de persistència. D'aquesta manera s'aconsegueix tractar independentment com s'obtenen les dades depenent de la font i que es pugui modificar sense alterar el comportament de l'aplicació. El patró DAO forma part del core de J2EE i és una bona pràctica usar-lo. Mitjançant Hibernate, el DAO heretarà de BaseHibernateDAO de tal manera que podrem obtenir la sessió d'Hibernate i realitzar consultes contra la base de dades. Cal destacar que el llenguatge pròpi d'hibernate HQL és molt potent i permet realitzar qualsevol tipus de consulta igual que SQL. En els objectes DAO és molt comú generalitzar la funció per tal que amb una funció es puguin realitzar la consulta adient només sobre un atribut. Així, s'evita repetir codi i s'aconsegueix que el manteniment d'aquest sigui fàcil.

Es poden veure exemples a l'annex de l'utilització d'aquests punts, concretament a l'apartat A

3.2 Client

L'aplicació client serà la destinada a la interactuació dels usuaris. S'ha dissenyat una aplicació senzilla i intuïtiva, que compleixi les especificacions del llibre d'estils del client. En les posteriors seccions es detallen com seran les pantalles de l'aplicatiu client i la funcionalitat de cadascuna.

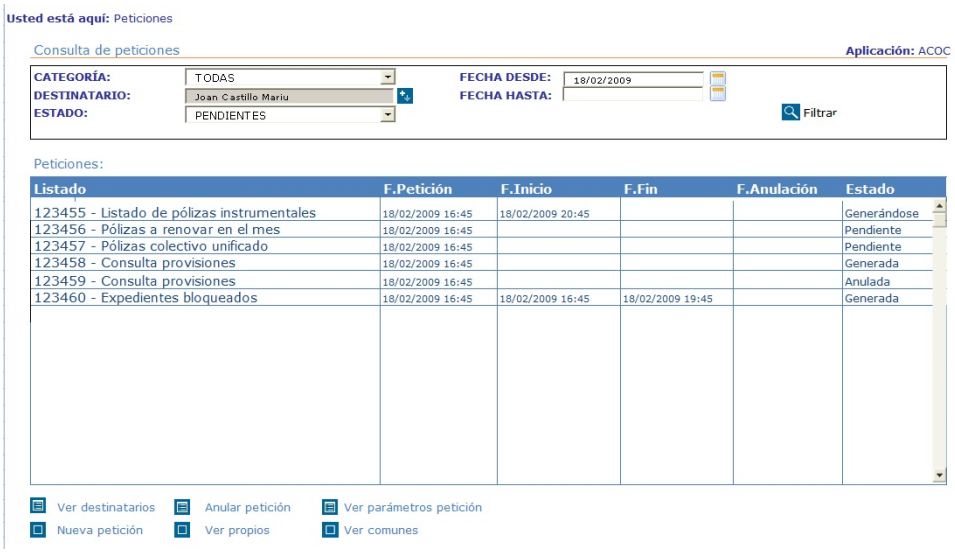

Figura 3.2: Consulta de peticions

3.2.1 Pantalla de consulta de peticions

Aquesta serà la pantalla principal de l'aplicatiu. S'hi poden veure la llista de peticions del servidor segons els criteris de filtratge. Aquests criteris són els següents:

- Categoria. Segons la categoria a què pertanyen les aplicacions.
- Destinatario. Segons el destinatari de la petició.
- Estado. Segons l'estat de la petició. Aquest estat pot ser: pendiente, en curso, finalizada o errónea.
- Fecha desde. Filtratge d'una data en davant
- Fecha hasta. Filtratge fins a una data en concret

A més, es disposarà d'una botonera a la part inferior de la pantalla per realitzar accions. Hi haurà tres accions que s'activaran en el moment que es seleccioni alguna de les peticions de la taula. Aquestes accions són: Ver Destinatarios, Anular petición i Ver parámetros petición.

- Ver destinatarios. Permet veure els destinataris de la petició.
- Anular petición. Permet anular la petició en cas que aquesta estigui només en estat pendent.
- Ver parámetros petición. Mostra els paràmetres de la petició en una finestra de tipus popup.

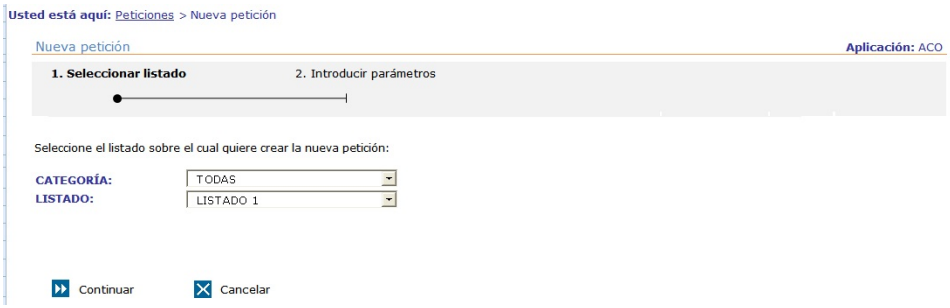

Figura 3.3: Pantalla crear una petició: selecció del llistat

A banda, a sota de la botonera anterior, es disposa de tres botons que sempre estaran actius i són:

- Nueva petición. Llança el flux d'execució de generar una nova petició.
- Ver propios. Permet veure les carpetes pròpies del servidor on es guarden els llistats (per a peticions manuals).
- Ver comunes. Permet veure les carpetes comunes del servidor on es guarden els llistats (per a peticions automàtiques).

Aquesta pantalla es pot veure a la figura 3.2.

3.2.2 Creació d'una petició

La creació d'una petició serà el nostre principal fil d'execució. A la part inferior de tot el procés sempre es disposarà dels botons de navegació 'Atrás', 'Siguiente' i 'Finalizar' que s'encarregaran que l'usuari pugui moure's correctament entre el procés. La petició no es posarà en cua fins que tot el procés estigui completat al 100%, però mitjançant les dades de la sessió es guarden els valors intermedis.

Una creació d'una petició sempre estarà formada per dues pantalles i una altra opcional.

1. Selecció del llistat. Es disposa de dues llistes desplegables per seleccionar la categoria i el llistat. Si es selecciona una categoria, aquest filtra automàticament els llistats de la categoria en la llista desplegable inferior.

   Aquesta pantalla es pot veure en la figura 3.3.

2. Introducció dels paràmetres. Una vegada seleccionat el llistat, cal introduir els paràmetres del llistat i la seva planificació. Aquesta planificació serveix per indicar a partir de quan volem que s'executi el llistat i és una mesura per no saturar el servidor amb llistats molt llargs.

   La introducció dels paràmetres es recuperen de la base de dades i es pinten en una taula dinàmicament inicialitzats amb els valors per defecte. Els

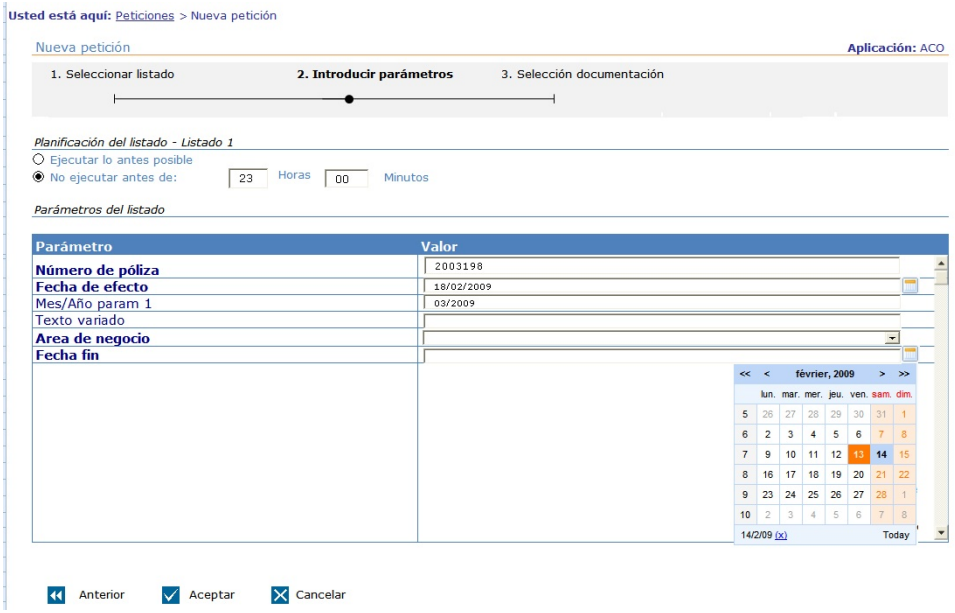

Figura 3.4: Pantalla crear una petició: paràmetres i planificació

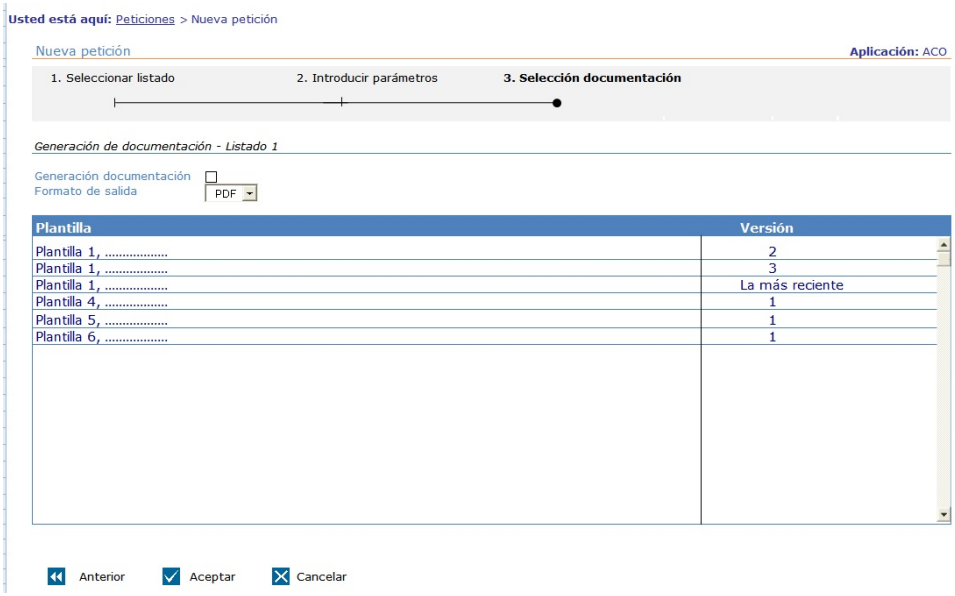

Figura 3.5: Pantalla crear una petició: selecció de documentació

paràmetres obligatoris estàn remarcats en negreta. Una vegada introduïts els paràmetres es realitzen les validacions, com per exemple, comprovar que tots els paràmetres obligatoris estan indicats o el format correcte dels paràmetres.

