"*eAula*: Espai web d'intercanvi de documents en una xarxa peer-to-peer"

José Cañellas Frau
Enginyeria Tècnica de Telecomunicacions (Telemàtica)

Rubén Mondéjar Andreu

20/01/2009

Índex de continguts

Índex de figures

Índex de pantalles

1 Prefaci

El projecte aquí tractat, "***eAula*: Espai web d'intercanvi de documents en una xarxa peer-to-peer**" és la memòria final del que s'ha anat desenvolupant al llarg d'aquest semestre, a l'àrea del TFC d'Aplicacions i sistemes distribuïts de la UOC. Com indica el seu nom *eAula* és una aplicació, concretament una aplicació web destinada a fer fàcil l'intercanvi de documents en un entorn col·laborador. El punt inicial que es pretén a *eAula* és la creació d'un entorn de treball semblant al clàssic BSCW (*Basic Support for Cooperative Work*), un sistema de suport al treball en grup que proporciona facilitats per a la cooperació entre persones allunyades físicament. Però hi ha una diferència estructural que fa diferir BSCW i *eAula* i és l'ús d'aquest darrer de la xarxa peer-to-peer en lloc de sistemes centralitzats. Aquest darrer punt, és a dir la creació d'un espaï col·laboratiu totalment distribuït, fa de *eAula* un projecte molt engrescador i amb moltes possibilitats.

2 Introducció

2.1 Resum introductori.

En els darrers anys s'ha pogut veure i constatar un increment molt important en l'ús de les noves tecnologies de la informació i comunicació (TIC) dins les aules dels centres públics, tan com element engrescador com per motius purament didàctics (nous enfocament pedagògics) . Característiques com la interacció, el dinamisme, poden suplir carències dels sistemes tradicionals d'ensenyament focalitzat més en l'ús dels llibres de text. L'aprenentatge col·laborador mitjançant ordinadors és un dels recursos més prometedors per a la millora de l'ensenyament, ja que a part d'introduir la tecnologia de la informació a les aules ho fa dins un context de cooperació i col·laboració, entès aquest aprenentatge com una tècnica pedagògica en la qual els estudiants treballen junts per aconseguir uns objectius comuns i cada individu assoleix els objectius marcats si i només si el resta de grup ho aconsegueix.

Els espais de treball compartit és un tipus de programari que consisteix en definir una o vàries àrees virtuals on els membres que conforten aquesta xarxa (grup d'usuaris) poden compartir informació, documents, enllaços web, i alhora informar de les accions als altres membres del grup dins d'aquest espai compartit de treball mitjançant un ambient integrat de comunicacions.

Actualment existeixen un gran nombre d'aplicacions per a la col·laboració a distància, dirigides a diferents grups de persones. Un exemple, ja considerat *clàssic* d'espai de treball compartit és el BSCW. El BSCW (*Basic Support for Cooperative Work*) és un sistema de suport al treball en grup que proporciona facilitats per a la cooperació entre persones allunyades físicament . El principal suport per a la comunicació d'aquest sistema és la Web, amb la qual cosa els usuaris necessiten únicament tenir navegador web. Els espais de treball resideixen al servidor de BSCW, que conté una base de dades per a guardar els objectes (documents, avents, calendaris, tasques , URL's, converses). La principal limitació del BSCW és que segueix un model *client-servidor*, on tots els documents que pertanyen a un

mateix grup de treball han de residir físicament a un servidor central (a una entitat en clúster). El principal problema del model *client-servidor* és que un fallo al servidor pot provocar pèrdua d'accés a la informació per part dels usuaris clients.

Així doncs, podem dir que el punt de partida del projecte *eAula* és la creació d'un espai col·laboratiu de treball compartit similar al BSCW, però seguint un model distribuït, en lloc del model *client-servidor*. Es pretén millorar aspectes com escalabilitat, tolerància a fallides i disponibilitat de la informació i com a darrera fita, tenir una xarxa de col·laboració auto organitzada sense la necessitat d'una entitat administradora.

Per fer del projecte *eAula* un model totalment distribuït s'ha fet servir el concepte de la xarxa p2p per resoldre els problemes del model *client-servidor*. El terme *peer-to-peer* o P2P (també a vegades anomenat d'igual a igual) es refereix al sistema on tots els ordinadors de la xarxa són capaços de compartir recursos entre ells sense la necessitat d'una entitat arbitral, commutador o proveïdor de recursos globals. Un sistema *peer-to-peer* organitza el sistema no fent la diferència entre rols de client i servidor, sinó en la cooperació i la col·laboració entre iguals, permetent als participants formar la seva xarxa pròpia formant una estructura lògica[1].

Per la seva pròpia estructura, tipologia i funcionament, la xarxa p2p són ideals per aplicacions distribuïdes, en concret xarxes de distribució de contingut i sistemes cooperatius de fitxers.

2.2 Objectius

L'objectiu principal d'aquest projecte és la creació d'un espai d'intercanvi de documents (espais de treball compartit) entre els membres d'una aula (entesa com a alumnes i professor d'una determinada matèria i grup) en entorns distribuïts usant els recursos aportats pels membres del grup.

La idea central és crear una aplicació web totalment distribuïda i descentralitzada sobre una xarxa p2p per fer possible aquest intercanvi de recursos. És necessari que implementi els següents serveis o capacitats:

1 Creant el que s'anomenra overlay network (http://en.wikipedia.org/wiki/Overlay_network)

- Possibilitat d'accedir a documents i carpetes creades i compartides per altres usuaris de la xarxa.
- Creació de carpetes o directoris
- Possibilitat de poder notificar missatges als altres usuaris que utilitzin aquest espai d'intercanvi de documents.
- Gestió (creació, modificació, esborrat) de marcadors (*marcadors*).
- Indexació del recursos compartits dins estructures Hash distribuïdes.
- Possibilitat de fer cerques ràpides i globals dels continguts compartits entre els membres de l'aula.

2.3 Enfocament i mètode seguit.

Una vegada fixats els objectius que s'han d'intentar aconseguir cal comentar la metodologia utilitza per a poder desenvolupar de la manera més optima possible el projecte eAula. El mètode que s'ha anat seguit està molt lligat en la manera d'estructurar l'aplicació.

Com veurem més endavant eAula s'ha dividit en capes, segons la seva funcionalitat: capa de presentació, capa de lògica i servei i persistència de dades. Cada capa representa un espai de funcionament diferent i un marc per a enfocar el treball.

Així primerament hem definit la capa de presentació: s'ha pensat i dissenyat la funcionalitat de les pantalles, el que es pretenia en cada una d'elles així com la manera d'interacció amb l'usuari.

Un cop definir les pantalles bàsiques s'ha continuat implementant els serveis necessaris per que la capa de presentació pogués interactuar de manera lògica. La implementació dels serveis va lligada amb la implementació de les classes de negoci (classes que representen les entitats pròpies del nostre model de negoci). Així que el desenvolupament de servei i classes de negoci ha hagut d'anar a la par.

A mesura que avançava el desenvolupament s'ha vist la necessitat de testetjar i provar cada mètode que s'anava implementant. Així hem hagut d'integrar les proves unitàries per a veure el correcte funcionament del mètodes més complicats (partició de fitxers, reensamblatge de les parts, calculador de costos dels distints nodes)

2.4 Breu descripció dels altres capítols de la memòria.

Dins aquesta memòria/resum del projecte eAula hi figuren vuit capítols on s'explica la manera com s'han anat implementant els objectius o fites proposades a l'inici del projecte. En cada capítol s'intenta primerament exposar el problema de manera global i després explicar cóm s'ha tractat específicament a eAula. Els capítols son:

- "Planificació del projecte": En aquest capítol o apartat ens dona la distribució temporal que s'ha anat seguint per desenvolupar el projecte eAula. Hi figuren aquí els diagrames de Gannt que han servit de model per el desenvolupament del mateix.
- "eAula i la xarxa p2p. EasyPastry" : es dóna una visió general de les problemàtiques de les xarxes p2p així com de les seves avantatges i inconvenients enfront a altres sistemes tipus client/servidor. S'explica en quina manera s'utilitza la xarxa p2p a eAula i com ens ajudam de EasyPastry per implementar funcionalitats p2p.
- "Estructura interna de eAula: aplicació en capes": capítol destinat a explicat l'arquitectura utilitzada per dissenyar eAula. Mencionam les avantatges d'utilitzar una estructura dividida en capes. És especialment útil el diagrama que ens esquematitza aquesta estructura. Un cop explicada com es divideix, els altres tres següents capítols desenvolupen cada una de les capes en què es divideix eAula.
- "Mecanisme de persistència (DHT)" : capítol destinat a explicar la primera capa de l'estructura anterior. Aquí hi apareix unes classes molt important que defineixen l'estructura del negoci (classes de negoci). Es comenta unes propietats importants que han de complir pel correcte funcionament i s'adjunta un model UML que clarifica les relacions entre aquestes entitats. En aquest capítol s'explica també la manera de compartició dels recursos i la forma de descarregar-los.
- "Accés a dades i serveis": Seguint explicant la segona capa en què està dividida l'aplicació: l'accés a dades i lògica de negoci. Aquí s'explica els mecanismes per accedir a la persistència així com els possibles errors que ens podem trobar al llarg d'aquesta lògica.
- "Presentant la informació": En aquest apartat resumim de quina manera eAula presenta la informació a l'usuari. S'explica l'ús que se'n fa de les tecnologies Ajax

com a mecanisme per incorporar trets de les RIA. Explicam també la forma com es comunica les dades des d'aquesta capa fins a la capa de servei per enviar i rebre informació.

- "Indexant i consultant la informació". Apartat dedicat a detallar la manera com indexam la informació compartida per a poder fer recerques de recursos utilitzant els serveis i la capa de persistència.
- "Projecte eAula. Instal·lant eAula": Baix aquest títol hi ha descrit la manera física cóm està organitzat el projecte. Explicam pas a pas la forma d'instal·lar el projecte, compilar-lo i desplegar-lo al servidor mencionat també a aquest capítol. Especialment important aquí és la distribució de les carpetes que defineixen cada mòdul així com l'explicació de la part de les proves unitàries.
- "Manual d'usuari": Finalment hi ha un manual d'ús de l'aplicació web, un cop instal·lat i desplegat al servidor.

3 Planificació del projecte

El desenvolupament del projecte es divideix principalment en dues fases, encara que hi ha present una fase inicial destinada a la configuració de l'entorn de treball així com totes les eines que s'utilitzaran (servidor, llibreries, tasques de compilació...)

- **Fase inicial**: La fase inicial consisteix en la instal·lació i configuració de l'entorn de treball, creació i configuració de l'estructura global del projecte (veure apartat anterior sobre l'estructura física del projecte), creació dels *builds* i les tasques necessàries i pertinents, encarregades de compilar i construir l'aplicació i desplegar-la al servidor. Aquesta fase inicial tenia una duració prevista de quatre dies (des de 1/10 fins a 4/10) i com es pot apreciar al diagrama Gannt s'ha assolida correctament. Concretament les tasques d'aquesta fase inicial son:
  - Instal·lació entorn de treball
  - Creació estructura del projecte
  - Creació del build.xml i les tasques necessàries, encarregades de compilar, construir l'aplicació i desplegar-la al servidor Jetty.
  - Testetjar el correcte funcionament de tot l'entorn de treball.
  - Testetjar el correcte funcionament del build.xml.
- **La primera fase** representa el gruix de l'aplicació i com hem dit abans, ha d'implementar les tasques bàsiques de l'espai de treball compartit. Les tasques més importants que s'han de realitzar al llarg d'aquesta fase són:
  - Disseny de la interfície d'usuari:
  - Interfície de validació.
  - Interfície de gestió de carpetes i recursos.
  - Interfície per a la gestió (creació, esborrament) de marcadors amb la possibilitat d'incorporar a l'espai compartit el document referencial.
  - Desenvolupament dels serveis de negoci:
    - Serveis destinats a la validació de l'usuari dins la xarxa.

- Serveis destinats a gestionar l'accés tan a documents com a directoris compartits.
- Creació de documents i directoris
- Serveis destinats a la gestió dels marcadors (creació, modificació, esborrament)

■ Integració dels serveis de negoci amb la interfície: Aquesta integració s'anirà fent al llarg de la implementació dels serveis i com a punt de partida per veure el correcte funcionament d'aquests serveis. Així mateix necessitarem una darrera fase d'integració per acabar d'ajustar els requisits de la interfície de l'usuari amb els requisits dels serveis creats.

■ Provar tan amb proves unitàries com en proves sobre l'entorn desplegat del correcte funcionament dels serveis creats i de les interaccions amb l'usuari.