Aquesta pantalla es pot veure en la figura 3.4.

3. Generació de documentació. En el cas que el llistat tingui plantilles de documentació es disposarà d'aquesta pantalla per seleccionar si es vol generar. En cas que així sigui, cal seleccionar el format i la plantilla que es vol utilitzar.

   Aquesta pantalla es pot veure en la figura 3.5.

Una vegada finalitzat aquest procés, es mostra un popup de confirmació que s'ha finalitzat el procés. En cas afirmatiu es visualitzarà la nova petició a la taula de peticions en estat pendent. Durant la creació d'una petició hi ha un esquema recordatori a la part superior, que indica quin dels passos de la creació d'una petició s'està realitzant.

3.3 Servidor

3.3.1 Planificació

La planificació de les tasques del servidor es duu a terme mitjançant Quartz. S'entén per planificar una tasca com aplicar una configuració per tal que s'executi un cert procediment amb una periodicitat establerta.

Aquestes tasques planificades en el context de Quartz s'anomenen jobs. En el nostre cas hi tindrem configurats tres jobs:

- ServerJob: la tasca principal que s'encarrega de comprovar és si hi ha peticions i executar-les. La planificació establerta és cada dos minuts, de dilluns a dissabte tot el dia, i diumenge cada dos minuts, de les 9:00 a les 24:00 hores.
- DeleteJob: s'encarrega d'esborrar els llistats generats de més de tres mesos d'antiguitat. La planificació establerta és cada dilluns a les 6:00 hores.
- TasquesPeriodiquesJob: consisteix en un job genèric on s'hi pot afegir qualsevol tipus de tasca que faci falta en un moment determinat. La planificació establerta és, dilluns, a les 4:00 hores.

Tot i que Quartz es pot implementar directament en Java, en el nostre cas per simplificar-ho hem fet les definicions en fitxers XML . Per tal de definir un job correctament, cal la definició del job i associar-hi un disparador (trigger). En el moment en que el trigger s'activi executarà la tasca, normalment implementada en Java. Per exemple per implementar el trigger de les tasques periòdiques cal el codi següent:

```
1 <bean name="lwbTasquesPeriodiquesJob" class="org.
     springframework.scheduling.quartz.JobDetailBean">
```

```
2     <property name="jobClass" value="es.sgch.lwb.server.
          TasquesPeriodiques" />
3   </bean>
4
5   <bean id="lwbTasquesPeriodiquesTrigger" class="org.
        springframework.scheduling.quartz.CronTriggerBean">
6       <property name="jobDetail">
7                   <ref bean="lwbTasquesPeriodiquesJob"/>
8           </property>
9           <property name="cronExpression">
10                  <value>0 0 4 ? * MON</value>
11          </property>
12  </bean>
13
14  <bean class="org.springframework.scheduling.quartz.
        SchedulerFactoryBean">
15      <property name="triggers">
16          <list>
17                  <ref bean="lwbTasquesPeriodiquesTrigger" />
18          </list>
19      </property>
20  </bean>
```

S'hi distingeixen tres parts:

- La definició de la tasca i la classe que s'executarà quan el trigger es dispari. En aquest cas, la classe per executar és es.sgch.lwb.server.TasquesPeriodiques.
- La configuració del trigger, on s'especifica el job que s'ha d'executar i la planificació. La planificació és la propietat 'cronExpression' i ara està configurada perquè s'executi cada dilluns a les 4:00 hores. Aquesta planificació és molt flexible i permet fer múltiples configuracions. [1]
- Afegir el trigger a la planificació del Quartz, on simplement es referencia el trigger creat perquè sigui tingut en compte.

3.3.2 Execució

Com hem pogut veure anteriorment, tenim planificada una tasca de quartz que cada dos minuts executa el nostre job d'execució de peticions. Aquest job crea les cues d'execució si encara no estan creades. Crear una cua d'execució és crear un nou thread, per tant cada cua dispondrà del seu propi fil d'execució. La cua té associat el seu objecte de dades, on trobem les dades pròpies de la cua. En cas que s'estigui creant de nou, cal inicialitzar-lo. Una vegada creades totes les cues, s'inicia en paral·lel l'execució de totes les cues. La utilització de threads concurrents fa que s'estiguin executant Diverses peticions de cues diferents a la

[1] http://www.quartz-scheduler.org/docs/tutorials/crontrigger.html

vegada i que per tant, es millori l'eficiència notablement. A més, mai no hi haurà variables compartides per diverses threads i, per tant, els problemes derivats de la concurrència són mínims. L'execució de cada un dels fils d'execució funciona de la manera següent:

1. Es comprova si la cua ja s'està utilitzant. En aquest cas no es fa res i s'esperarà que, al cap de dos minuts, es torni a fer la comprovació. Per tal de saber si una cua està en execució, es disposa d'un flag en les dades de la cua que així ho indica.

2. Es comprova la data de generació de les peticions automàtiques. En el cas que aquesta sigui anterior a l'actual cal generar-les i per tant es cridarà a la funció encarregada de fer-ho.

3. S'obté la següent petició que cal executar. Si no hi ha peticions pendents, s'acaba l'execució de la cua.

4. S'executa la petició obtinguda i es torna al pas 1.

3.3.3 Generació de peticions automàtiques

La generació de les peticions automàtiques es fa obtenint les aplicacions associades a la cua. Una vegada obtingudes les aplicacions, per cada aplicació cal obtenir els llistats que estan configurats com a automàtics i els que per la seva configuració cal executar. Per tal de controlar-ne la periodicitat s'ha establert una representació de la planificació pròpia. Hi ha 3 tipus de planificacions: diària, mensual i trimestral.

- Diaria o semanal. S'utilitza un array de 7 bits per indicar els dies de la setmana que s'ha d'executar un llistat. Un 1 a la posició del dia indicarà que s'ha d'executar. Per exemple: 1010101 executarà el llistat dilluns, dimecres, divendres i diumenge.

- Mensual. Indicarà en un camp específic el dia del mes en què s'ha d'executar el llistat.

- Trimestral. S'indicaran en tres camps específics les dates en què cal executar el llistat.

La lògica, que controla quan un llistat ha de ser peticionat perquè així ho indica la planificació, no és gaire complexa, es tracta d'obtenir la data actual i comparar-la amb les diferents planificacions a executar. Una vegada obtinguts els llistats, cal crear les peticions corresponents i posar-les en cua. Abans de fer-ho, cal validar els paràmetres de la petició. Els paràmetres que ses posaran a la petició seran sempre els paràmetres per defecte. Per tant, si hi ha algun paràmetre per defecte que no està indicat i és obligatori hi haurà un error de validació i la petició no es durà a terme. La prioritat que se li assignarà a la petició serà sempre la mateixa que tindrà el llistat.

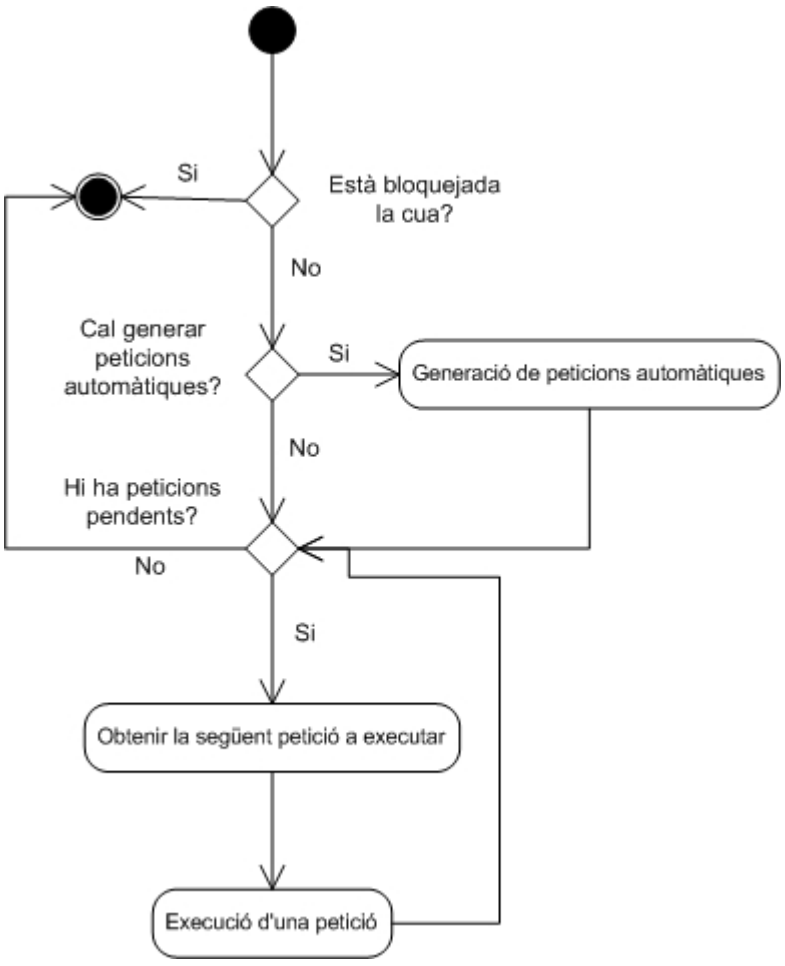

Figura 3.6: Flux d'execució del servidor, cada dos minuts

3.3.4 Obtenció de la següent petició per executar

Per tal de saber quina és la següent petició per executar, cal tenir en compte que les peticions poden tenir assignada aquella cua perquè la seva aplicació pertany a aquella cua o perquè se li ha especificat que vagi per aquella cua expressament. El funcionament és així per tal de poder establir cues per a treballs ràpids, on es pugui independitzar l'aplicació de la cua en casos puntuals. De tota manera el funcionament per defecte és que a tota nova petició (ja sigui manual o automàtica) se li assigna la cua de l'aplicació a la què pertany. Malgrat això, la prioritat hi juga un paper important i sempre serà el factor més decisiu a l'hora d'escollir una petició. Els passos que seguir per tal d'obtenir la següent petició per executar seran:

1. Obtenir les peticions pendents assignades per cua. Guardar la de prioritat més alta, en cas que hi hagi prioritats iguals, sempre la de data més antiga.
2. Obtenir les aplicacions de la cua.
3. Per cada aplicació, obtenir la petició més prioritària i més antiga.
4. De totes les peticions assignades per aplicació, guardar la de prioritat més alta i la més antiga.
5. Comparar la petició assignada per cua amb la assignada per aplicació. En cas d'igual prioritat, la següent petició per executar serà l'assignada per cua (fast queue).