La programació temporal d'aquest primera fase presentava un diagrama Gantt com el següent:

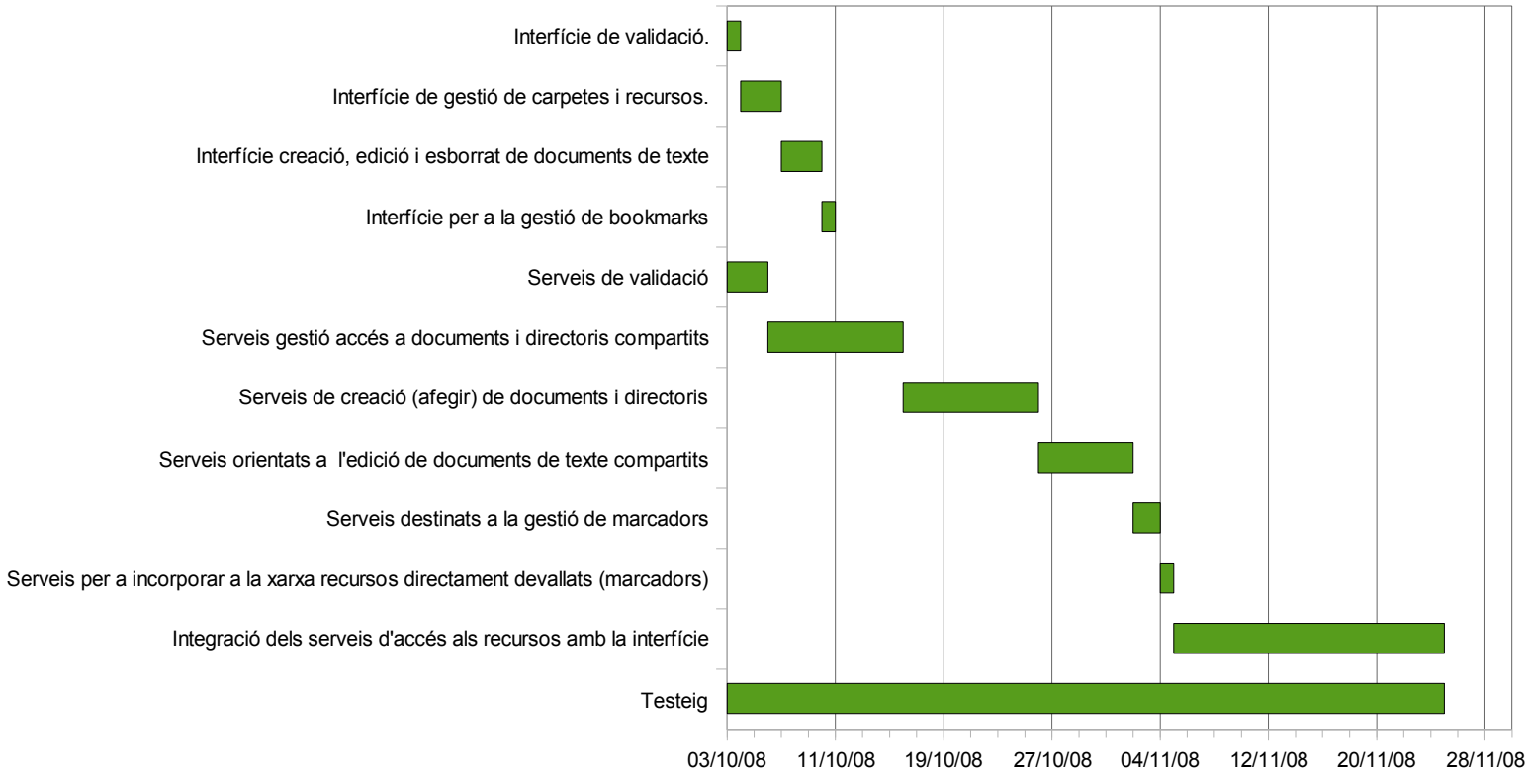


Figura 1: Diagrama de Gannt de la primera fase

En aquesta fase que començava el 3/10 i arribava fins a 27/11, s'havia de definir les interfícies d'usuari -interfície de validació, interfície de gestió de carpetes i recursos, la interfície per a la gestió dels bookmarks.

- La **segona fase** representa un valor addicional a l'espai de treball compartit. S'ha d'afegir un sistema d'indexació de la informació compartida (veure capítol 7: Indexant i consultant la informació)

    - Indexació de les dades dins estructures DHT.
    - Creació dels serveis necessaris per tal de poder fer cerques de la informació enregistrada.
    - Disseny de la interfície d'usuari per tal de poder fer recerques de continguts.
    - Interfície per a fer recerques senzilles.
    - Provar tan amb proves unitàries com en proves sobre l'entorn desplegat del correcte funcionament dels serveis creats i de les interaccions amb l'usuari.

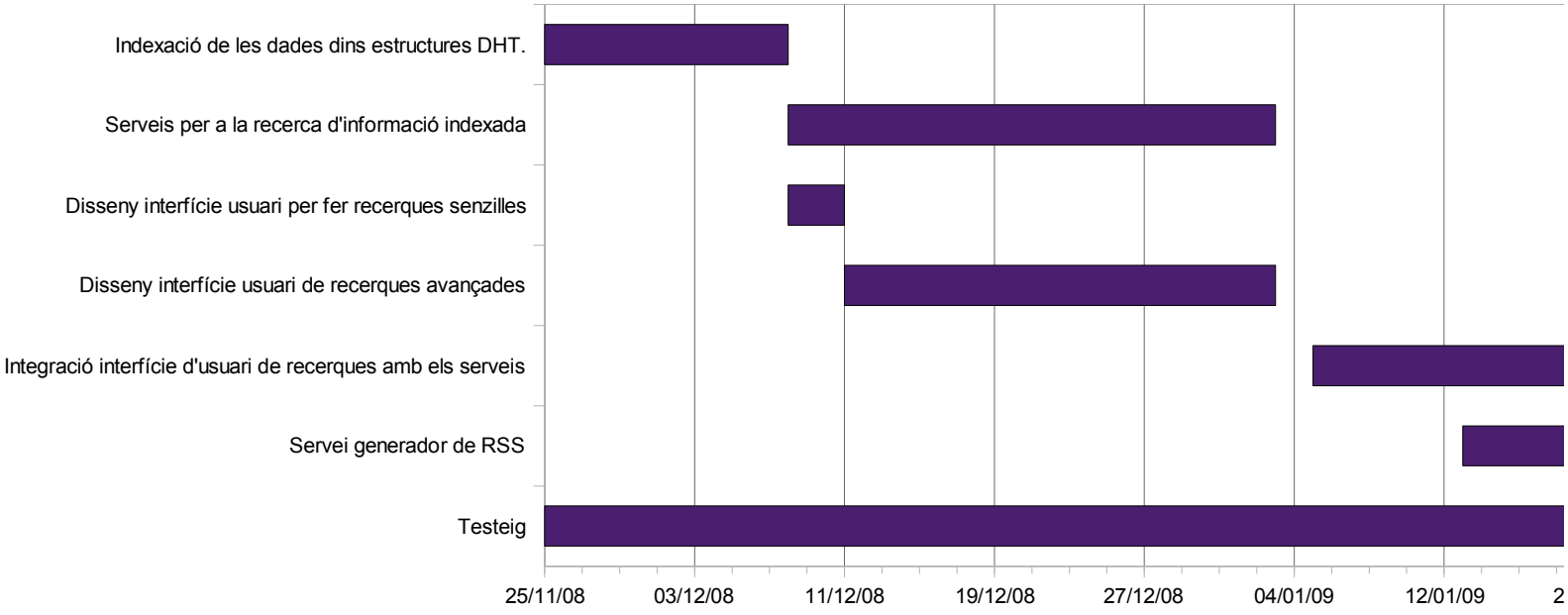


Figura 2: Diagrama de Gannt de la segona fase

- Si observam la distribució temporal d'aquesta segona fase mitjançant el diagrama Gantt anterior podem veure que s'havia plantejat iniciar-la dia 25/11 i presentava una duració fins el 20/01.

4 eAula i la xarxa p2p. EasyPastry

Com hem introduït abans, un sistema *peer-to-peer* organitza la xarxa no en termes de rols diferents client i servidor, sinó en la cooperació entre iguals. En aquest tipus de sistema els ordinadors o nodes que tradicionalment venien a desenvolupar el rol del client únicament, passen a actuar com a clients i servidors alhora: aporten i consumeixen recursos de la xarxa. No hi ha cap node (ordinador) que determini el comportament de la xarxa sinó que és un comportament emergent o auto organitzat. El model peer-to-peer aprofita l'avantatge que ens aporta la potència col·lectiva de tots els nodes de la comunitat o xarxa que participa.

Els avantatges d'utilitzar una xarxa p2p enfront als sistemes client-servidor ve donat de començament per la pròpia naturalesa cooperativista de l'aplicació *eAula*. També podem enumerar els següents punts on es posa de manifest els aspectes forts d'aquesta tecnologia:

- **Robustesa**. Donada la natura distribuïda de les xarxa *peer-to-peer* i la no presencia de cap punt singular (servidor central), en cas de fallida d'un node en concret, els altres usuaris podran continuar desenvolupant la seva tasca sense pèrdua d'informació.
- **Escalabilitat**. El sistema és escalable si es manté efectiu quan hi ha un increment significatiu en la quantitat de recursos i nombre d'usuaris. A les estructures en xarxes p2p, quan més nodes hi hagi connectats, millor serà el seu funcionament. Així quan els nodes es van connectant a la xarxa i afegeixen recursos propis, els recursos globals del sistema augmenten.
- **Repartiment de costos**. Els costos globals estan repartits entre els distints nodes que formen la xarxa. Així podem dir que es pot utilitzar un espai extra d'emmagatzemament per enregistrar còpies dels documents.
- **Alta disponibilitat**. En sistemes peer-to-peer la caiguda d'un node no causa un bloqueig del servei.
- **Transparència**: procurar que certs aspectes del sistema siguin invisibles a les aplicacions (per exemple la rèplica de dades entre nodes)

Com hem anat veient, els sistemes p2p permeten als participants formar la seva pròpia xarxa, confeccionant una estructura lògica anomenada *overlay network*. Les xarxes *peer-to-peer* overlay i descentralitzades reben el nom també de xarxes o substrats **KBR (Key Based Routing)** degut al fet que l'enrutament dels missatges es funció de la clau dels nodes.

Les xarxes KBR proporcionen mecanismes tan valuosos com poden ser les taules de hash distribuïdes (DHT) i el Cast. Les xarxes KBR cada *peer* és responsable d'un cert rang d'identificadors. El cost d'enrutar un missatge cap a un node (*peer*) responsable l'un identificador s'incrementa considerablement amb la mida de la xarxa.

4.1 DHT

Com hem dit abans les xarxes *overlay* distribuïdes ofereixen un servei anomenat DHT (*Distributed Hash Table*), taula *hash* distribuïda, que és una classe d'un sistema descentralitzat distribuït que proveeix un servei de recerca similar a les taules *hash* tradicionals confeccionades a partir d'un parell {clau,valor}. En una DHT cada node o *peer* participant de la xarxa pot accedir de manera eficient al valor de la clau emmagatzemada. La responsabilitat del manteniment de les relacions del parell {clau,valor} es distribuït al llarg de tots els nodes, de tal manera que un canvi en el conjunt dels participants causa els mínim trastorn. Això permet a les estructures DHT poder escalar una quantitat elevada de nodes. Les DHT utilitzen un sistema de replicat entre els nodes per a assegurar que les dades emmagatzemades sobreviuran a les caigudes dels nodes. Utilitzen també un sistema de *caching* per a realitzar copies transitòries i balancejar la càrrega de les peticions sobre cada instància de la DHT.

Les principals característiques de les DHTs posen èmfasis en les següents propietats:

- **Descentralització**. Els nodes formen un sistema col·lectiu sense cap coordinació central.
- **Escalabilitat**. El sistema funciona eficientment amb una quantitat relativament gran de *peers*.
- **Tolerància a fallides**. El sistema ha de ser tolerant i fiable tenint contínuament nodes adherint-se, abandonant i fallant en la xarxa.

La tècnica més comunament utilitzada per implementar i aconseguir aquests trets és el fet que un node necessita coordinar-se amb només uns altres quants nodes del sistema, de tal manera que només influeix un quantitat limitada de càrrega per a cada membre de la xarxa.

4.2 Cast (Multicast / Anycast)

Un altre servei comunament utilitzat a les xarxes KBR és el CAST. Entendrem per cast com un servei genèric, escalable i eficient sistema de comunicació de grup i notificació d'esdeveniments. Les xarxes overlay distribuïdes en nodes sovint han de subscriure's a un determinat grup, enviar i rebre missatges de grup o simplement enviar missatges a determinats nodes de la xarxa. Els serveis de CAST estan dissenyats a tal fi: gestionar la comunicació del grup de nodes i permetre enviar i rebre missatges de manera eficient.

4.3 EasyPastry

Per tal d'implementar la xarxa peer-to-peer *eAula* utilitza unes llibreries *middleware* anomenades EasyPastry. Aquestes llibreries estan construïdes i dipositades a sobre Pastry (abstreuen en seu comportament) i és la encarregada de proveir una completa façana a tal fi de simplificar el desenvolupament d'aplicacions p2p.

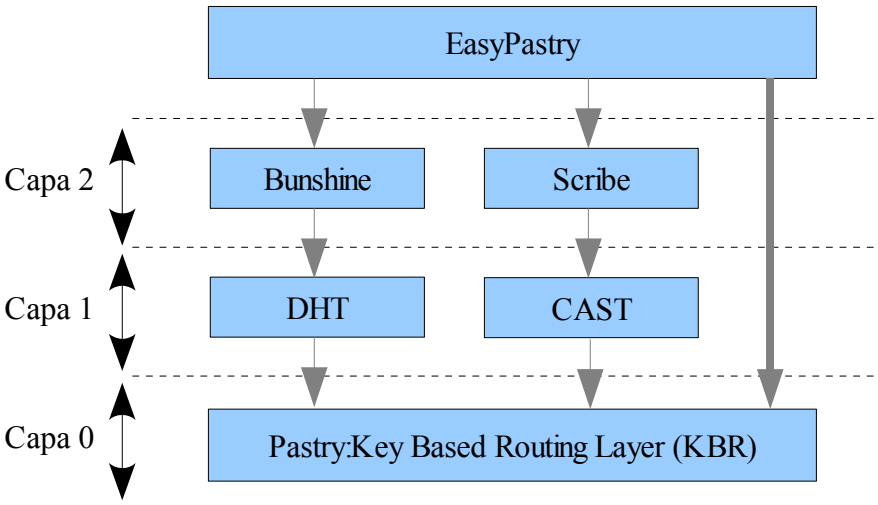


Figura 3: Estructura EasyPastry

Com es pot apreciar a l'esquema de la figura anterior, utilitza **Pastry** per a implementar la capa KBR. Pastry és un sistema genèric de localització d'objectes *peer-to-peer* i enrutador

descentralitzat basat en enrutament de claus sobre xarxes p2p. Pastry s'ha d'entendre com un substrat general per a la construcció d'aplicacions web p2p com sistemes globals de compartició d'arxius, emmagatzemament de recursos, comunicació de missatgeria i sistema de noms.
Per implementar les funcionalitats de DHT, EasyPastry utilitza un mòdul anomenat **Bunshin**, que proveeix els mètodes estàndards d'accés a les DHTs, és a dir el *get* i el *put.*
Finalment s'ha de dir que EasyPastry utilitza el sistema **Scribe** com sistema de comunicació de grup i notificació d'esdeveniments

Com s'ha dit abans, eAula és una aplicació web que es fa servir de EasyPastry per a implementar/utilitzar les funcionalitats pròpies de les aplicacions p2p. El primer problema que hem de solucionar és la manera d'inicialitzar la nostra xarxa KBR amb les llibreries EasyPastry. Això es fa just en el moment quan es desplega l'aplicació al servidor mitjançant un servlet amb la propietat on startup, encarregat de inicialitzar la xarxa KBR.

5 Estructura interna de eAula: aplicació per capes.

eAula s'ha dissenyat per ser una aplicació web en capes. S'entén per aplicació multi-capa (arquitectura n-capa) com una arquitectura de *software* tipus client-servidor en la qual la presentació, gestió de dades i lògica de negoci o procés lògic, es troben separats de manera lògica (o inclús físicament, utilitzant distintes màquines). Distintes responsabilitats han de formar distintes capes (separació d'incumbències) que interaccionen entre elles mitjançant interfícies. El model bàsic d'aplicacions n-capes, com hem dit abans, és el model 3-capes:

- Presentació: gestiona la navegabilitat i interacció amb l'usuari, és a dir aquells aspectes relacionats amb la lògica de presentació de l'aplicació.
- Dades o persistència: aquesta capa és l'encarregada de la persistència de les dades , és a dir gestiona l'emmagatzemament i accés a les mateixes.
- Domini de l'aplicació: resultat de l'anàlisi funcional de l'aplicació. Hi haurà les classes que ens abstreuen el model conceptual així com les operacions de negoci requerits.

5.1 Beneficis del model per capes

Separant les responsabilitats dels processo de l'aplicació en diverses capes fa més fàcil l'escalabilitat de l'aplicació. Si consideram que cada capa pot estar en màquines físicament separades (no és el cas, en principi, del projecte eAula) podem suportar millor càrregues de tràfic i connexions elevades.

Utilitzant aquest model d'arquitectura per capes, es permet separar el treball de manera més clara. Es pot anar enfocant el desenvolupament de l'aplicatiu de forma paral·lela de les tres parts: presentació (part client), persistència o gestió de dades i la lògica d'aplicació i accés a dades.

Una aplicació estructurada en n-capes facilita el seu manteniment i la seva llegibilitat. I fa que els seus components siguin **reusables**.

Finalment podem dir que l'estructura per capes de les aplicacions web fa augmentar la seva robustesa i tolerància a errades.

5.2 Estructura de eAula

Com es pot veure (figura 1) *eAula* s'ha estructurat l'arquitectura com una estructura dividida bàsicament en tres capes:

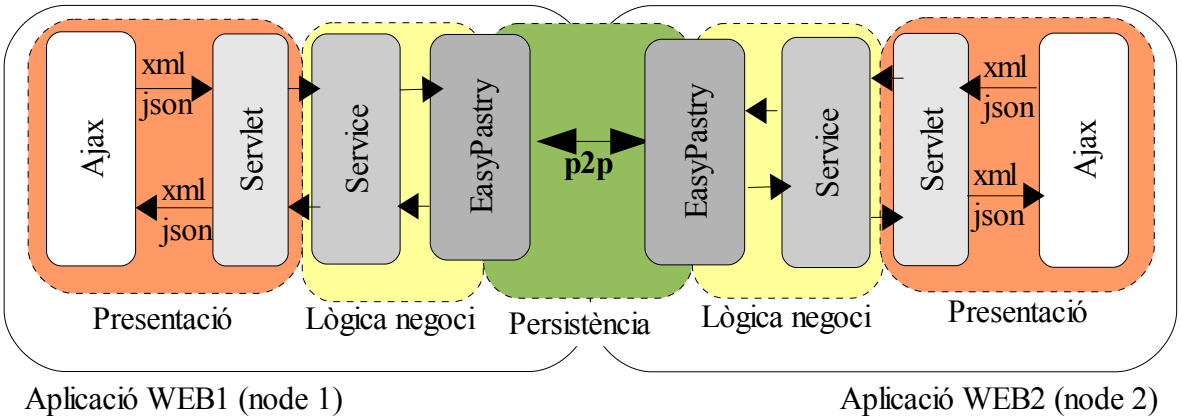


Figura 4: Estructura global de eAula

- **Persistència**: la capa de persistència o dades implementa mitjançant l'ús de DHT. L'accés a les DHT ve totalment abstreta i facilitada per EasyPastry. S'encarrega d'emmagatzemar i actualitzar les dades distribuïdes al llarg de xarxa p2p. Aquesta capa s'encarrega també de les tasques de replicació de la informació, sincronització, etc.
- **Lògica de negoci i servei**: Aquesta capa encapsula la lògica d'aplicació, les classes de domini identificatives de les entitats pròpies (classes que abstreuen els directoris, els usuaris,..). Aquesta capa ens publica també tots els serveis necessaris per accedir a les dades persistents: per exemple tenim a aquesta capa un servei encarregat de portar les dades concretes d'un directori donat el seu nom (concretament el seu identificador). També ens proveeix de les excepcions personalitzades per fer arribar a la capa de presentació qualsevol tipus d'incidència en el flux d'aquesta lògica.
- **Presentació**: És l'encarregada de la interacció amb l'usuari. Utilitza un conjunt divers de tecnologies per tal de fer les aplicacions web interactives i més usables (sense la necessitat de recarregat tota la pàgina) i els doten de comportaments i esdeveniments propis de les aplicacions d'escriptori

Per altre banda cal dir que per aprofitar clarament les avantatges que ens presenta el fet de ser una aplicació web. Això ens permet que a un **node** en concret on s'hi hagi instal·lat l'aplicació web s'hi podran connectar varis **usuaris.** Per tant no hi ha una correlació directa entre node i usuari, comportament intrínsec de les aplicacions web.

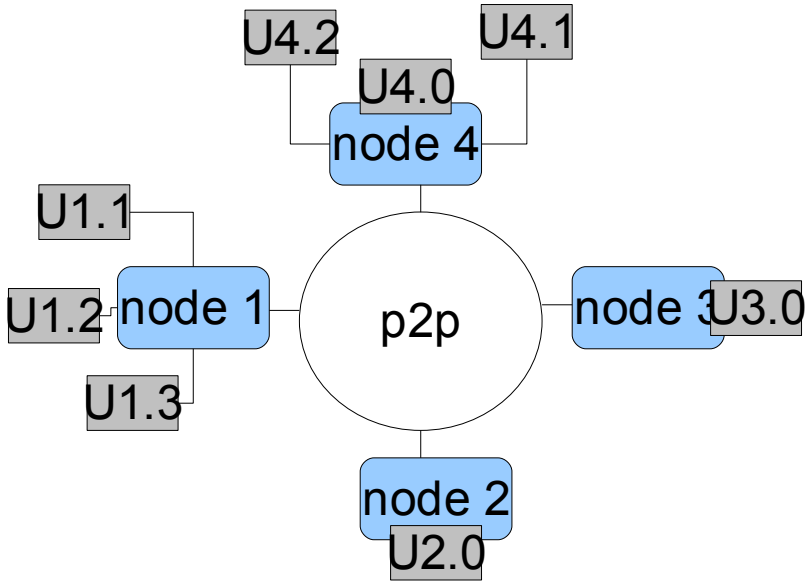


Figura 5: Estructura xarxa eAula

L'estructura de la xarxa d'usuaris i nodes a eAula presenta un esquema semblant a l'anterior on cada node (ordinador amb l'aplicació web instal·lada) pot funcionar donant servei a un únic client o usuari terminal de la xarxa -el propi node-, o pel contrari es pot aprofitar de la naturalesa d'aplicació web i poder oferir serveis a altres usuaris.

D'aquesta manera el fet de posar un recurs a compartir esdevé fer un upload a un node en concret del recurs que es vol compartir

6 Mecanisme de persistència (DHT).

Hem comentat anteriorment que les DHT proporcionen funcionalitats semblants a les taules *hash* tradicionals, emmagatzemant la relació {clau, valor} però en sistemes distribuïts i en xarxes KBR. Els valors emmagatzemats dins aquestes DHT poden ser de qualsevol tipus d'objecte amb la restricció de que han de ser *serializables*.

Les taules de *hash* distribuïdes proveeixen de dues operacions bàsiques: put({clau,valor}), inserir un parell {clau, valor}, i la funció per a obtenir un determinat valor segons la clau, valor = get(clau).La capa de persistència DHT presenta les següents característiques principals:

- Replicació : Per garantir la fiabilitat de les dades en xarxes p2p no solament han d'estar presents al node responsable sinó que els altres nodes també les han de contenir. Per aconseguir-ho s'ha de replicar aquestes dades als altres nodes, especialment als nodes veïns del node responsable.
- *Caching* adaptiu: Per millorar el rendiment a l'accés a la informació dels nodes s'utilitza una tècnica de *caching* anomenada sistema de *caching* adaptatiu, segons el nombre de peticions en un espai de temps sobre un determinat parell {clau,valor}
- El mode de persistència utilitzat és el mode de persistència a disc, emmagatzemant les dades d'un context en un determinat directori.

6.1 Classes de negoci

A *eAula* tots els objectes emmagatzemats a les estructures DHT configuren l'anomenat domini de l'aplicació o classes de domini. Les classes o objectes de domini ens defineixen i representen les entitats que el nostre model ha de ser capaç´de suportar. Les classes de domini encapsulen i abstreuen totes les dades i comportaments del negoci associades amb l'entitat que representa.

Les classes de negoci són les responsables de representar els conceptes del negoci (recursos, directoris, marcadors,...), informació sobre la situació i rols del negoci[2].

2 Martin Fowler

6.2 Serialització i clonació

Al projecte *eAula*, en general, les classes / objectes de negoci presenten dues propietats importants: **serialitzables** i **clonables**.