3.4 Gestió de peticions

Una vegada seleccionada la petició, passarem a executar-la. Cal destacar que no hi ha diferència, pel que fal al flux d'execució, en cap de les peticions, ja sigui manual, automàtica, una petició amb documentació o fusió manual. Això és perquè el motor segueix els mateixos fil d'execució per tots els casos i, segons la informació emmagatzemada a la petició, es realitzen les accions que pertoquen. Per executar una petició, serà necessari obtenir les dades de la seva aplicació i els seus paràmetres. Els passos que cal seguir seran:

1. Comprovar si és una petició de fusió manual. En cas afirmatiu, caldrà anar a 4.
2. Si és una petició sobre un llistat, caldrà obtenir els procediments relacionats al llistat i executar-los.
3. Exportar els resultats segons el format indicat (Excel97, OpenXML o TXT).
4. Es comprova si s'ha de demanar la documentació associada i es demana, si s'escau.

5. S'envia un correu electrònic a l'usuari conforme la petició ha estat executada, amb el resultat obtingut.

En qualsevol dels passos del procés d'execució, si es produeix una excepció o un resultat erroni s'envia un correu a l'usuari indicant que no s'ha pogut generar el llistat i la causa.

3.4.1 Execució de procediments

En el nostre motor d'execució els procediments poden ser de dos tipus: procediments Java o procediments emmagatzemats en una base de dades.

Procediments Java

Els procediments Java afegeixen una funcionalitat a l'aplicació que la fa més modular i extensible. No era un requeriment del projecte però es va entendre que era una bona manera de deixar portes obertes a futures extensions del projecte. Una possible aplicació d'un procediment Java podria ser que quan es peticiona un llistat amb informació sensible s'enviï un correu a determinades persones perquè n'estiguin al corrent.

Per tal de crear un procediment Java cal implementar la interfície IPostProceso. Aquesta interfície és de la manera següent:

```
1 public interface IPostProceso {
2         public void ejecutar(TListadosData oListado,
3                 TPeticionesData oPeticion,
4                 List<TParametrosData> parametrosListado,
5                 List<TParametrosPeticionData>
                      parametrosPeticion)
6         throws PostProcesoException, Exception;
7 }
```

Com podem veure és molt senzilla. Disposa d'un mètode "ejecutar"que hauria de contenir el cos del procediment que es vol executar. Els paràmetres del mètode proporcionen al desenvolupador, tota la informació que li pot ser necessària a l'hora de dur a terme el procediment: informació del llistat i de la petició, els paràmetres del llistat i els seus valors. Per tal d'unificar el control d'errors, també s'ha creat una excepció que el desenvolupador podrà llançar quan ell desitgi.

Una vegada explicat com es creen nous procediments Java, podem entendre com s'executen. A la base de dades es disposa del nom de la classe que conté el procés. Mitjançant reflection, s'instancia aquesta classe i d'aquesta manera ja es pot llançar el mètode 'ejecutar' i obtenir el resultat. Reflection és una tècnica que s'utilitza en la programació per tal que es pugui canviar dinàmicament el codi a executar, com és en el nostre cas. En el nostre motor d'execució no sabem la classe que s'haurà d'executar fins que en llegim el nom a la base de dades i per tant, es fa

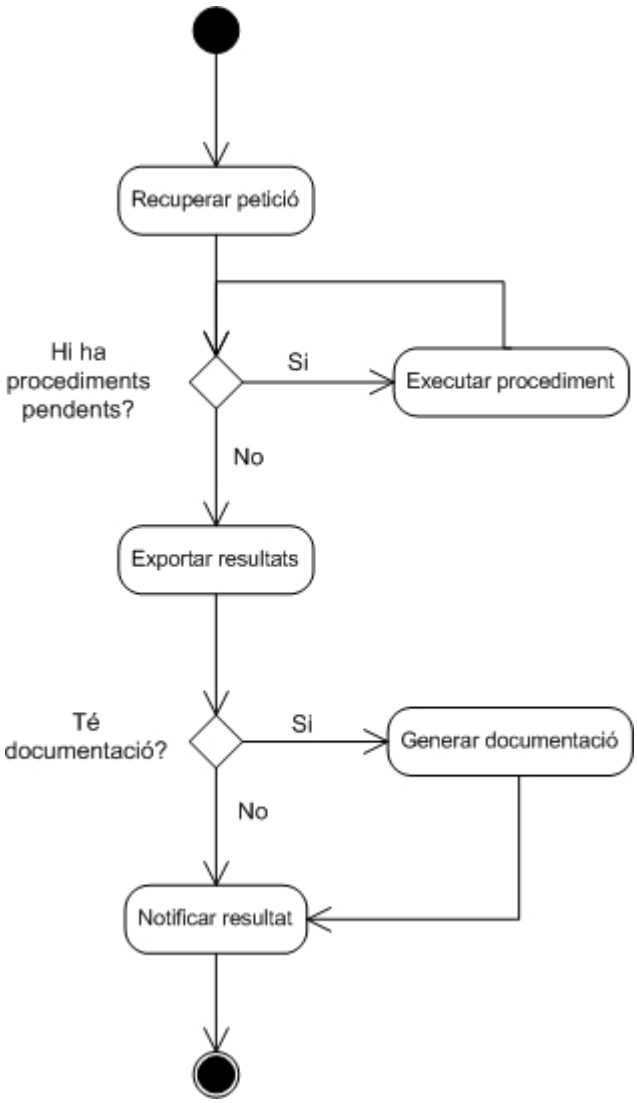

Figura 3.7: Detall d'‘Execució d'una petició' (figura 3.6)

necessari l'ús d'aquesta tècnica. En el nostre cas, reflection s'utilitza de la manera següent:

```
Class c = Class.forName(classeAExecutar);
Object objecte = c.newInstance();
IPostProceso proces = (IPostProceso) objecte;
//...Altres comprovacions...
try {
        proces.ejecutar(listado, peticion, listaParametros,
            parametrosPeticion);
}catch (Exception e) {
        enviarEmailError(e);
}
```

Procediments emmagatzemats a la base de dades

Un dels principals requeriments del nostre client és que es poguessin executar procediments contra diferents bases de dades. Per tal de fer-ho utilitzarem JDBC (Java Database Connectivity). JDBC és una api que permet executar operacions sobre base de dades. Com està implementat en Java, el sistema operatiu o la base de dades que hi hagi al darrere no hi tenen cap influència, hi tindrà influència la versió de Java que s'utilitzi tot i que en el nostre cas no és gaire rellevant. Un dels punts importants que caldrà tenir en compte és que el driver jdbc el proporciona cada fabricant de base de dades i per tant, caldrà tenir instal·lada al servidor la llibreria corresponent amb el driver jdbc de la base de dades contra la qual es volen cridar els procediments. La connexió entre aplicacions Java i bases de dades es realitza mitjançant el paquet java.sql.*. A l'entitat 'Origenes de datos', hi tenim totes les dades que ens calen per tal de poder cridar el mètode. Un dels camps més importants és la cadena de connexió. La cadena de connexió és una manera estàndard d'indicar les dades del servidor de base de dades al què cal connectar. La seva notació és la següent:

jdbc:[subProtocol]://[serverName[instanceName][:portNumber]]

Amb les dades de la cadena de connexió, l'usuari i el password de la base de dades podem establir una connexió amb la base de dades mitjançant la classe DriverManager. A continuació ja només faltarà preparar els paràmetres i fer la crida al procediment. Una particularitat del sistema és que, com sempre són procediments i no funcions el que es crida, no s'obté mai valor de retorn i no se sap si el procediment s'ha executat correctament. Per tal de tenir control, en tots els llistats sempre hi haurà un paràmetre anomenat OUT, que es registrarà de sortida i on s'emmagatzemarà el resultat de l'execució.

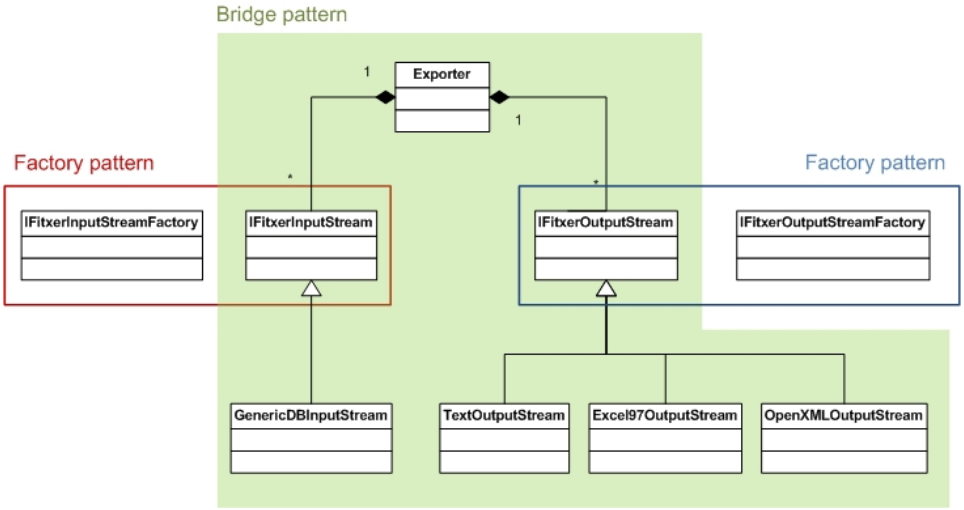

Figura 3.8: Patrons de disseny utilitzats a l'exportació dels llistats

3.5 Exportació dels llistats

Una vegada s'han executat els procediments, cada procediment crea a la base de dades una taula amb les dades resultat. A continuació s'ha de accedir a aquesta taula i transformar les dades al format especificat. D'aquest procés se'n distingeixen dos tasques: com s'extreuen les dades de la taula mitjançant una cache de files i com s'exporten les dades.

Tant en la recolecció com a l'exportació s'utilitzen els patrons de disseny software Bridge [2] i Factory [3] El patró Factory ens permet eliminar la dependència entre el codi i les classes de tal manera que es pugui treballar mitjançant les seves interfícies. El patró Bridge té la finalitat de separar el contingut de la implementació.

Amb l'utilització d'aquests patrons s'aconsegueix que el codi sigui més entenedor, més eficient i més escalable. En els punts següents s'explica els detalls de la implementació de cada patró.

3.5.1 Recolecció de dades

La recolecció de les dades es du a terme mitjançant una cache de 500 files. L'ús d'una cache es una bona pràctica per evitar sobrecarregar la xarxa i per augmentar la velocitat del procés.

En la recolecció de dades, l'utilització del patró bridge té com a objectiu unificar les crides a les classes importadores de dades. D'aquesta manera es poden afegir noves fonts de dades sense que el codi es vegi afectat.

Per dur a terme l'objectiu s'ha dissenyat una interfície que hauran d'heretar totes les classes importadores.

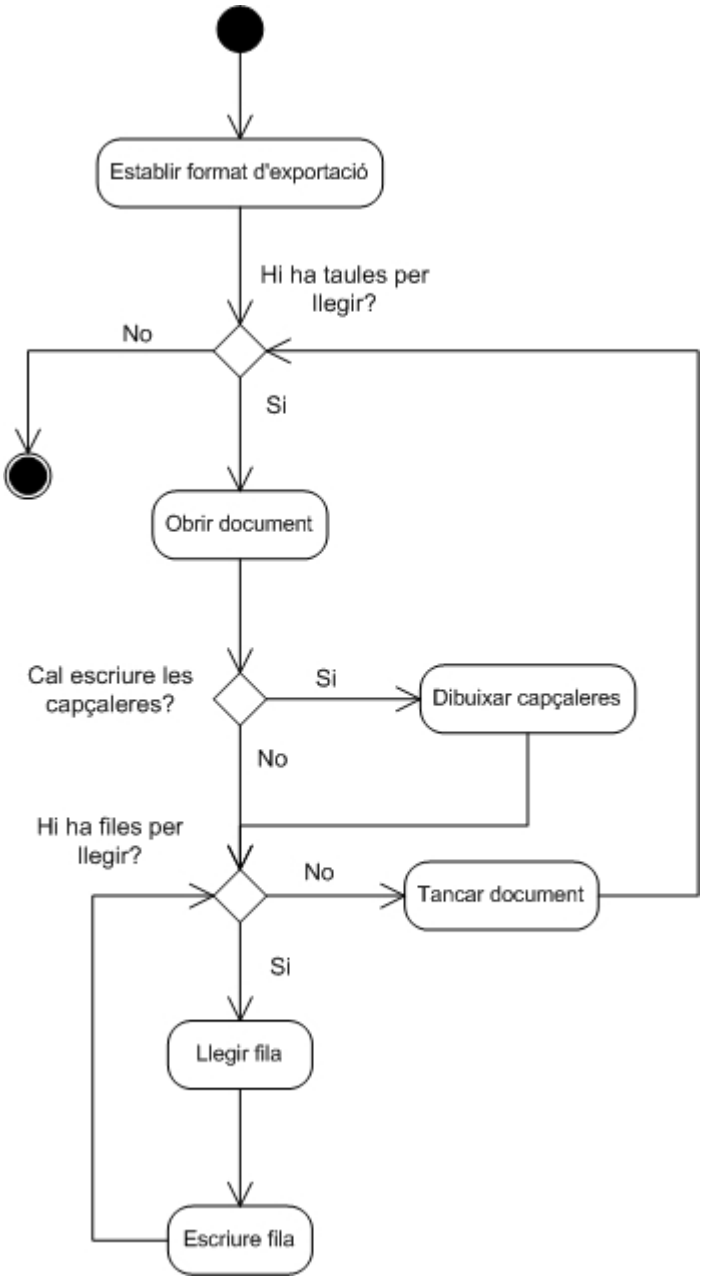

Figura 3.9: Detall d'‘Exportar resultats’ (figura 3.7)

```
public interface IFitxerInputStream {
  public void setDBConnection(Connection connection);
  public LinkedHashMap<Long, Object> read();
  public long getNumberRecords();
  public void setTableName(String tableName);
  public LinkedHashMap<Long, String> getColumnNames();
  public ResultSetMetaData getRsMetaData();
}
```

A continuació explicarem la finalitat de cada mètode i ho exemplificarem amb una connexió a una base de dades genèrica. D'aquesta manera, es pot veure la facilitat en que es poden crear nous origens de dades.

- SetDBConnection, estableix la connexió a la base de dades per utilitzar.

```
public void setDBConnection(Connection connection) {
  conn = connection;
}
```

- Read, llegeix de la cache la següent fila que cal servir. En cas que estigui buida, va a buscar el origen de dades les següents files.

```
public LinkedHashMap<Long, Object> read() {
// La caché està buida, llegim nous elements.
  if ((cacheFilesLlegides==null) || (cacheFilesLlegides
      .size()==0)) {
    setCacheFromQueryWindow();
    //Si continua buida vol dir que no hi res més a
       llegir
    if ((cacheFilesLlegides==null) || (
       cacheFilesLlegides.size()==0))
      return null;
  }
  return cacheFilesLlegides.poll();
}
```

- GetNumberRecords, retorna el nombre de files de l'origen de dades.

```
public long getNumberRecords() {
  // si és null retornem 0
  if ((conn == null) || (tableName == null)) return 0;

  try {
    String sql = "select count(*) from "+tableName;
    Statement st = conn.createStatement();
```

```
8
9       if (st.execute(sql)) {
10        ResultSet rs = st.getResultSet();
11        rs.next();
12        Long resultats = rs.getLong(1);
13
14        if (resultats != null) return resultats.longValue
              ();
15        else return 0;
16      }
17    } catch (Exception e) {
18    // Si es produeix un error es retorna 0
19    return 0;
20    }
21    return 0;
22  }
```

- SetTableName, estableix el nom de la taula que serveix d'origen de dades.

```
1   public void setTableName(String tableName) {
2     columnNames=null;
3     cacheFilesLlegides=null;
4     this.tableName=tableName;
5   }
```

- GetColumnNames, retorna el nom de les columnes de la taula.

```
1   public LinkedHashMap<Long, String> getColumnNames() {
2     return columnNames;
3   }
```

- GetRsMetaData, retorna la metadata associada a la taula.

```
1   public ResultSetMetaData getRsMetaData() {
2     return rsMetaData;
3   }
```

A banda de la utilització del patró Bridge, per tal que el procés d'importació no pateixi modificacions, cal utilitzar també el patró Factory.

En aquest cas la implementació del patró factory és la següent:

```
1   public class InputStreamFactory {
2     public static IFitxerInputStream getInstance(String
          tipusImportacio) {
```

```
3       if (tipusImportacio.equals(TipusImportacioEnum.
            GENERICDB))
4         return new GenericDBInputStream();
5
6       throw new RuntimeException("Format d'importació no
            vàlid: "+ tipusImportacio);
7     }
8   }
```

D'aquesta manera desde el procés d'exportació fent la crida corresponent obtenim l'objecte de tipus IfitxerInputStream i podem treballar-hi independentment de la implementació.

```
1   IFitxerInputStream isFitxer = null;
2   isFitxer = InputStreamFactory.getInstance(formatoEntrada);
```

3.5.2 Exportació de dades

En l'exportació de les dades, la utilització del patró bridge té com a objectiu unificar les crides de les classes exportadores, de tal manera que el procés d'exportació sempre seguirà el mateix flux d'execució. Això ens permetrà afegir, modificar o eliminar diferents formats d'exportació sense que la resta de codi es vegi afectada. Addicionalment, la utilització del patró bridge permet treballar simultàniament durant el desenvolupament les implementacions del procés d'exportació i dels seus formats.

Per dur a terme l'objectiu s'ha dissenyat una interfície que hauran d'heretar totes les classes exportadores.

```
1   public interface IFitxerOutputStream
2   {
3     void createFitxer(String nombreDirectorio, String
          nombreFichero);
4     void setInputStream(IFitxerInputStream inputStream);
5     long read();
6     void closeFitxer();
7     void setGenCabeceras(boolean genCabeceras);
8   }
```

A continuació explicarem la finalitat de cada mètode i ho exemplificarem en el cas més senzill d'exportació, els documents de tipus TXT. D'aquesta manera, es pot veure la facilitat en que es poden crear noves classes exportadores de llistats.

- createFitxer, crear les carpetes indicades si no existeixen i inicialitzar el document perquè estigui preparat per a l'escriptura.

```
public void createFitxer(String nombreDirectorio,
    String nombreFichero) {
  try {
    File arxiu = new File(nombreDirectorio); // Folder
        path
    //Creem directoris si no existeixen
    if(!arxiu.exists())
      arxiu.mkdirs();

      bufferedOut = new BufferedOutputStream(new
          FileOutputStream(new File(nombreFichero)),
          BUF_SIZE);

    } catch (IOException e) {
      log.error("Error creant fitxer: ",e);
    }
}
```

- SetInputStream, guarda en una variable membre de la classe de tipus IFitxerInputStream l'objecte importador d'on s'aniran llegint les dades.

```
public void setInputStream(IFitxerInputStream
    inputStream) {
  this.inputStream = inputStream;
}
```

- Read, procés de lectura de les dades desde l'objecte IfitxerInputStream.

```
public long read() {
  //Comprovem els objectes de ES
  if (bufferedOut==null || inputStream == null)
    return 0;

  LinkedHashMap<Long, Object> fila = null;

  try {
    //Agafem la fila
    fila=inputStream.read();

    // Mirem si hem de posar el nom de les columnes
    if (!nomColumnes && genCabeceras)
    escriuColumnes();

    if (fila!=null) {
      escriuFila(fila);
      return 1;
    } else
```

```
20          return 0;
21      } catch (IOException e) {
22        return 0;
23      }
24    }
```

- CloseFitxer, fa totes les operacions necessàries per tancar el document perquè estigui preparat per ser enviat a l'usuari.

```
1   public void closeFitxer() {
2     try {
3       if (bufferedOut!=null) {
4         bufferedOut.flush();
5         bufferedOut.close();
6       }
7     } catch (IOException e) {
8       log.error("Error tancant fitxer: ",e);
9     }
10  }
```

- SetCabeceras, guarda en una variable membre de la classe de tipus boleà si cal generar les capçaleres del llistat.

```
1   public void setGenCabeceras(boolean genCabeceras) {
2     this.genCabeceras = genCabeceras;
3   }
```

A banda de la utilització del patró Bridge, per tal que el procés d'exportació no pateixi modificacions cal utilitzar també el patró Factory.