- **Serialització**. La serialització (*marshalling*) és un procés de codificació d'un objecte (estam parlant de programació orientada a objectes POO) en un mitjà d'emmagatzemament (arxius, buffers,...) amb la finalitat de transmetre'l a través d'una connexió de xarxa com una sèrie de bytes. Aquesta sèrie de bytes podes ser tornats a utilitzar per a crear un nou objecte clon de l'original. Per tant les classes de negoci de *eAula* implementaran la interfície de java *Serializable*.
- **Clonació**. Per tal d'evitar la referència creuada entre dos objectes del mateix tipus s'utilitza el mètode de clonació d'objectes. En aquestes situacions desitjam que un element que se passa a una funció o un objecte que crida a una funció membre no es modifiqui durant la cridada, per això és útil tenir la possibilitat de poder realitzar còpies de l'objecte original i realitzar les transformacions/consultes a la còpia deixant intacte l'original. La major part de les classes de negoci de *eAula* implementen la interfície de java *Cloneable*.

6.3 Classes de negoci a eAula

A eAula tenim present les següents classes de domini: *UserData*, *FileDescriptorData*, *FolderData*, *ResourceData*, *MessageData*, *BookmarkData*, *FilePartData* i *HostData*. Hem seguit la terminologia d'usar el sufixe "Data" per fer constar que es tracta de classes de negoci.

- **UserData**: Classe del domini que abstreu i implementa les dades pròpies d'un usuari de la xarxa d'intercanvi. En aquesta estructura són presents propietats com nom usuari, paraula de pas (password), etc. Implementa les interfícies *Serializable* i *Cloneable.*

- **ResourceData**: És una classe base que representa qualsevol tipus de recurs dels que es gestionaran a *eAula*. Les altres classes que utilitzarem per a representar recursos hereten les propietats de ResourceData. Presenta dues propietats: un identificador (ID) i un tipus. Els identificadors dels recursos han de ser únics dins la nostra xarxa. La propietat tipus la feim servir per a saber en tot moment de quin tipus de recurs estam parlant.
- **FolderData**: Classe que ens representa les un directoris compartit dins eAula. Hereta de ResourceData les propietats de ID i tipus. Presenta les propietats pròpies a la correcta identificació i gestió de carpetes / directoris: nom, propietari, data creació, i directori pare. També presenta tres col·leccions on s'aniran emmagatzemant els distints recursos situats a aquest directori. Implementa les interfícies *Serializable* i *Cloneable.*
  - *id*: (Propietat heretada de ResourceData), cadena que identifica de forma única el directori. Els IDs es calculen a partir de la ruta completa del directori. Per exemple, un directori que presenta la següent ruta /home/usuari/Projecte1 l'id es calcularia aplicant un hash a aquesta ruta
  - *type* (heretada de ResourceData): tipus de recursos.
  - name: nom del directori. A l'exemple anterior el nom seria "Projecte1"
  - *owner*: identifica el propietari del directori. Aquesta propietat es fixa quan se crea el directori.
  - *created*: data de creació del directori.
  - *modified* : data de modificació de la carpeta
  - *parent*: Directori pare, és a dir directori on es situa l'actual. En cas de que sigui el directori arrel, parent serà nul.
  - *childFolders*: Col·lecció de carpetes o subdirectoris filles.
  - *files*: Col·lecció de arxius o recursos continguts per aquest directori.
  - *bookmarks*: Col·lecció de marcadors continguts per aquest directori.

  Es pot dir que FolderData pot ser vista com una **agregació** de directoris, fitxers i bookmarks.

- **FileDescriptorData**: Descriptor lògic d'un recurs tipus fitxer o arxiu que ens abstreu i presenta les propietats pròpies d'aquests recursos:
  - *id*: (Propietat heretada de ResourceData), cadena que identifica de forma única el fitxer. Com s'ha dit pels directoris, els IDs es calculen a partir de la ruta completa del fitxer. Continuant l'exemple, un arxiu que presenta la següent ruta /home/usuari/Projecte1/arxiu.txt l'id es calcularia aplicant un únic hash a aquesta ruta.
  - *type* (heretada de ResourceData): tipus de recursos.
  - *name*: nom de l'arxiu. A l'exemple anterior el nom seria "arxiu.txt".
  - *owner*: identifica el propietari del fitxer.
  - *contentType*: Cadena que defineix el Content-Type del fitxer.
  - *size*: ens informa de la mida de l'arxiu.
  - *created*: data de creació del fitxer (data en què es va compartir el fitxer)
  - *parent*: Representa el directori que el conté. És del tipus FolderData.
  - *keywords*: Paraules claus o descripció, que ens serveixen per caracteritzar aquest recurs. Seran utilitzades a l'hora de fer recerques de recursos.
  - *parts*: nombre de parts en què s'ha dividit el recurs.
  - *distributedHosts* :Col·lecció utilitzada per emmagatzemar els nodes on està distribuït aquest arxiu.
- **HostData:** Descriptor utilitzat per a abstreure les propietats dels nodes on estan distribuïts els fitxers així com les parts d'aquests fitxers. Presenta les següents propietats:
  - *host*: Cadena que identifica el host on està distribuït el fitxer.
  - *port*: Port utilitzat per a davallar el fitxer,o parts del fitxer, del host fixat anteriorment.
  - *parts*: Col·lecció de parts disponibles del fitxer a aquest host.
- **FilePartData:** Entitat que representa una part concreta del fitxer. Presenta una ID i una mida d'aquesta part.
- **BookmarkData**: Aquesta classe de la lògica de negoci representa els marcadors. Un marcador és un link o adreça associada a un nom en concret. Els marcadors es

registren també dins els directoris com si fossin fitxers normals. Presenta les següents propietats:

- *id*: (Propietat heretada de ResourceData), cadena que identifica de forma única el bookmark. Els IDs es calculen a partir de la ruta completa del bookmark. Continuant l'exemple, un arxiu que presenta la següent ruta /home/usuari/Projecte1/Bookmark1 l'id es calcularia aplicant un únic hash a aquesta ruta.
- *type* (heretada de ResourceData): tipus de recursos.
- *name*: Nom del marcador.
- *url*:URL a la que es refereix el marcador.
- *created*: Data de creació del marcador.
- *owner*: Usuari creador i propietari d'aquest marcador.
- *parent*: Directori al que pertany aquest marcador.
- *keywords*: Paraules claus o descripció, que ens serveixen per caracteritzar aquest boomark. Seran utilitzades a l'hora de fer recerques de recursos.

- **MessageData:** Classe utilitzada a l'hora d'enviar i rebre missatges de grup o d'usuaris (Cast). Utilitza les següents propietats:
  - *date* : Data d'enviament del missatge.
  - *text*: Cos del missatge que s'envia o es rep.
  - *sender*: usuari que envia en missatge.
  - *recipientsTo*: Col·lecció d'usuaris destinataris del missatge.

6.4 Diagrama UML de les classes de domini

A continuació presentam el diagrama UML de les classes que conforten el nostre domini per tal de veure esquematitzades les relacions entre elles.

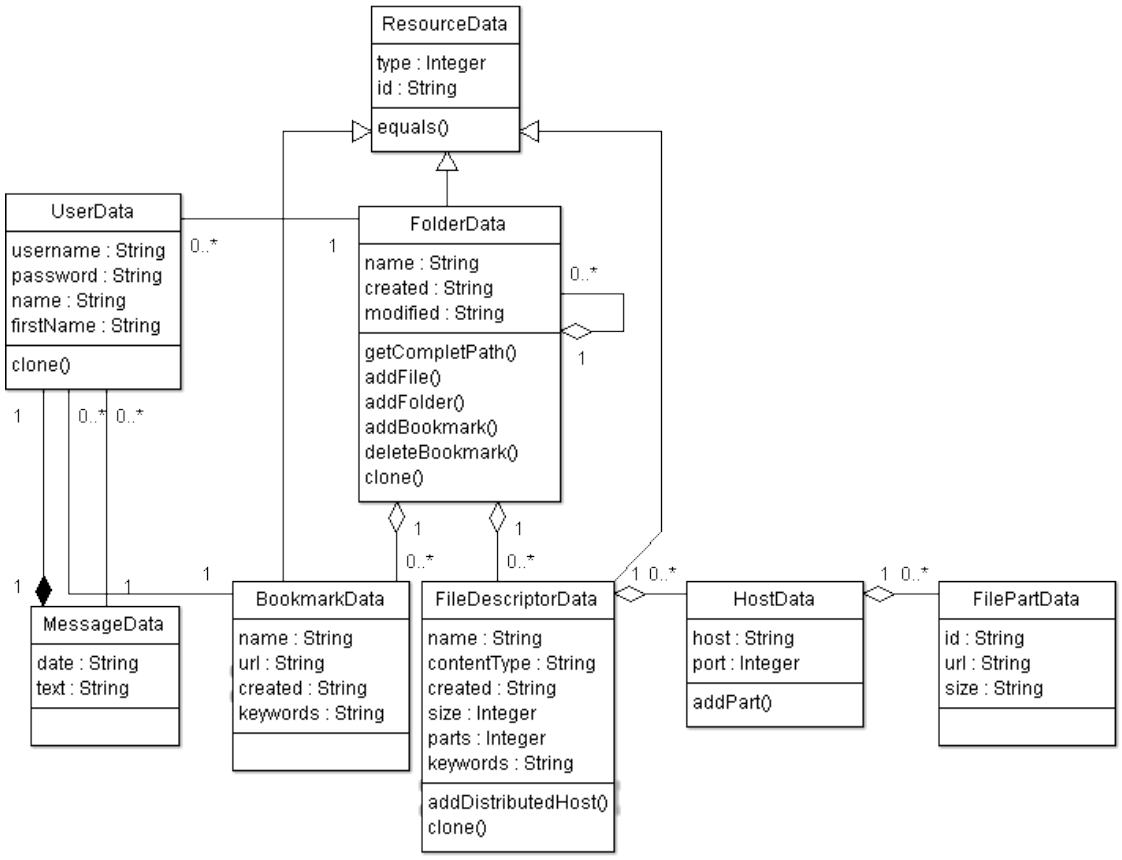


Figura 6: Diagrama UML de les classes de negoci

6.5 Emmagatzemament dels fitxers compartits i mètode de descàrrega.

Als punts anteriors, que feien referència a la part de la persistència de les dades i a la definició de les classes de negoci que abstreuen les entitats del model lògic de *eAula,* veiem que utilitzàvem els FileDrecriptorsData com a classes que representen el nostre fitxer que volem posar a compartició. Com podem apreciar aquesta classe no presenta cap propietat per emmagatzemar el contingut pròpiament dit del fitxer (del tipus *array* de bytes, buffers serialitzables,...), sinó que conté una col·lecció de HostData. Aquesta col·lecció de

HostData representa els distints nodes on hi ha replicat aquest arxiu. A la vegada, cada HostData es pot veure com una col·lecció o agregat de parts del fitxer FilePartData.

Així per exemple quan un usuari,situat a un de terminar node (node A), vol compartir el fitxer, primerament el divideix en n parts cada una de 1Mb. A continuació actualitza la DHT inserint el descriptor del fitxer (utilitzant un mètode servei de FileService, veure apartat següent) FileDescriptorData amb la col·lecció de HostData informada (recordem que utilitzam la classe HostData per identificar el node en el qual està replicat un fitxer). A la vegada anam actualitzant la col·lecció de parts de fitxer del HostData amb la informació de cada una de les n parts que s'han anat creat al procés de *splitting* del fitxer.

Quan un usuari d'un altre node (node B) vol descarregar-se aquest fitxer primerament obté la informació d'aquest descriptor i comprova inspeccionant la col·lecció de HostData que aquest fitxer es troba al node A. Comença a davallar-se les n parts via *URL connection*[3]. A mesura que va obtenint les parts actualitza els tan els valors de HostData (afegint un altre node, en aquest cas el node B) com els valors de les parts d'aquest nou HostData.

Al final d'aquest procés de davallada del fitxer tendríem que aquest descriptor de fitxer o FileDescriptor tendria una col·lecció de dos elements de HostData (node A i node B). A la vegada cada HostData d'aquesta col·lecció tendria una llista de parts del fitxer corresponents a les parts que té cada node en aquell moment.

Si en aquest moment un altre node que no té replicat aquest fitxer, per exemple el node C, intenta davallar-se'l se li presenta un dilema: de quin node ho ha de fer, del node A o del node B?

6.6 Polítiques de descàrrega

Hem vist a l'exemple anterior que quan tenim fitxers replicats en varis nodes i els volem davallar s'ha de decidir de quin node es davalla. Per fer una política de descàrrega més eficient hi ha vàries possibles solucions:

- **Aleatòria**: per evitar carregar els recursos d'un determinat node es pot fer un *random* entre tots els nodes que tenen aquest fitxer. Aquest mètode no té en compte temes de millor resposta o càrrega del node.