En aquest cas la implementació del patró factory és la següent:

```
1   public class OutputStreamFactory {
2     public static IFitxerOutputStream getInstance(String
          tipusExportacio) {
3       if (tipusExportacio.equals(FormatFitxerLlistatEnum.
            TEXTE))
4         return new TexteOutputStream();
5
6       if (tipusExportacio.equals(FormatFitxerLlistatEnum.
            EXCEL97))
7         return new Excel97OutputStream();
8
9       if (tipusExportacio.equals(FormatFitxerLlistatEnum.
            OPENXML))
10        return new OpenXMLOutputStream();
```

```
11
12      throw new RuntimeException("Format d'exportació no
            vàlid: "+ tipusExportacio);
13    }
14  }
```

D'aquesta manera desde el procés d'exportació fent la crida corresponent obtenim l'objecte de tipus IfitxerOutputStream i podem treballar-hi independentment de la implementació.

```
1  IFitxerOutputStream osFitxer = null;
2  osFitxer = OutputStreamFactory.getInstance(formatoSalida);
```

Exportació en format Excel

Per dur a terme l'exportació en format Excel s'han utilitzat les llibreries POI [14] . Les llibreries POI són un projecte de la Apache Software Foundation que donen suport al tractament de documents del paquet d'aplicacions Microsoft Office. En la versió utilitzada per a aquest projecte (juliol 2009) han incorporat la funcionalitat de poder crear documents en format OpenXML, que és l'utilitzat per a les noves versions de Microsoft Office (a partir del 2003). S'han creat les classes Excel97OutputStream i OpenXMLOutputStream per tal que l'aplicatiu tingui la possibilitat d'exportar en els dos formats.

3.6 Generació de documentació

Una vegada s'ha generat el llistat corresponent es comprova si cal generar la documenació associada a aquest llistat. Aquesta documentació no es genera en el nostre aplicatiu sinó que cal fer la crida a un webservice de l'aplicació SGD (Servicio Generador de Documentación). A grans trets, el que realitza l'aplicació SGD és executar una sèrie de reports creats amb la tecnologia Jasper Reports [15]. Aquests reports utilitzen com a font d'entrada els llistats generats per LWB i extreuen la documentació associada.

La comunicació amb el servei SGD es duu a terme mitjançant serveis web (webservices). En el nostre cas, ens caldran dos serveis web: un per comunicarnos amb SGD i fer la petició de documentació i l'altre perquè SGD ens contesti un cop ha acabat. Aquest procés és totalment asíncron i, per tant, no és blocant per a la nostra aplicació.

3.6.1 Generar petició de documentació

Per generar la petició de documentació ens cal comunicar-nos amb el webservice de SGD. Aquest webservice està definit en el WSDL [5] de l'aplicatiu SGD i s'hi poden trobar totes les dades necessàries per realitzar la connexió. També caldrà

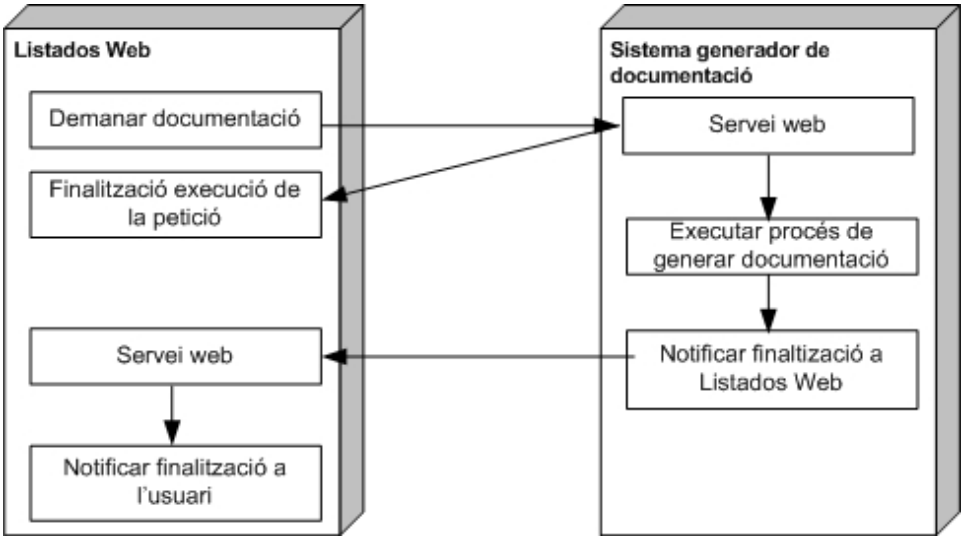

Figura 3.10: Comunicació entre LWB i el servei generador de documentació

que a la petició de documentació s'indiqui la URL del webservice de LWB de tal manera que es pugui rebre la documentació una vegada finalitzada la generació. La implementació de la comunicació amb el webservice no és gens costosa ja que SGD proporciona una classe anomenada WSProxyFactory que simplifica el procediment. Un exemple de crida al webservice seria de la següent manera:

```
1  try {
2    DatosWebService datosServicio = new DatosWebService();
3    datosServicio.setServiceUri(serviceURI);
4    datosServicio.setServiceSpace(serviceSpace);
5    datosServicio.setServiceName(serviceName);
6
7    DatosCallback datosCallback = new DatosCallback();
8    datosCallback.setServiceUri(callbackURI);
9    datosCallback.setServiceSpace(callbackSpace);
10   datosCallback.setServiceName(callbackName);
11
12   DateFormat dateFormat = new SimpleDateFormat(es.sgch.lwb.
         data.enums.Constants.FORMAT_DATES_BD);
13   String sData = dateFormat.format(new Date());
14
15   WSProxyFactory srvFactory=WSProxyFactory.getInstance();
16   IGeneradorDocumentos srv = (IGeneradorDocumentos)
         srvFactory.getService(datosServicio,
         IGeneradorDocumentos.class);
17
18   //Executem le peticio de documentacio
19   ResultadoReport resultadoReport = srv.
```

```
      generaReportDiferido(
20      es.sgch.lwb.data.enums.Constants.
           WS_LWB_CODIGO_APLICACION,
21      userName,
22      datosReport,
23      sData,
24      datosCallback);
25
26      //Mirem el resultat
27      if (resultadoReport.getResultado() == 0 &&
           resultadoReport.getIdPeticion() != null) {
28        return resultadoReport.getIdPeticion();
29      }
30      // Si hi ha error
31      return Constants.NOK;
32  } catch (MalformedURLException e) {
33    log.error("MalformedURLException ",e);
34    return Constants.NOK +" " + e.getMessage();
35  } catch (Exception e) {
36    log.error("Exception ",e);
37    return Constants.NOK +" " + e.getMessage();
38  }
```

3.6.2 Recepció documentació

La recepció de la informació passa per crear un servei web. El sistema generador de documentació ja ens proporciona una interfície anomenada ISGDCallbackService amb els mètodes que cal que implementi el nostre servei web per tal dque la comunicació entre les dues parts funcioni correctament. D'aquesta manera només cal definir la interfície proporcionada com a definició del nostre servei web de tal manera que el mateix servidor d'aplicacions ja s'encarrega de configurar els tipus dels paràmetres i els valors de retorn. La implementació d'aquest servei web és molt senzill usant anotacions i es pot veure a continuació:

```
1   @Stateless
2   @WebService(endpointInterface = "es.sgch.sgd.callback.
       ISGDCallbackService")
3   @Remote(ISGDCallbackService.class)
4   @BindingType(value=javax.xml.ws.soap.SOAPBinding.
       SOAP12HTTP_MTOM_BINDING)
5   public class RecepcionDocumentacionWSService implements
       ISGDCallbackService, Serializable {
6
7     private static final long serialVersionUID = 1L;
8
9     public void generadorDocumentosCallback(
10      String idPeticion,
```

```
11      ParametroReport[] parametros,
12      String mensajeResultado,
13      int resultado,
14      String rutaReportGenerado,
15      byte[] documento)
16    {
17      Logger log = Logger.getLogger(this.getClass());
18
19      ResultadoReport resultadoReport = new ResultadoReport()
            ;
20      resultadoReport.setIdPeticion(idPeticion);
21      resultadoReport.setMensajeResultado(mensajeResultado);
22      resultadoReport.setResultado(resultado);
23      resultadoReport.setRutaReportGenerado(
            rutaReportGenerado);
24
25      RecepcionDocumentacionService
            recepcionDocumentacionService =
26          new RecepcionDocumentacionService();
27      recepcionDocumentacionService.procesaNotificacio(
            resultadoReport);
28    }
29  }
```

Finalment, ja només resta implementar la funcionalitat que es desitja a dins de la funció. Així una vegada el servei de gestió documental s'encarregui de contactar amb el nostre servei web caldrà processar el resultat, informar a l'usuari que ja s'ha rebut la documentació i donar per finalitzada la petició.

3.7 Seguretat

Per tal d'integrar l'arquitectura d'Access Manager amb la nostra aplicació cal configurar diversos paràmetres, tant al servidor d'Access Manager, com a l'agent instal·lat al servidor d'aplicacions com a la mateixa aplicació. A continuació, es detalla la configuració que s'ha dut a terme en cadascun d'ells. En aquestes configuracions es parteix des d'un escenari ideal, és a dir, que la integració entre l'Access Manager, l'agent i una aplicació d'exemple ha estat provada i funciona correctament.

Configuració a l'aplicació

Cal activar la seguretat a l'aplicació. Per fer-ho definirem un filtre d'Access Manager que s'encarregarà d'interceptar les peticions a recursos protegits i autenticar-los contra Access Manager usant l'agent. També cal definir el mètode d'autenticació via formulari. D'aquesta manera quan hi ha una petició d'auntenticació l'agent tindrà configurats aquests mateixos paràmetres i podrà redirigir-se a la pantalla

d'autenticació d'Access Manager. Aquesta pantalla és única i està unificada per a totes les aplicacions que s'autentiquen contra el mateix Access Manager. De tota manera és totalment personalitzable mitjançant fulls d'estil. Aquestes configuracions s'han de realitzar al descriptor web.xml de l'aplicació de la manera següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="2.5" xsi:schemaLocation="http://java.sun.com/
    xml/ns/javaee   http://java.sun.com/xml/ns/javaee/web-
    app_2_5.xsd">

  <display-name>LWB - Listados web</display-name>
  <!-- Filtre AM -->
  <filter>
    <filter-name>Agent</filter-name>
    <filter-class> com.sun.identity.agents.filter.
       AmAgentFilter </filter-class>
  </filter>
  <filter-mapping>
    <filter-name>Agent</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
  </filter-mapping>

<!-- Login form -->
  <login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
      <form-login-page>/html/login.html</form-login-page>
      <form-error-page>/html/accessdenied.html</form-error-
         page>
    </form-login-config>
  </login-config>

  <!-- Altres configuracions -->

</web-app>
```

Configuració a l'agent

La configuració a l'agent és un punt important. L'agent serà l'encarregat de dur a terme la comunicació entre la nostra aplicació i el servidor d'Access Manager de manera que puguem dotar l'aplicació d'una arquitectura 'single sign on'. La

configuració de l'agent és una tasca difícil i complicada que no es tractarà en aquest projecte, ja que els encarregats de dur-la a terme són el Departament de Sistemes. Pel que a nosaltres ens interessa, que és com es configura l'aplicació a dins l'agent, l'única configuració que cal realitzar és indicar la configuració les planes de login que s'han indicat al descriptor web.xml de l'apartat anterior. A més, opcionalment es poden indicar recursos no protegits (en anglès 'not enforced') que seràn útils per no protegir recursos com poden ser documents JPEG o CSS. Les configuracions són les següents:

```
1 com.sun.identity.agents.config.login.form[0]=/LWB/html/
      login.html
2
3 com.sun.identity.agents.config.login.error.uri[LWB]=/LWB/
      html/error.html
4
5 com.sun.identity.agents.config.notenforced.uri[0] = /LWB/
      images/*
6 com.sun.identity.agents.config.notenforced.uri[1] = /LWB/
      public/*.html
7 com.sun.identity.agents.config.notenforced.uri[2] = /LWB/
      style/*.css
```

Configuració a l'Access Manager

La configuració d'un Access Manager funcional que s'autentica contra diferents dominis i arquitectures també és força complicada. El que nosaltres ens interessa, és com configurar l'aplicació al servidor Access Manager. En aquest cas, s'encarregarà de definir les polítiques que s'han de seguir a l'hora de protegir recursos. Aquests recursos es protegeixen mitjançant URL, mètodes de passar informació (get i post majoritariament) i decisions que ha de pendre el servidor (deny o allow). Per qüestions de confidencialitat, no es pot mostrar cap detall de la configuració del servidor Access Manager.

3.8 Arquitectura de servidors

Les aplicacions client i servidor han de ser desplegades en servidors diferents. L'elecció d'aquests servidors és molt important, ja que no tots els servidors són compatibles amb les tecnologies utiltizades i cal buscar-ne que s'ajustin a les necessitats de les nostres aplicacions. A més, també es justificarà l'elecció de la base de dades de l'aplicatiu LWB.

JBoss Application Server 5.1

Jboss Application Server [16] és un servidor d'aplicacions de codi obert implementat en Java. Jboss implementa tot el paquet de serveis J2EE i és un dels servidors punters en la implementació de noves tecnologies.

S'ha decidit utilitzar Jboss 5.1 GA per al desplegament de l'aplicació servidor. La raó principal és per la posició de privilegi de Jboss dins del mercat actual, és un dels servidors de referència. Això fa que totes les tecnologies que necessitem utilitzar estiguin soportades per aquest servidor. A més ofereix una gran flexibilitat a l'hora de treballar-hi amb ell. Un altre factor a tenir en compte també és que els desenvolupadors de l'empresa hi estem familiaritzats i per tant no hi ha cap tipus d'aprenentatge. L'únic inconvenient d'utilitzar Jboss és que és massa pioner amb noves tecnologies i, a vegades, algunes versions no són gaire estables, és per això que s'ha decidit escollir una versió GA (General Availability).

Websphere Application Server 6.1

Websphere Application Server [17] és un servidor d'aplicacions desenvolupat per IBM i que forma part de la família de productes Websphere. La principal virtut d'aquest servidor és l'estabilitat que proporciona. Aquesta estabilitat ve donada com a servidor ja que utilitza un jdk propi i que només es treuen al mercat versions molt ben provades. A més hi ha la possibilitat de contractar suport tècnic 24x7 i a darrere hi ha una companyia de renom internacional com IBM.

S'ha utilitzat Websphere per al desplegament de l'aplicació client. La decisió d'utilitzar Websphere no ha estat nostra, sinó que ha estat presa pel client. Això ha fet que hàgim hagut d'aprendre a utilitzar la seva consola gràfica i a configurar-lo per poder utilitzar les tecnologies que ens feien falta.

Oracle 10

Oracle 10 és un sistema de gestió de bases de dades relacionals desenvolupat per Oracle Corporation. Es considera que Oracle és un dels sistemes de bases de dades més complets i cal destacar: el suport a transaccions, l'estabilitat, l'escalabilitat i el suport multiplataforma.

S'ha utilitzat Oracle com a base de dades de l'aplicatiu LWB. La decisió d'utilitzar Oracle no ha estat nostra però tampoc ha estat cap inconvenient ja que és la base de dades amb què estem habituats a treballar. A més proporciona una eina gràfica de treball com Oracle SQL Developer que ens ha estat molt útil.

3.9 Entorns

Per l'aplicatiu de LWB s'han instal·lat diversos entorns que proporcionen estabilitat i coherència, per tal que els individus que intervenen en el projecte puguin realitzar cadascú les seves tasques. Un entorn estarà compost per tots els servidors

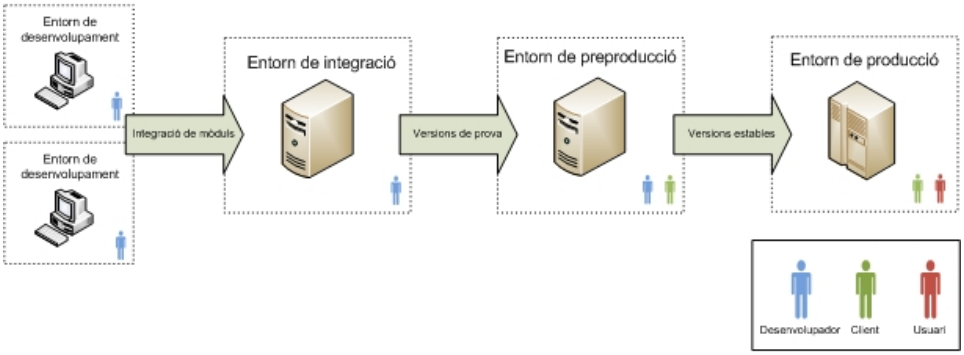

Figura 3.11: Entorns de treball de l'aplicació

explicats en el punt 3.8. Concretament s'han configurat els entorns que es detallen a continuació:

- Entorn de desenvolupament, entorn local del desenvolupador on es van fent desplegaments durant el desenvolupament de l'aplicació per fer proves funcionals. No és un entorn estable i, per tant, mai no es pren com a referència.

- Entorn d'integració, entorn local comú per a tots els desenvolupadors on es van fent els desplegaments de les versions definitives que es pujaran després als entorns del client. Sovint és utilitzat per fer proves d'integració de diversos mòduls de l'aplicació fetes per diferents desenvolupadors, però ha de ser una reproducció exacta dels entorns del client. D'aquesta manera, quan es rebi una incidència, es reproduirà en aquest entorn i es buscarà la solució. Qualsevol desplegament que es faci als entorns del client ha d'haver estat provat i validat en aquest entorn primer.

- Entorn de preproducció, entorn proporcionat pel client on es fan els desplegaments de les versions preliminars de l'aplicatiu. També serveix per realitzar proves que, per qüestions de l'arquitectura del client, no es poden reproduir en els entorns locals. En qualsevol cas, ha de ser un entorn el màxim estable possible ja que és aquí on el client farà les validacions pertinents de l'aplicatiu i en donarà el vistiplau per passar-ho a producció.

- Entorn de producció, entorn on els usuaris treballen amb l'aplicatiu. Qualsevol versió que està desplegada en aquest entorn és estable i només es substituirà per versions estables provades abans en els entorns anteriors. Aquest entorn ha d'estar actiu i funcional sempre. És per això que, en aquest entorn es guarden còpies dels paquets de les versions anteriors de l'aplicatiu per tal que si hi hagués qualsevol incidència, es pogués tornar a versions estables.

Capítol 4

Resultats

Una vegada s'ha finalitzat el desenvolupament de l'aplicació, el client ha hagut de validar el funcionament correcte. En aquest capítol es detallen com s'han encarat les proves a realitzar i quins han estat els aspectes més importants a tenir en compte. Una vegada finalitzades les proves i quan l'aplicació ha entrat en producció, s'han obtingut una sèrie d'estadístiques que han estat analitzades. Finalment, es valora quin ha estat el impacte de l'aplicació una vegada aquesta ha estat en marxa durant dos mesos (Octubre i Novembre).

4.1 Validació client

Durant s'han anat desenvolupant els diferents mòduls el client ha hagut de validar-los amb els jocs de proves subministrats. S'han creat tres jocs de proves: un destinat a validar el funcionament de l'aplicació client, un altre per a l'aplicació servidor i un altre per a la generació de documentació. Tal i com està indicat al capitol 3.9, les validacions per part del client sempre s'han realitzat en l'entorn de preproducció.

La validació de l'aplicació client s'ha centrat bàsicament en proves a la interfície gràfica amb què treballarà l'usuari final. Bàsicament s'han tingut en compte aquests aspectes:

- Autenticació de l'usuari i accés a recursos segons les aplicacions definides.
- Visibilitat de la taula de peticions i de les diferents opcions de filtratge.
- Control de fluxos d'execució correctes, per exemple que el procés de crear una petició segueix els passos definits.
- Validacions de camps obligatòris i formats d'aquests en els formularis de l'aplicació.
- Comprovacions sobre si s'emmagatzemen correctament les dades introduïdes pel client en la base de dades. Aquest punt és molt sensible ja que, després,

l'aplicació servidor ha d'interpretar les dades de la base de dades i, si no són correctes, la funcionalitat de l'aplicació serà completament errònia.

- Aspectes en general sobre estètica i fulls d'estils que ha de seguir la interfície gràfica.

Aquestes validacions es duen a terme mitjançant un navegador web connectant-se al servidor on està desplegada l'aplicació client.

La validació de l'aplicació servidor s'ha centrat en la resposta del servidor segons les dades de què es disposava a la base de dades. Els aspectes que s'han tingut en compte han estat:

- Creació de peticions automàtiques segons els llistats configurats.
- Gestió de les peticions: quina és la següent petició a executar segons la seva prioritat i la prioritat del llistat.
- Gestió de les cues.
- Flux d'execució de les peticions i crida dels procediments.
- Mesures de rendiment de la cache de files.
- Exportació en diferents formats i l'emmagatzematge d'aquests en les carpetes d'usuari pertinents.
- Recepció de correus electrònics segons la casuística que s'ha donat en la prova.

La validació del servei de generació documental s'ha tractat com un joc de proves ja que comporta una comunicació amb un altre aplicatiu i per tant cal coordinar bé les proves. En aquest joc de proves es poden distingir dues etapes: una primera on el servei SGD estava en estat embrionari i només es disposava d'un servei dummy que permetia comprovar si la comunicació era correcta, i una segona on el servei ja estava operatiu i es podien validar els resultats de la generació documental. Els aspectes que s'han tingut en compte han estat:

- La comunicació amb el servei web de SGD i la validació dels paràmetres de la crida.