3 URLConnection crea un vincle (petició/resposta) entre el node que està executant l' aplicació i el computadir que i el node que ofereix el recurs.

- **Ping**: esbrinar el temps de latència dels servidors (nodes) més proper i per tant més òptim a l'hora de descarregar les parts de l'arxiu.
- **Càrrega del node**: esbrina quin dels nodes on estan replicades/distribuïdes les parts presenta menor càrrega i per tant la resposta podria ser millor.

Als darrers anys han sortit varis *plugins* per a resoldre aquest tema a l'hora d'escollir el millor node per a fer la descàrrega. Hi ha un *plugin* molt interessant anomenat ONO que fa enginyeria inversa sobre xarxes distribuïdes de continguts per a obtenir una resolució DNS dels nodes que tenen el fitxers o les parts, ordenar per latència les llistes i arriben a millorar un rendiment fins a un 30%.

Al projecte *eAula* hem implementat el mètode del *ping*. Hi ha un *plugin* anomenat CostCalculator encarregat d'esbrinar el menor temps de resposta ping d'una col·lecció de HostData.

7 Accés a dades i serveis.

Al capítol anterior hem vist de quina manera gestionam la persistència de les dades al projecte *eAula*. Hem vist que utilitzam les DHTs com a mecanismes per a enregistrar les nostres dades i que aquestes dades estan representats per estructures, anomenades classes de negoci, que ens abstreuen les entitats representades. Per accedir a aquestes representacions emmagatzemades (capa de dades) necessitam un conjunt de serveis destinats a gestionar la lògica d'accés a la persistència i la lògica de negoci.

Definim la capa de serveis com la lògica que ens permet accedir, recuperar, i actualitzar les dades. En aquesta lògica es poden produir excepcions. Per aquesta raó necessitam capturar les possibles errades d'accés i informar-ne a les capes superior (capa presentació) per a informar correctament a l'usuari dels esdeveniments.

Els serveis s'encarreguen de presentar a les capes superiors (web) una interfície per accedir a la persistència, de tal forma, que per exemple un servlet mai gestionarà directament les dades DHTs sinó que seran els distints serveis publicats. Un servei és una funcionalitat del negoci atòmica que accepta una o més sol·licituds i retorna una o més respostes a través d'una façana definida[4].

Els tipus de serveis es classifiquen en tres grans grups:

- **Bàsics**:centrats en l'administració de dades i regles de negoci associats
- **Compostos**: Integració de serveis bàsics a partir de la seva composició.
- **Processos**: serveis destinats a gestionar processos de negoci complexes mantenint el seu estat.

A *eAula* utilitzam el primer tipus de serveis: els serveis implementats es centren en l'administració de dades i regles de negoci. Administra la persistència i la gestió transaccional.

7.1 Excepcions

4 Un sistema completament construït a partir d'interfícies definides d'aquesta manera es diu SOA (Service Oriented Architecture)

L'accés a dades i gestió de la lògica de negoci produeixen excepcions. A *eAula* tenim les següents excepcions definides que ens ajuden a mantenir el flux lògic de l'execució de l'aplicació web : InvalidUserException, FolderException, FileException, ResourceException i BookmarkException. Les excepcions produïdes a la capa de servei per accés a dades incorrectes, insercions de objectes que ja existeixen, etc, es passen a les capes superiors (servlet i capa de presentació) mitjançant aquestes excepcions personalitzades.

7.2 Serveis de eAula

Gairebé totes les classes que conformen el nostre domini de l'aplicació tenen "associat" un servei que ens permet poder inserir i recuperar informació a les DHT, i en cas de presentar-se alguna anomalia llançar la corresponent excepció capturada a la capa superior (presentació). Així podem veure que disposam dels següents serveis:

- **FolderService**: servei encarregat de gestionar l'accés als directoris. Presenta tres mètodes per a manejar els directoris:
  - setFolder: Inserta un nou directori a la DHT. Llança una exceptió en cas que el nou directori ja existeixi (ID igual)
  - getFolder: Retorna la informació de directori associada a un ID. En cas que aquest ID no existeixi llançarà un FolderException indicant que aquesta carpeta no figura a la DHT de recursos.
  - updateFolder: Actualitza els valors del directori. S'utilitza quan afegim un determinat recurs a un directori. Llança un FolderException en cas de no poder realitzar la operació (per exemple que el directori no existeixi)
- **UserServide**: servei encarregat del correcta accés a les dades dels usuaris. Els mètodes que es poden fer servir son:
  - getUser: Retorna la informació de l'usuari donat un *username*. Retorna una excepció en cas que l'identificador d'usuari no existeixi.
  - setUser: Insereix informació sobre un usuari a la DHT. En cas que l'usuari ja existeixi llança una InvalidUserException
- **BookmarkService**: És l'encarregat de gestionar l'accés als marcadors. Es poden fer servir els següents mètodes:

- getBookmark: Retorna la informació d'un determinat bookmark donat un ID. En cas de no existir llança una excepció personalitzada.
- setBookmark: Inserta un marcador nou dins la DHT de recursos. També, en cas d'existir aquest ID llançarà un BookmarkException.
- deleteBookmark: esborra un marcador de la DHT de recursos.

- **ResourceService**: És un servei senzill i bàsic destinat a pode obtenir un determinat recurs independentment del tipus de recurs que sigui. Així només tendrà un mètode que serà:
  - get: Retorna la informació de recurs associada a un ID. En cas que aquest ID no existeixi llançarà un ResourceException indicant que aquesta recurs no figura a la DHT de recursos. Com que totes les classes de recursos hereten de ResourceData inspeccionant la propietat tipus es pot saber de què es tracta efectivament.
- **QueryService**: Servei utilitzat tan per indexar la informació com per executar consultes en base a descripcions a fi d'obtenir recursos.
  - insertMetaTags : insereix paraules claus i valors IDs a la DHT d'indexació.
  - executeQuery: executa una consulta en base a un determinada paraula o seqüència de paraules per a obtenir una col·lecció de recursos.
  - index: indexa un determinat recurs a través de la seva propietat de descripció o de *keywords*.
- **FileService:** servei encarregat de gestionar l'accés als fitxers. Presenta dos mètodes per a manejar els recursos de tipus arxius:
  - setFile: Posa un descriptor de fitxer dins la DHT. Si la ID d'aquest recurs ja existeix aleshores llança un FileException informant-ne del succés.
  - getFile: Obtenir un recurs de dins la DHT a partir de la ID donada. Si aquest arxiu no existeix dins la DHT es llança un FileExeceptions cap a la capa superior.

7.3 Serveis de missatgeria

Com hem comentat abans EasyPastry ofereix funcionalitats per gestionar serveis genèrics de comunicació de grup i notificacions de missatges i esdeveniments, l'anomenat CAST.

La primera acció per a poder usar correctament els serveis de missatgeria oferts per EasyPastry és **subscriure's** a la nostra xarxa. Per això cada usuari nou que entre a l'aplicació després de validar-se, s'ha de subscriure a eAula mitjançant.

Una vegada s'ha subscrit a la xarxa eAula ha de fixar l'event que s'executarà cada vegada que es rebi un missatge, és a dir s'ha de fixar un **Listener** (escoltador) que s'encarregarà del tractament de cada missatge rebut.

El **listener** implementat a eAula (**HttpListener**) és un servei molt senzill i s'encarrega d'emmagatzemar els missatges que arriben (MessageData) dins una col·lecció compartida per tots els usuaris del node. Per tal que sigui visible per tots els distints usuaris del node on ha arribat el missatge (veure figura 3: "Estructura de la xarxa eAula: nodes i usuaris") aquesta col·lecció es diposita al context de l'aplicació (*Servlet Context*).

Serà la capa de presentació l'encarregada de accedir periòdicament (mitjançant un timmer) a aquesta col·lecció de missatges rebuts, destriar-ne els missatges amb destinatari correcte i presentar-los per pantalla.

8 Presentant la informació.

Per implementar la capa de presentació, és a dir l'aspecte més proper a l'usuari eAula utilitza tecnologia Ajax. El terme Ajax és un acrònim de Asynchronous Javascript + XML i fa referència a un conjunt heterogeni de tecnologies independents que s'utilitzen conjuntament per tal de fer les aplicacions web més interactives i usables, alhora que les doten de comportaments propis de les aplicacions d'escriptori, el que es coneix com RIA (*Rich Internet Applications*). Les aplicacions implementades amb Ajax s'executen en el navegador web, i es comuniquen al servidor de manera asíncrona (en segon plà) per a obtenir dades (especialment en format *xml* o *json*) o per obtenir codi HTML ja formatat. Però Ajax pot ser vista com una arquitectura d'aplicacions web.

8.1 Ajax: Asynchronous JavaScript and XML

Com ja hem dit anteriorment Ajax pot ser vist com una combinació de diverses tecnologies operant en conjunt. Bàsicament aquestes són:

- XHTML i CSS per a crear una presentació basada en estàndards.
- DOM per a la interacció i manipulació dinàmica de la presentació.
- XML, XSLT i JSON per a l'intercanvi i manipulació de la informació.
- XMLHttpRequest, objecte per l'intercanvi asíncron de les dades.
- JavaScript per a unir programàticament les altres tecnologies.

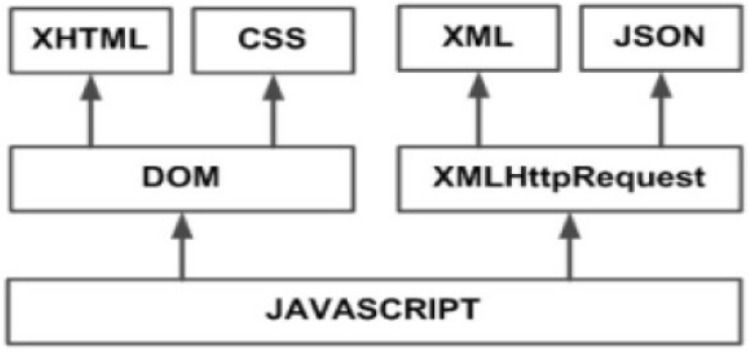


Figura 7: Tecnologies Ajax

8.2 Ajax: l'arquitectura

Ajax permet una nova arquitectura a l'hora d'implementar aplicacions web. Les parts importants d'aquesta arquitectura són:

- *Petits events de servidor*: Actualment els components d'una aplicació web poden fer petites peticions al servidor, obtenir informació i refrescar peces de la pàgina utilitzant el DOM, sense una completa càrrega de la pàgina.
- *Asíncron*: les peticions realitzades al servidor no causen un bloqueig del navegador. L'usuari pot continuar utilitzant altres parts de l'aplicació i el UI pot ser actualitzar quan el resultat de la petició estigui disponible.
- *onAnything*. Es pot fer una comunicació al servidor en gairebé qualsevol event que l'usuari realitzi (clicks, moseover, keypress,..)

8.3 Ext Js

Ext JS és un complet *framework* i llibreries de JavaScript que simplifica considerablement la tasca d'implementar i desenvolupar aplicacions AJAX. Ext JS usa molts de components reutilitzables així com widgets UI. Ext JS proveeix un disseny totalment orientat a objectes que inclou:

- Alt rendiment i personalitzable UI widgets.
- Suportat per la majoria de navegadors Web.
- Model de components extensible i ben dissenyat.
- Llicència Open Source.

Entre els components que aporta Ext Js cal mencionar: controladors de finestres de missatges, llistes desplegades, graelles de dades, menús, barra de tasques, arbres, etc. També aporta tot una sèrie de controls destinats a poder fer peticions asíncrones al servidor i tractar la resposta en diversos formats (json, xml, text , html).

8.4 Format de dades

Com hem dit abans per tal de fer interaccionar d'interfície d'usuari amb el servidor i poder recuperar la informació i posteriorment refrescar certes parts de la pàgina utilitzam

peticions asíncrones al servidor. El servidor ens retorna la informació bàsicament en dos formats:

- **XML**: És el format d'intercanvi dades de les aplicacions típiques en Ajax. És molt útil quan la complexitat estructural de les dades que es vol rebre/transmetre és elevada. Necessita un analitzador semàntic, a vegades costós (i no estàndard a tots els navegadors) per *parsetjar* el document i carregar-lo a memòria per a poder recorrer-lo
- **Json**: Acrònim de JavaScript Object Notation és un format lleuger d'intercanvi de dades, subconjunt de la notació literal d'objectes JavaScript. La simplicitat del JSON ha desenvolupat el seu ús, especialment com alternativa a XML ja que no necessita un analitzador semàntic tan costós com amb XML

8.5 Accés a dades des del client

El client (Ajax, Ext Js) necessita obtenir les dades en algun dels dos format comentats anteriorment: XML o JSON (veure figura Estructura global eAula). Per servir aquest tipus d'informació ens feim valer dels Servlets. Aquests utilitzen els serveis comentats anteriorment (*FileService*, *FolderServide*,...) per accedir a les dades emmagatzemades a la capa de persistència i els presenten al client en formats utilitzables.

És a dir la funció dels Servlets és utilitzar els serveis per accedir a les dades i presentar-les cap a la capa de presentació en un format adient. A l'hora de formatar les dades xml o json els Servlets es redireccionen o despatxen als jsp que son els encarregats de formatar les dades en el format adient.

L'accés a les dades es pot resumir en el següent esquema:

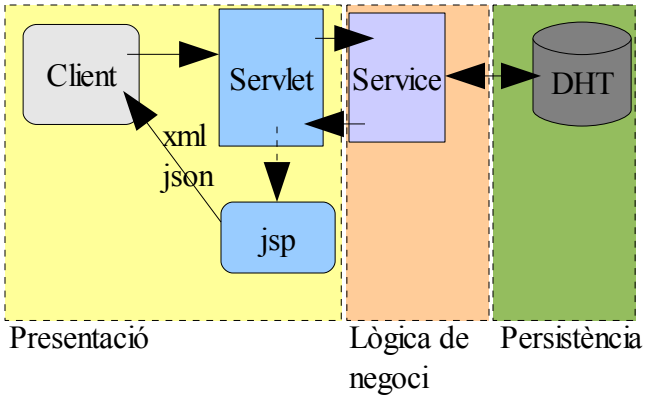


Figura 8: Esquema accés a les dades

8.6 Accés al missatges des del client

A l'apartat anterior s'ha explicat de quina manera es tractaven els missatges a la capa de servei. Quant a missatgeria la capa de presentació és l'encarregada d'accedir periòdicament (mitjançant un **timmer** implementat amb peticions Ajax cap el servidor) a la col·lecció de missatges rebuts emmagatzemada al context. Aquest col·lecció està composta de objectes tipus MessageData amb informació de missatges rebuts pel node.

Cada petició Ajax feta pel **timmer** inspecciona aquesta col·lecció i en destria únicament els missatges que el seu destinatari sigui igual a l'usuari validat. La informació de l'usuari validat es guarda en sessió i és fàcilment accessible des que qualsevol Servlet. Una vegada presentat aquest missatge cap al client s'esborra de la col·lecció de missatges a fi de no tornar-lo a enviar cap al client.

9 Indexant i consultant la informació.

La recerca de recursos (especialment fitxers i directoris) és un punt clau de la implementació de la segona fase del projecte. Per a poder fer aquestes recerques sobre una aplicació distribuïda cal tenir indexada de qualque manera la informació que volem cercar.
Les estructures DHT sobre una xarxa *peer-to-peer* KBR com el cas de EasyPastry (Pastry) fa simple l'accés a una determinada dada fixada per la clau coneguda anteriorment. A la pràctica els usuaris recerquen freqüentment recursos emmagatzemats en sistemes *peer-to-peer* únicament coneixent parts identificatives o informació parcial per identificar quests recursos. El que ens interessa és poder tenir indexada d'alguna manera els recursos per a poder accedir-hi a través de paraules claus. DHTs només suporten recerques amb un patró exacte per clau.
Com s'ha dit abans, cada recurs (fitxer, directori i marcadors) estan identificats per un ID únic per tota la xarxa. A cada recurs fixat per aquest ID li correspon com s'ha dit abans un FileDescriptor, un descriptor que ens permet associar-li dades com propietari, data creació, i una descripció textual identificadora. Per tant el nostre problema és bàsicament poder cercar documents per aquesta descripció textual, pel nom del fitxer o pel seu tipus.

9.1 Indexant

Coneguts els problemes estructurals de les DHT per a implementar recerques de recursos, al projecte *eAula* hem desenvolupat una recerque de recursos mitjançant índexs hash. L'esquema bàsic d'indexació és la partició de la descripció del recurs (FileDescriptor) en n cadenes per a ser indexades dins un DHT d'índexs. Així per exemple tenim una recurs que presenta una ID I i una descripció Q que es pot descomposar en n cadenes, per tant Q = {q1,q2...qn}. A la DHT d'índexs hi introduiríem els parells (q1,I), (q2,I)....(qn,I). Imaginem que volem indexar ara un altre recurs amb un ID J que presnta una descripció P = {p1,p2,...pk}. Per tant al hash d'indexació hi inserirem (p1,J),(p2,J),...(pk, J). Si algunes d'aquests p1...pk és igual a q1...qk (per exemple qi=pj) lo el resultat d'inserir dins la DHT seria (qi, [I,J]). Això significa que el valor de la clau qi, que és el mateix que pj, seria una col·lecció de ID's formada per I i J.
A mode d'exemple: tenim un FileDescriptor com el següent:

```
FileDescriptor1{
       id : 'AD58A570',
       description: 'JAVA BOOK EAULA'
}
```

A la DHT d'indexació hi tendríem les següents túples:

```
{'JAVA BOOK EAULA','AD58A570'}, {'JAVA','AD58A570'}, {'BOOK','AD58A570'}
i {'EAULA','AD58A570'}
```

Cal apuntar que primerament es posa dins la taula *hash* la descripció completa del recurs. A continuació s'insereixen la resta de les subcadenes que conformen la descripció completa. Imaginem ara que es comparteix un altre recurs amb un FileDescriptor com:

```
FileDescriptor2{
       id : 'BF223212A',
       description: 'AJAX BOOK EAULA'
}
```

Caldria inserir o actualitzar els següents parells-valors:

```
{'AJAX BOOK EAULA','BF223212A'}, {'AJAX','BF223212A'},
{'BOOK',['AD58A570','BF223212A']} i {'EAULA',['AD58A570','BF223212A']}
```

Actualment a eAula el rescursos s'indexen pels següents conceptes:

- Descripció dels recursos tan si són fitxers, directoris o marcadors.
- Nom complet del recurs.
- En el cas de fitxers, s'indexa per tipus de fitxers basats en el seu Content-Type

Quan un recurs és compartit per primer cop s'ha d'indexar amb els conceptes vists anteriorment. L'elecció de la descripció apropiada pel recurs baix la qual s'indexarà és arbitraria i depèn de l'usuari que decideix compartir aquest recurs.

9.2 Recerques i consultes

Un cop tenim la informació indexada en les estructures mencionades anteriorment cal realitzar tot un seguit de mètodes per a poder accedir i recercar aquesta informació. A l'apartat de serveis s'ha comentat l'existència d'un servei destinat a realitzar recerques i consultes de la informació prèviament indexada. L'objectiu és aconseguir els IDs de recursos que en la seva descripció (*keywords*), nom, o tipus coincideixi amb el patró de recerca. La consulta en base a un patró es realitza de la següent manera:

- Es cerca primerament si hi ha recursos amb un índex igual al patró de cerca complet.
- Per a cada paraula (subcadena) del patró es cerca un índex igual a aquesta subcadena. Per cada resultat obtingut es junta amb la col·lecció obtinguda al punt anterior a tal fi de no repetir IDs iguals.

Com a resultat de la recerca obtenim una col·lecció no repetida de recursos que, mitjançant el seu *type*, podem saber exactament de quin tipus estem parlant.

10 Projecte eAula. Instal·lant eAula

Hi ha dues formes d'instal·lar eAula i posar-ho en funcionament: a partir de les fonts del projecte o bé a partir de l'aplicació generada al fitxer eAula.war.

10.1 Requeriments previs a la instal·lació

10.1.1 Java 6

Per poder compilar i executar sense cap problemàtica eAula cal usar una versió 6 de Java i tenir fixar clarament el CLASSPATH cap als directoris d'aquesta versió.

10.1.2 Contenidor Web Jetty

eAula és una aplicació web. Com a tal necessita executar-se sobre un contenidor o servidor web. eAula s'ha dissenyat per a poder executar-se sobre un contenidor lleuger ja que no necessita de les funcionalitats del EJB ni dels web services, en primera instància. Per tant un contenidor idoni per al desplegament de eAula és el servidor d'aplicacions web Jetty, encara que es pot utilitzar qualsevol altre contenidor capaç de desplegar .war correctament.
Jetty és un servidor web open-source lleuger, basat en estàndards que es pot davallar de la següent pàgina: http://www.mortbay.org/jetty/

10.1.3 Ant

Ant és una eina utilitzada a l'hora de crear projectes java per a la realització de tasques mecàniques i repetitives, normalment durant la fase de compilació i construcció de l'aplicació. Presenta l'avantatge de no dependre de les ordres de shell de cada sistema operatiu sinó que es basa en arxius de configuració xml per a la realització de les distintes tasques, fent d'aquesta manera una solució multi-plataforma. Es pot trobar i descarregar Ant a la pàgina: http://ant.apache.org/

10.1.4 Xdoclet

Xdoclet són unes llibreries de codi obert generadores de codi, que està associada a la programació orientada a atributs, és a dir s'aconsegueix més funcionalitat afegint o agregant meta-data (atributs) al codi Java, utilitzant tags JavaDocs. D'aquesta manera la tasca de afegir Servlets, EJB, ... a les aplicacions Java ens facilita la confecció dels arxius de configuració, per exemple el web.xml o el ejb-jar.xml

10.2 Instal·lant *eAula* a partir dels fonts del projecte

L'arxiu eAula_1.0-src.zip conté l'estructura dels fonts del projecte. Un com descomprimida dins un directori de treball presenta un esquema com el següent:

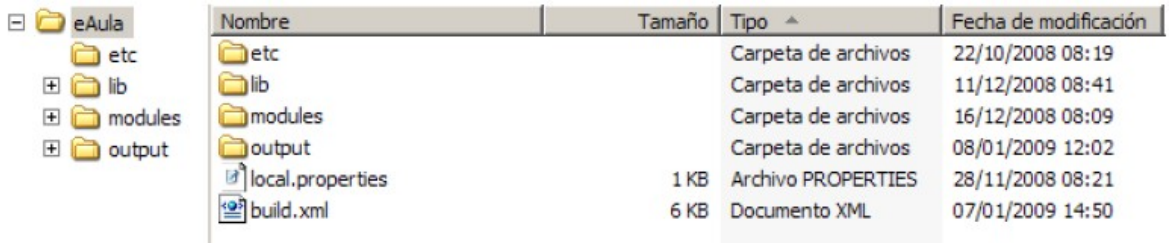


Figura 9: Estructura física del projecte eAula

Com es pot apreciar el directori de més alt nivell conté els següents subdirectoris: etc, lib,modules, output i els fitxers local.properties i build.xml.

- **etc** : és el directori on hi ha els fitxers de configuració de l'aplicació. Fitxers necessaris tan per una correcta compilació com per una correcta execució.
- **lib**: directori on hi ha totes les llibreries necessàries a l'hora tan de compilar com d'executar. Cal tenir en compte les versions de les llibreries. Així, dins aquesta carpeta hi podrem trobar per exemple easypastry.jar, bunshin.jar, pastry.jar. També hi podem veure que hi ha una subcarpeta anomenada xdoclet, on hi ha present les llibreries necessàries per a la correcta execució de Xdoclet
- **modules** : dins aquesta carpeta s'hi pot trobar tot el codi font pròpiament dit, organitzat en tres mòduls:
- **www**: funcionalitat pròpia de la web, tan elements estàtics (imatges, js, html) com elements dinàmics (servlets, jsp,..).
- **logic**: aquí dintre hi ha present tot l'aspecte de la lògica de l'aplicació: classes de domini, excepcions, serveis.
- **test**: mòdul per a fer les proves unitàries i el testeig global de eAula. A aquest mòdul hi figura un build.xml propi per a poder compilar-lo independentment que els altres dos mòduls.

- **output** :carpeta on es dipositaran les classes compilades així com el resultat final de la nostra creació del projecte (eAula.war).
- ***build.xml*** : fitxer general de configuració de les tasques ant, per a preparar, compilar, generar i desplegar l'aplicació.
- ***local.properties*** : fitxer de les propietats locals on definim, per exemple, el directori on s'ha de desplegar l'aplicació quan executam la tasca ant.