- La comuniació de SGD cap el servei web de LWB i el tractament de la informació rebuda.
- Els diferents formats en què es pot generar la documentació i validació dels resultats obtinguts.

	Mínim	*Mitjana*	*Màxim*
Atenció a la petició	0	21	45
Temps d'execució	0	3	43

Figura 4.1: Estadístiques temporals de LWB

4.2 Estadístiques i impacte

L'objectiu d'aquest punt és comentar l'impacte que ha tingut l'aplicació des que ha estat posada en marxa. L'impacte s'avaluarà en tres sentits:

- Temps. Suposa un estalvi de temps respecte l'aplicatiu anterior? És eficient l'aplicació?
- Econòmic. S'ha produït un estalvi de diners?
- Emocional. Quina és la valoració dels usuaris? I del client?

Temps

Abans de plantejar aquest punt cal tenir en compte la dada següent: en els mesos d'octubre i de novembre de 2009 que l'aplicatiu ha estat en producció, s'han atès 6.626 peticions. Els càlculs de temps obtinguts en minuts es poden veure a la figura 4.1.

Pel que fa al temps d'execució mitjà de les peticions, aquest serà exactament el mateix en la nova aplicació i en el funcionament antic. La diferència la marca l'anteció a la petició: en el funcionament antic hi havia poca planificació i la generació dels llistats dels usuaris estava condicionada pel rendiment del Departament de Sistemes. El Departament de Sistemes tenien l'obligació de dedicar-hi unes hores determinades durant la semana però no tenien cap tipus d'obligació envers als usuaris i això generava temps d'espera molt llargs. Ara en canvi, s'ha aconseguit que l'usuari obtingui el llistat generat en un temps màxim d'aproximadament una hora i mitja.

Econòmic

Segons Infojbos Trends [1] el salari mitjà d'un enginyer informàtic tècnic de sistemes a Espanya al 2008 és de 27.183EUR. Fent números ràpdis això es tradueix a un salari de 2.265EUR. mensuals i de 523EUR. setmanals. Si considerem un treballador a jornada completa que fa 40 hores setmanals, dedica cinc hores a la semana a la generació de llistats, en termes econòmics està dedicant 65EUR. semanals a la generació de llistats. Suposant que un any té 52 semanes l'estalvi del client per treballador del departament de sistemes és de 3.380EUR. anuals. Vist d'aquesta manera, amb l'estalvi produït per vuit treballadors ja es pot contractar una altra persona que s'encarregui de realitzar altres tasques.

[1] http://salarios.infojobs.net/

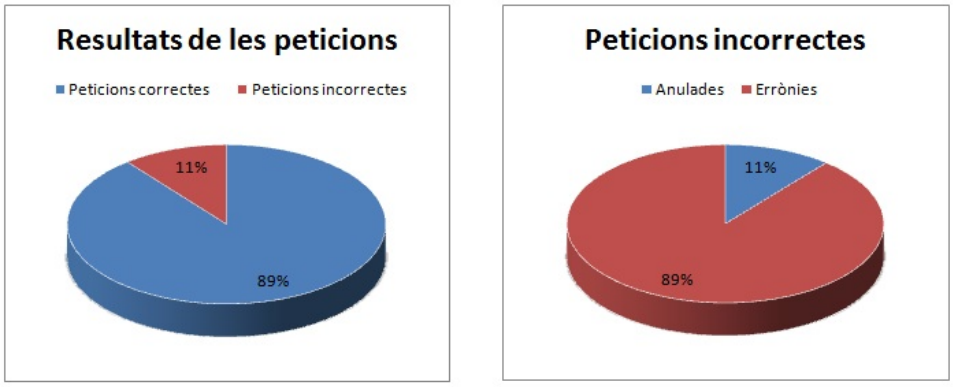

Figura 4.2: Estadístiques globals de LWB

Emocional

La valoració dels usuaris de l'aplicatiu és molt bona. Sobre tot en destaquen que és intuïtiu i permet generar els llistats de forma ràpida. A més en destaquen que la generació de documentació els facilita i els agilitza molt la feina. El client ha aconseguit millorar la productivitat dels seus treballadors i un estalvi considerable de diners. A més s'han alliberat tensions entre departaments ja que sempre sorgien conflictes derivats del temps d'atenció de la generació de llistats. A més, els resultats de les peticions que fan els usuaris tenen un alt percentatge d'acabar correctament i això fa que els usuaris estiguin contents amb l'aplicació. Això es pot comprovar a la figura 4.2.

El client també en destaca el control estadístic, que ara es pot dur a terme de les peticions. Abans no es portava cap tipus de control i, per tant, sempre hi podien haver situacions no desitjades. Per exemple, aquest control estadístic els ha servit per veure quines són les hores del dia i els dia de la semana que els servidors estan més saturats, per tal de planificar una dedicació més alta dels recursos durant els pics de saturació. Les gràfiques relacionades amb aquest fenomen es poden veure en les figures 4.3 i 4.4.

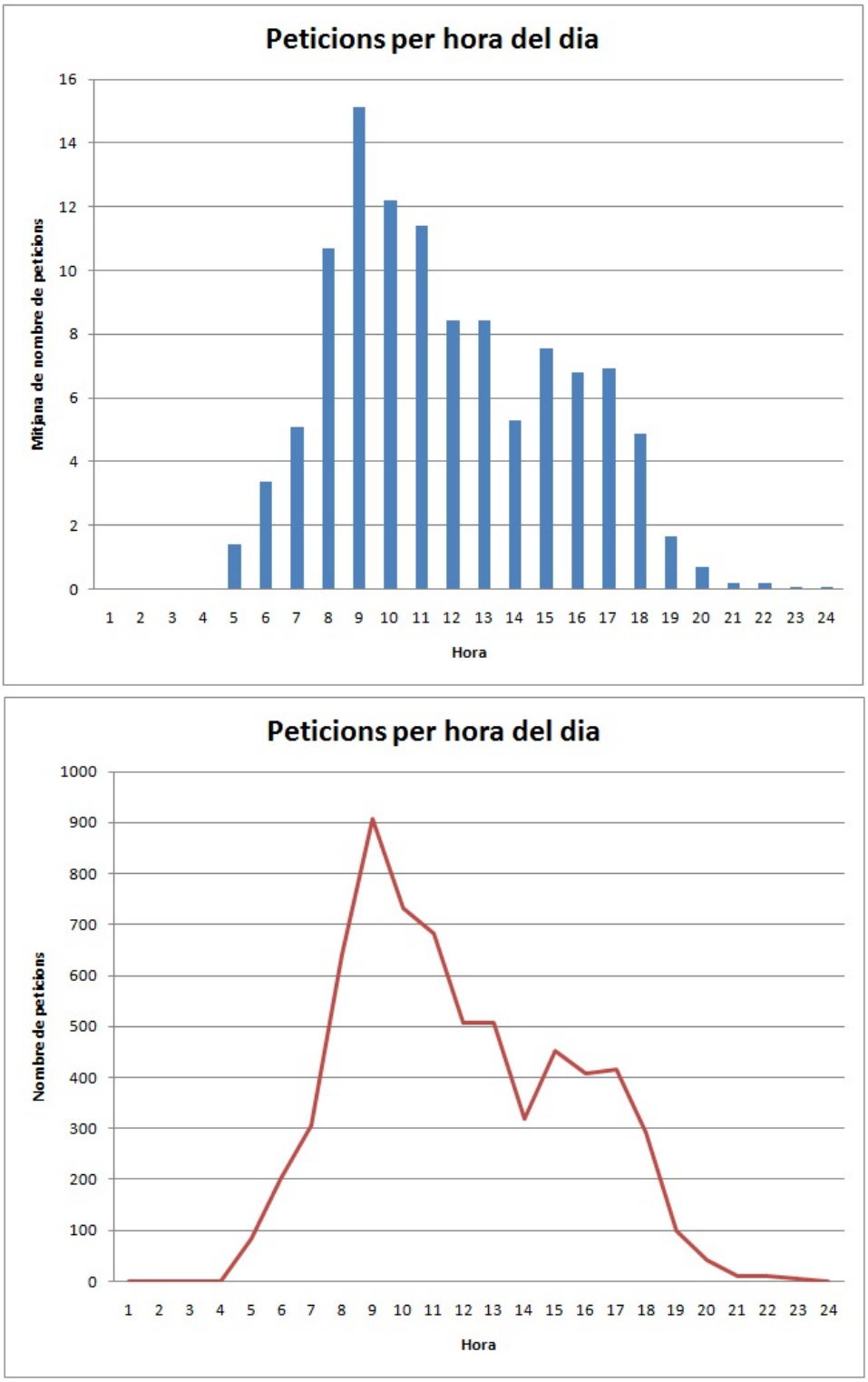

Figura 4.3: Estadístiques totals i mitjana de peticions segons hora del dia

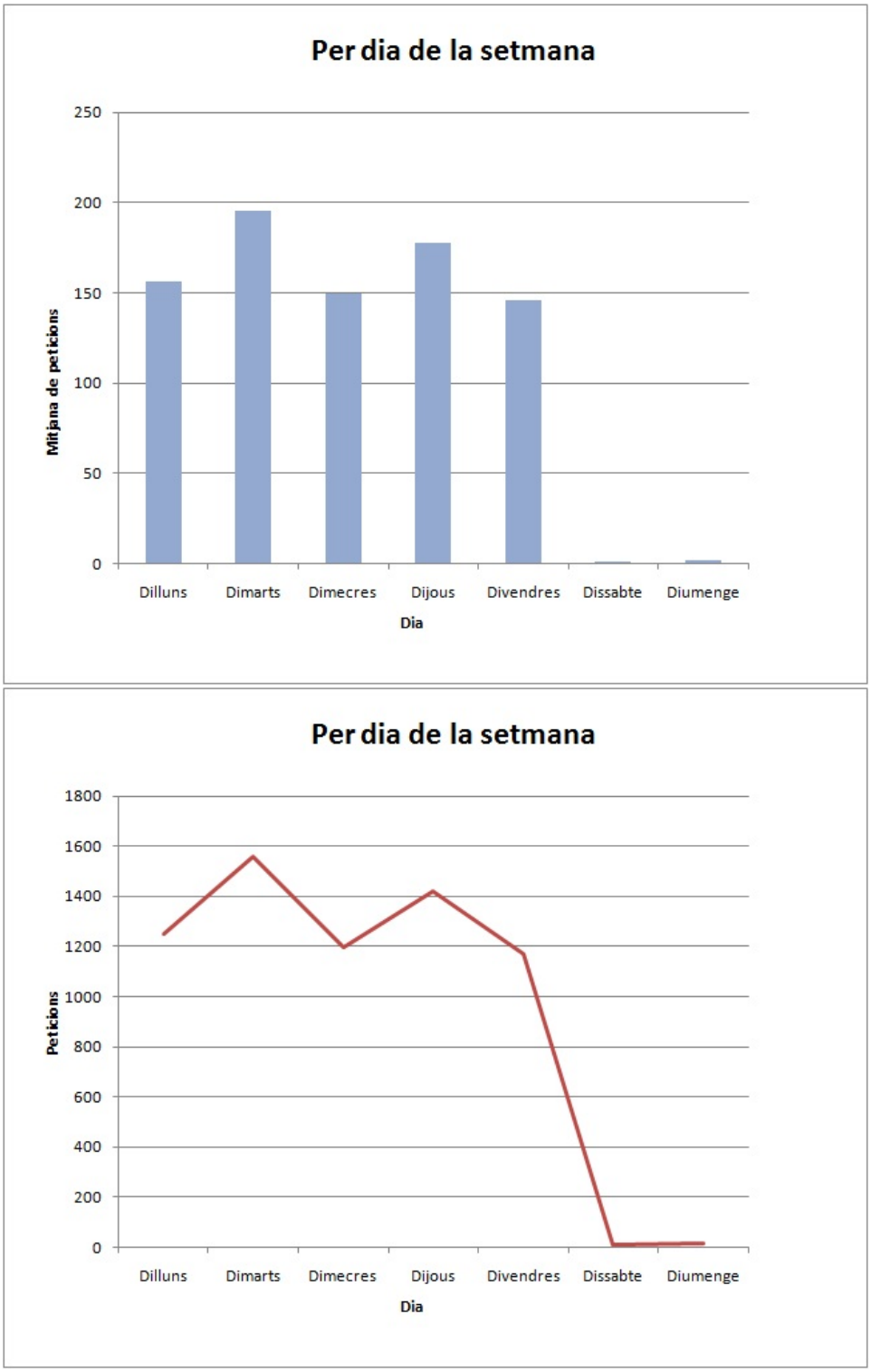

Figura 4.4: Estadístiques totals i mitjana de peticions per dia de la semana

Capítol 5

Conclusions

El client disposa d'un entorn de bases de dades molt heterogeni, d'on s'han d'extreure llistats. El procés d'extracció involucrava anteriorment el Departament de Sistemes. Aquest procés era poc automatitzat: es depenia massa del personal disponible i dels seus coneixaments. A més, les demandes de peticions de llistats havien crescut considerablament, cosa que havia fet que el Departament de Sistemes no donés l'abast.

En conseqüència, calia buscar una solució per tal de dotar els usuaris d'una manera fàcil i entenedora de generar llistats. D'aquesta manera aconseguiriem alliberar el Departament de Sistemes d'aquesta tasca i centrar-se realment en processos més productius. Per tal de resoldre la falta d'automatització en el procés generador de llistats, s'ha creat una aplicació que permet als usuaris generar llistats en diferents formats, independentment de les tecnologies que hi ha darrere. Els objectius plantejats s'han resolt de la manera següent:

- L'execució de procediments de bases de dades de diferents tecnologies es pot dur a terme mitjançant el seu driver i amb la tecnologia JDBC. D'aquesta manera s'obté una solució unificada i parametritzable d'executar els llistats emmagatzemats.

- S'ha dissenyat mitjançant patrons de disseny software una estructura d'exportació de llistats flexible i escalable. S'han implementat tres formats d'exportació: TXT, Excel i OpenXML; i un format d'importació ampliables als que faci falta.

- El seguiment de la documentació proporcionada pel servei generador de documentació ha permès establir connexió i poder fer peticions de generació de documentació. D'aquesta manera, els usuaris poden enviar els llistats i rebre la seva documentació associada mitjançant els serveis web creats.

- Mitjançant l'arquitectura d'autenticació Access Manager s'ha dotat a l'aplicació de seguretat, de tal manera que s'han integrat els mòduls d'autenticació i protecció de recursos amb l'aplicació.

- Creació d'una pàgina web intuïtiva i propera a l'usuari.

Per tant, s'han assolit satisfactòriament els objectius. Cal remarcar el gran impacte que ha tingut l'aplicació sobre la manera de treballar dels usuaris. A banda, ha permès al client dedicar els recursos alliberats a d'altres objectius i, per tant, hi ha hagut un augment de la productivitat.

De cara a un futur, per tal de millorar l'eficiència de l'aplicació, es podrien muntar diversos servidors web en cluster per tal de permetre a l'usuari una navegació més fluida en pics de saturació. A més, es podrien afegir servidors d'aplicacions amb l'aplicació servidora instal·lada en cluster, de tal manera que una o diverses cues s'executin en diferents servidors. D'aquesta manera es podrien aïllar els llistats que tenen més cost i evitar tenir temps d'espera més alts. Una bona manera de facilitar l'administració de l'aplicació podria ser crear una consola d'administració de l'aplicació servidora. Aquesta consola d'administració tindria la funcionalitat de monitoritzar el servidor i permetre administrar les cues i les peticions en curs. A més, també podria servir per visualitzar dades i estadístiques en temps real sobre l'ús de l'aplicatiu.

Bibliografia

[1] Chuck Cavaness. What is quartz. Plana web. http://onjava.com/pub/a/onjava/2005/09/28/what-is-quartz.html.

[2] James W. Cooper. The bridge pattern. Plana web. http://www.patterndepot.com/put/8/bridge.pdf.

[3] James W. Cooper. The factory pattern. Plana web. http://www.patterndepot.com/put/8/factory.pdf.

[4] Keith Donald. Spring web flow 2 released; introduces new faces and javascript modules. Plana web. http://www.springsource.org/webflow/.

[5] Greg Meredith i Sanjiva Weerawarana Erik Christensen, Francisco Curbera. Web services description language (wsdl) 1.1. Plana web. http://www.w3.org/TR/wsdl.

[6] Apache Software Foundation. Poi api documentation. Plana web. http://poi.apache.org/apidocs/index.html.

[7] Max Rydahl Andersen Emmanuel Bernard i Steve Ebersole Gavin King, Christian Bauer. Hibernate - relational persistence for idiomatic java. Plana web. http://docs.jboss.org/hibernate/stable/core/reference/en/html_single/.

[8] Jacob Hookom. Inside facelets part 1: An introduction. Plana web. http://www.jsfcentral.com/listings/FL100;jsessionid=205AC7D1E7B32ED3308DC7C688511F34?link.

[9] Kent Beck i Erich Gamma. Junit cookbook. Plana web. http://junit.sourceforge.net/doc/cookbook/cookbook.htm.

[10] JBoss.org. Richfaces developer guide. Plana web. http://docs.jboss.org/richfaces/latest_3_3_X/en/devguide/html_single/.

[11] Sun Microsystems. The java ee 5 tutorial. Plana web. http://java.sun.com/javaee/5/docs/tutorial/doc/bnaph.html.

[12] Sun Microsystems. Sun java system access manager 7.1 documentation center. Plana web. http://docs.sun.com/app/docs/doc/819-6619/fxjbo?a=view.

[13] Activewidgets SARL. Activewidgets documentation. Plana web. http://www.activewidgets.com/general.intro/.

[14] Tony Sintes. It's poi-fect. Plana web. http://www.javaworld.com/javaworld/javaqa/2002-05/01-qa-0503-excel3.html.

[15] Erik Swenson. Reports made easy with jasperreports. Plana web. http://www.javaworld.com/javaworld/jw-09-2002/jw-0920-opensourceprofile.html.

[16] VVAA. Jboss wiki. Plana web. http://www.jboss.org/.

[17] VVAA. Online information center. Plana web. http://www-01.ibm.com/software/webservers/appserv/was/.

Apèndix A

Exemple del model de dades

AbstractTAplicacionData.java

```
1  public abstract class AbstractTAplicacionData
2          implements java.io.Serializable {
3
4          /** Fields */
5          private String aplCodigo;
6          private String aplNombre;
7          private Set tlistados;
8
9          /** default constructor */
10         public AbstractTAplicacionData() {
11         }
12
13         /** minimal constructor */
14         public AbstractTAplicacionData(String aplCodigo) {
15                 this.aplCodigo = aplCodigo;
16         }
17
18         /** full constructor */
19         public AbstractTAplicacionData(String aplCodigo,
               String aplNombre, Set tlistados)
20                 this.aplCodigo = aplCodigo;
21                 this.aplNombre = aplNombre;
22                 this.tlistados = tlistados;
23         }
24
25         /** getters and setters **/
26         public String getAplCodigo() {
27                 return this.aplCodigo;
28         }
29         public void setAplCodigo(String aplCodigo) {
30                 this.aplCodigo = aplCodigo;
31         }
```

```
32          public String getAplNombre() {
33                  return this.aplNombre;
34          }
35          public void setAplNombre(String aplNombre) {
36                  this.aplNombre = aplNombre;
37          }
38          public Set getTListadoses() {
39                  return this.TListadoses;
40          }
41          public void setTListadoses(Set tlistados) {
42                  this.tlistados = tlistados;
43          }
44  }
```

TAplicacionData.java

```
1   public class TAplicacionData
2           extends AbstractTAplicacionData
3           implements java.io.Serializable {
4
5           /** Fields */
6           private boolean alive;
7
8           /** default constructor */
9           public TAplicacionData() {
10
11          }
12
13          /** minimal constructor */
14          public TAplicacionData(String aplCodigo) {
15                  super(aplCodigo);
16          }
17
18          /** full constructor */
19          public TAplicacionData(String aplCodigo,String
                aplNombre,Set tlistados) {
20                  super(aplCodigo, aplNombre,tlistados);
21          }
22
23          /** getters and setters */
24          public boolean isAlive() {
25                  return this.alive;
26          }
27          public void setAlive(boolean alive) {
28                  this.alive = alive;
29          }
30  }
```

TAplicacionData.hbm.xml

```
<hibernate-mapping>
  <class name="es.sgch.lwb.data.TAplicacionData" table="
     T_APLICACION" schema="lwb">
    <id name="aplCodigo"
        type="java.lang.String">
      <column name="APL_CODIGO"
              length="5" />
      <generator class="assigned" />
    </id>
    <property name="aplNombre"
              type="java.lang.String">
      <column name="APL_NOMBRE"
              length="25" />
    </property>
    <set name="TListadoses" inverse="true">
      <key>
        <column name="LIS_APLICACION"
                length="5" />
      </key>
      <one-to-many class="es.sgch.lwb.data.TListadosData" /
         >
    </set>
  </class>
</hibernate-mapping>
```

TAplicacionDAO.java

```
public List findByProperty(String propertyName, Object
   value) {
        try {
                String queryString =
                        "from TAplicacionData as model
                           where model."
                        + propertyName + "= ?";
                Query queryObject = getSession().
                   createQuery(queryString);
                queryObject.setParameter(0, value);
                return queryObject.list();
        } catch (RuntimeException re) {
                throw re;
        }
}
```