Un cop descomprimit l'aplicació, abans de compilar-la cal fer dues passes:

- Modificar el local.properties per fer apuntar al directori on es despleguen les aplicacions al contenidor Jetty. Per exemple

  `deploy.dir= e:/jetty-6.1.12/webapps`

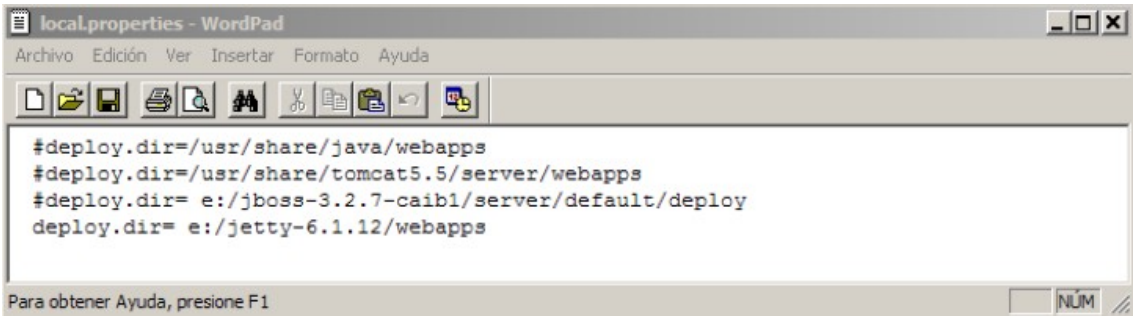


Figura 10: Modificant local.properties

- Dins el directori etc hi ha el fitxer easypastry-config.xml. Cal editar-ho i modificar l'entrada host: `<entry key="host">eedupro15</entry>` posant el nom del host on instalarem l'aplicació per primer cop:

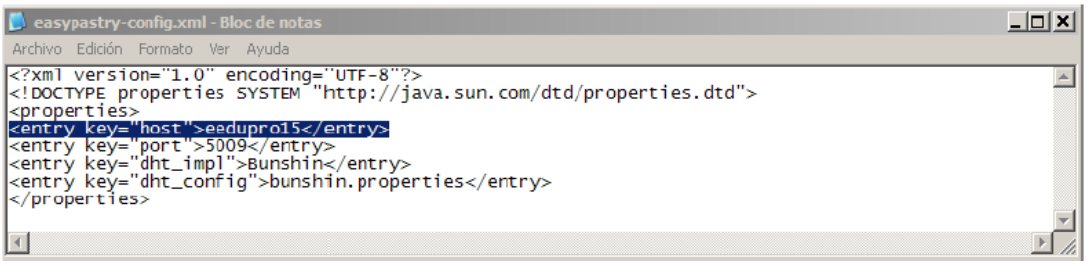


Figura 11: Modificant easypastry-config.xml

Compilar el projecte: situarse al directori eAula i executar la la tasca ant all

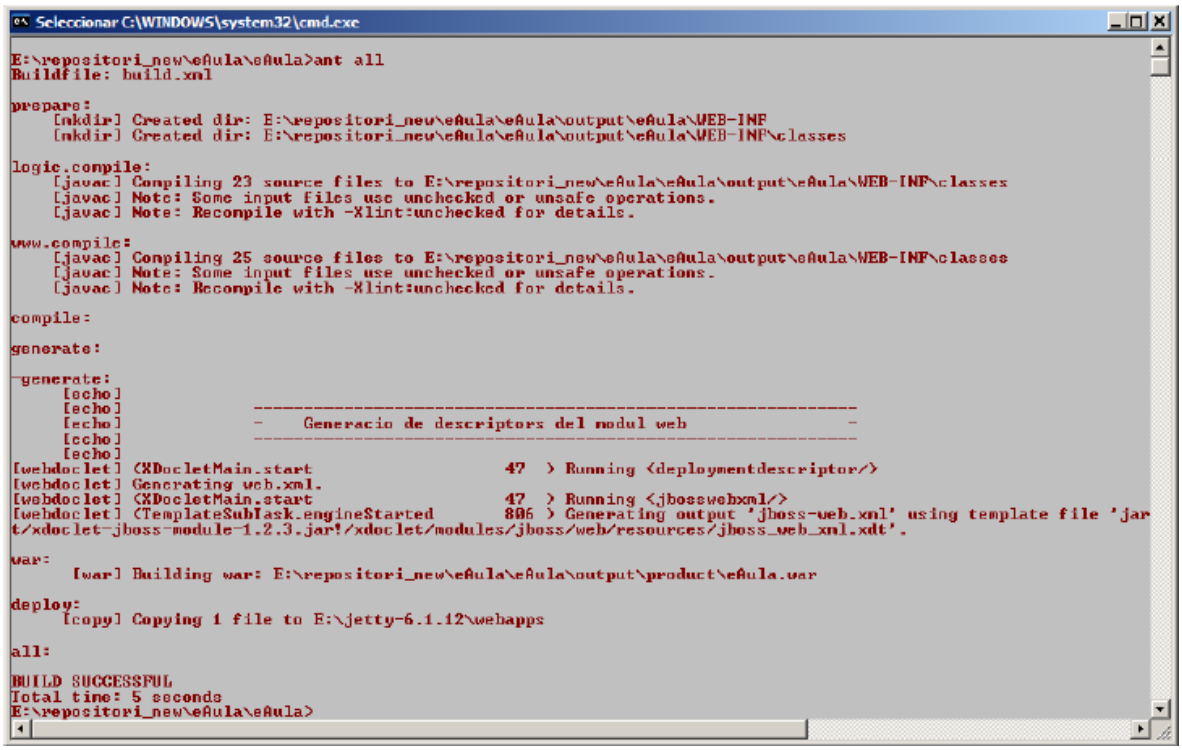


Figura 12: Compilació projecte eAula

Una vegada acabada la compilació (a la consola toca haver sortit el missatge "BUILD SUCCESSFUL") al directori output hi ha d'haver totes les classes compilades així com el directori product amb l'aplicació eAula.war creada.

Arrancar el servidor d'aplicacions Jetty. Situar-se al directory JETTY_HOME\bin (directori home del Jetty) i executar Jetty-Service.exe. Assegurar-se que dins el directori JETTY_HOME\webapps hi ha un fitxer anomenat eAula.war

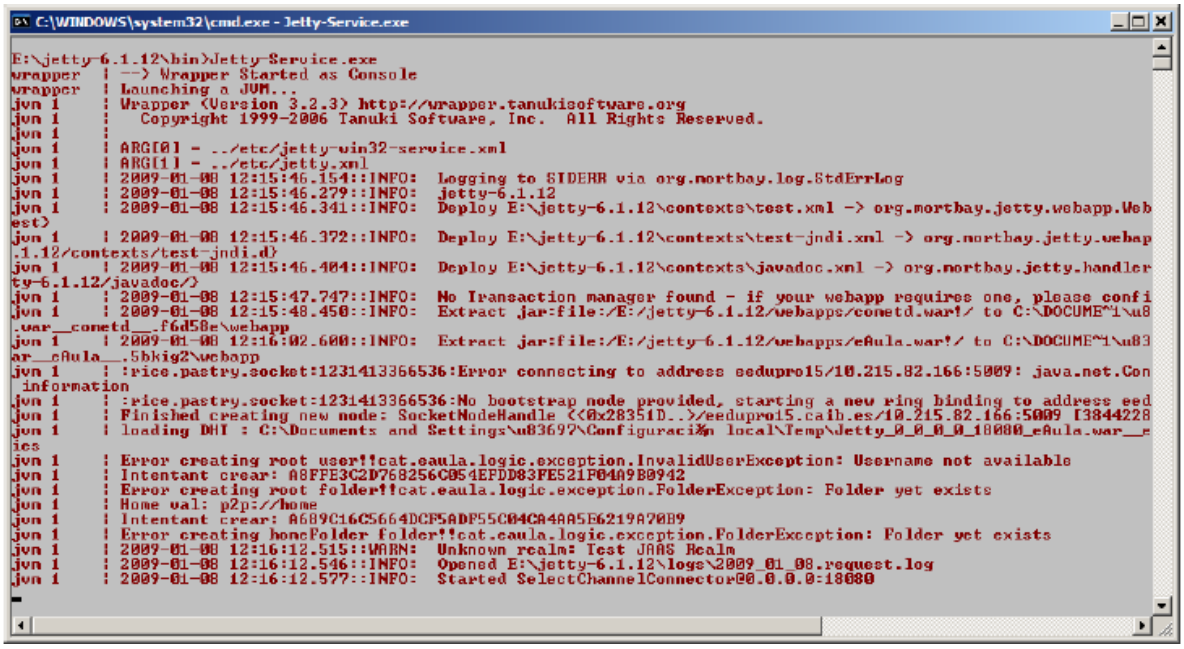


Figura 13: Desplegament eAula al servidor Jetty

10.2.1 Provant eAula: joc de proves.

Com s'ha dit abans al directori *modules* hi figura un subdirectori anomenat *test*. Aquest és un mòdul espacial destinat a fer les proves unitàries de l'aplicació. És una manera de testejar el correcte funcionament dels altres mòduls, especialment el de serveis i accés a les dades. Això ens assegura que si passa el conjunt de proves la instal·lació de eAula anirà correctament.

Per a fer aquests jocs de proves s'utilitzen les llibreries jUnit, (dipositades dins el directori lib/test).

Si es vol executar el joc de proves cal situar-se al directori del mòdul, és a dir dins modules/test i executar la tasca ant run.reports. Aquesta tasca primerament compilarà tota l'aplicació i anirà passant tot una sèrie de proves unitàries com poden ser:

- Inicialització del serveis.
- Connexió a la xarxa KBR. Cal tenir configurat correctament el host de connexió (veure apartat anterior)
- Inicialització del Cast
- Creació d'un usuari nou i emmagatzemar-lo a la DHT.

- Creació un directori nou.
- Compartició d'un fitxer, realitzant l'operació de particionar-lo amb n parts.
- Rejuntar les parts de l'arxiu anterior.
- Inserir índexs per a posteriorment realitzar una consulta.

A l'acabament de la execució de la tasca ant hem de tenir una pantalla semblant a la següent:

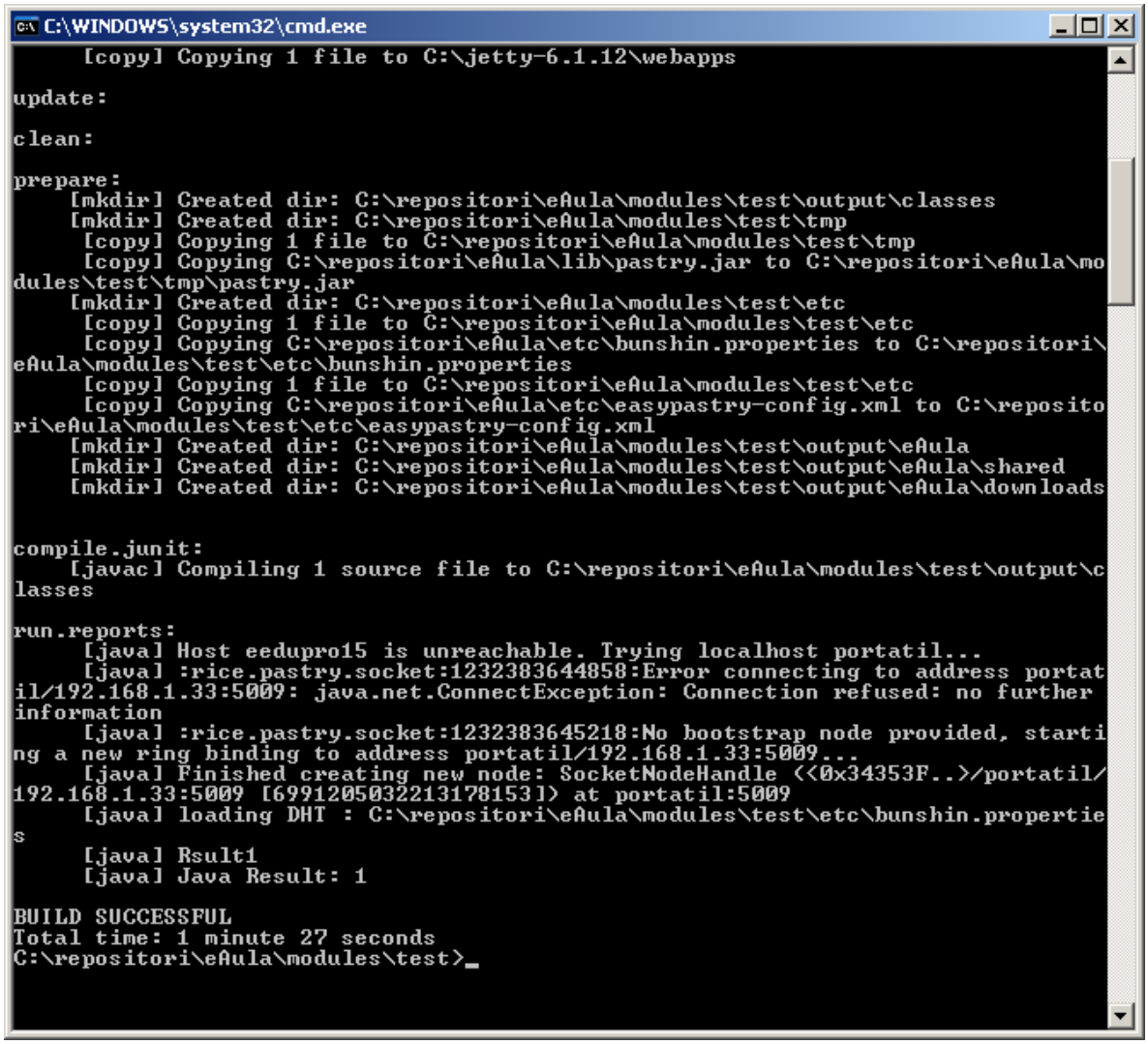


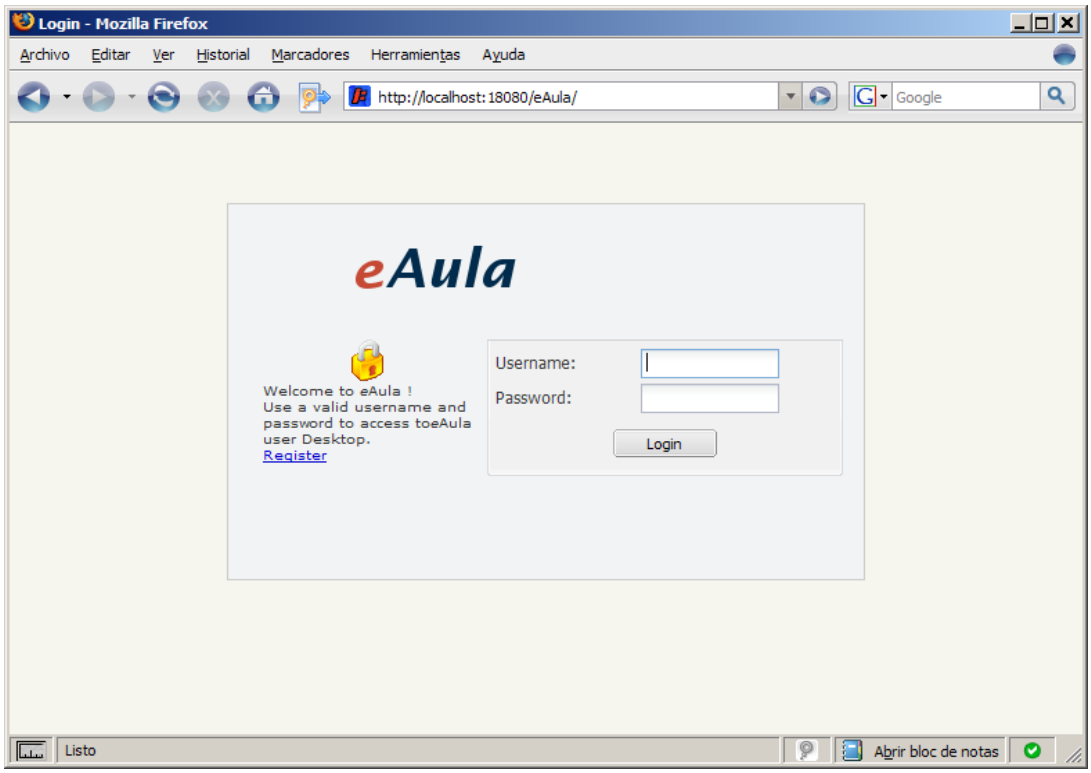
Figura 14: Executant les proves unitàries

10.3 Instal·lant *eAula* a partir de *eAula.war*

Amb la distribució de eAula hi figura l'aplicació ja compilada i construïda (eAula.war). Per a desplegar-la cal copiar-la dins el directori del contenidor on es despleguen les aplicacions (per exemple al Jetty és al directori JETTY_HOME\webapps) . A continuació s'ha d'obrir aquest *eAula.war* (amb un programa que pugui obrir zips o rars) i modificar el fitxer easypastry-config.xml que es troba dins el directori META-INF, posant el nom del host on instal·larem l'aplicació per primer cop. A continuació engegam el servidor d'aplicacions Jetty comprovant que es té accés a la pàgina http://localhost:8080/eAula/index.html

11 Manual d'usuari.

Un cop instal·lat i desplegat correctament eAula al servidor d'aplicacions cal obrir un navegador web i adreçar-se a la pàgina http://localhost:8080/eAula/index.html[5]. La pàgina inicial (figura següent) és la plana per identificar-se com a usuari de l'espai de treball compartit. Per accedir cal inserir un usuari, prèviament registrat, i una paraula de pas.

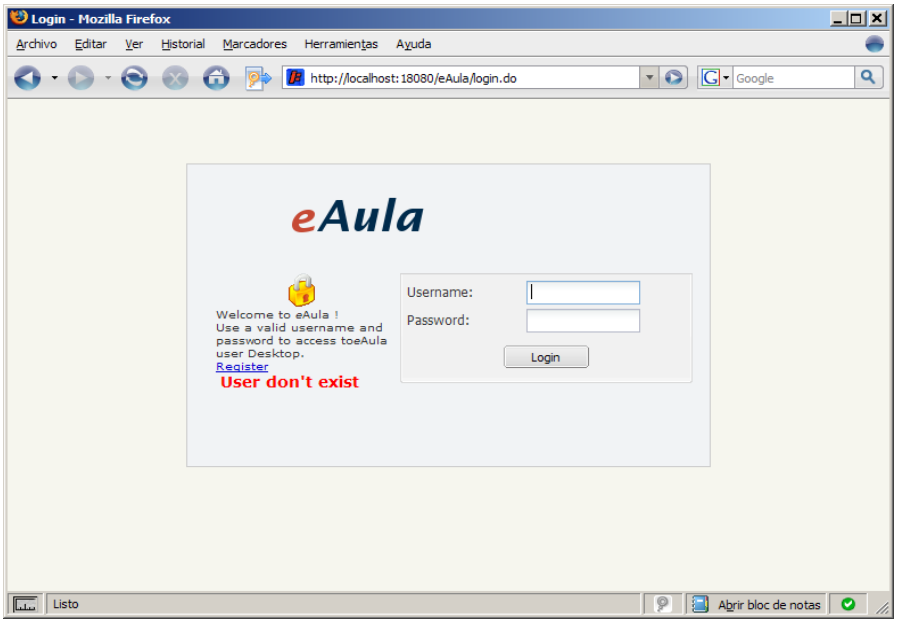


Pantalla 1: Pàgina inicial

Si ens erram a l'hora d'introduir el nom d'usuari o la paraula clau el sistema ens avisa amb un missatge d'error. A la pantalla següent es pot veure de quina manera s'informa a l'usuari el tipus de errada.

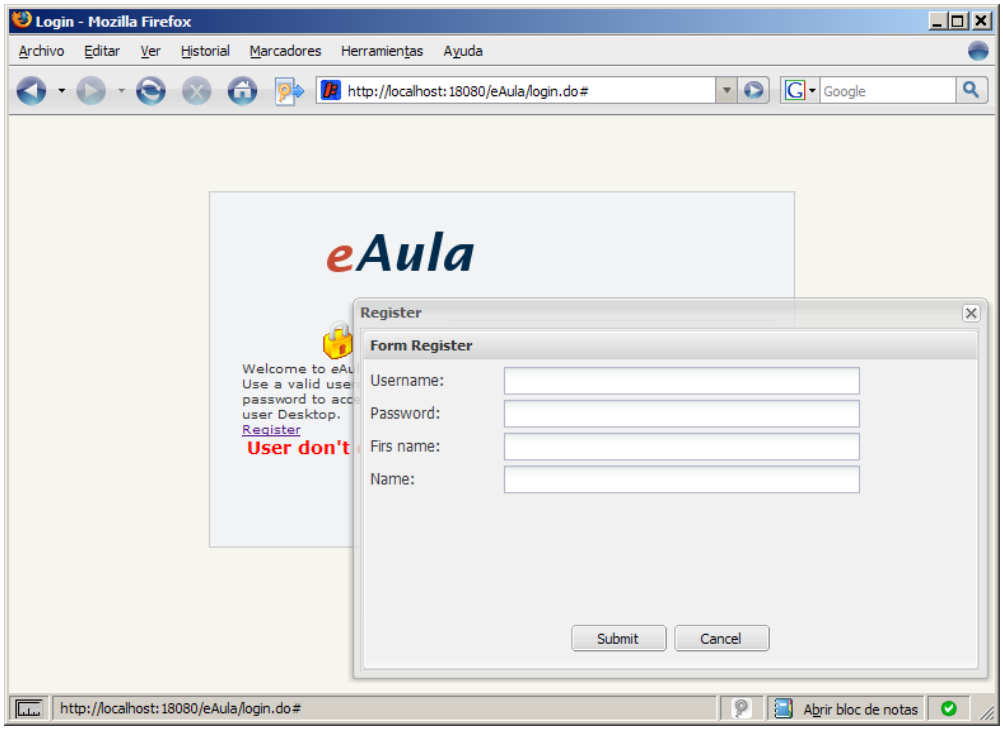
5 Aquesta adreça depèn de la configuració del servidor d'aplicacions. Per defecte, el servidor Jetty es desplega sobre el port 8080.

Per a registrar-se com a usuari nou cal adreçar-se al link “Register” i obrir el formulari per



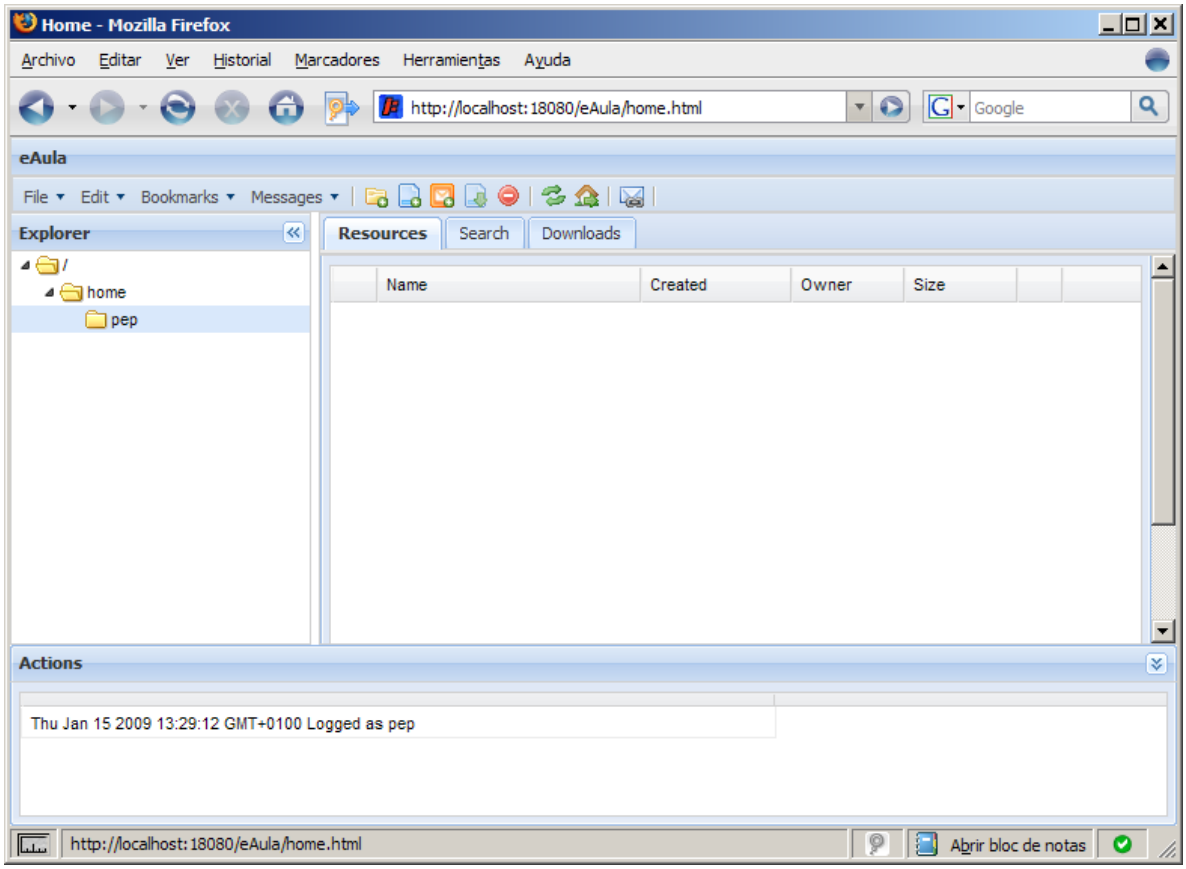
Pantalla 2: Errada d'usuari / password

donar d'alta usuaris nous. L'aparença del formulari per a donar-se d'alta és el representat a la següent captura de pantalla:



Pantalla 3: Formulari enregistrament nou usuari

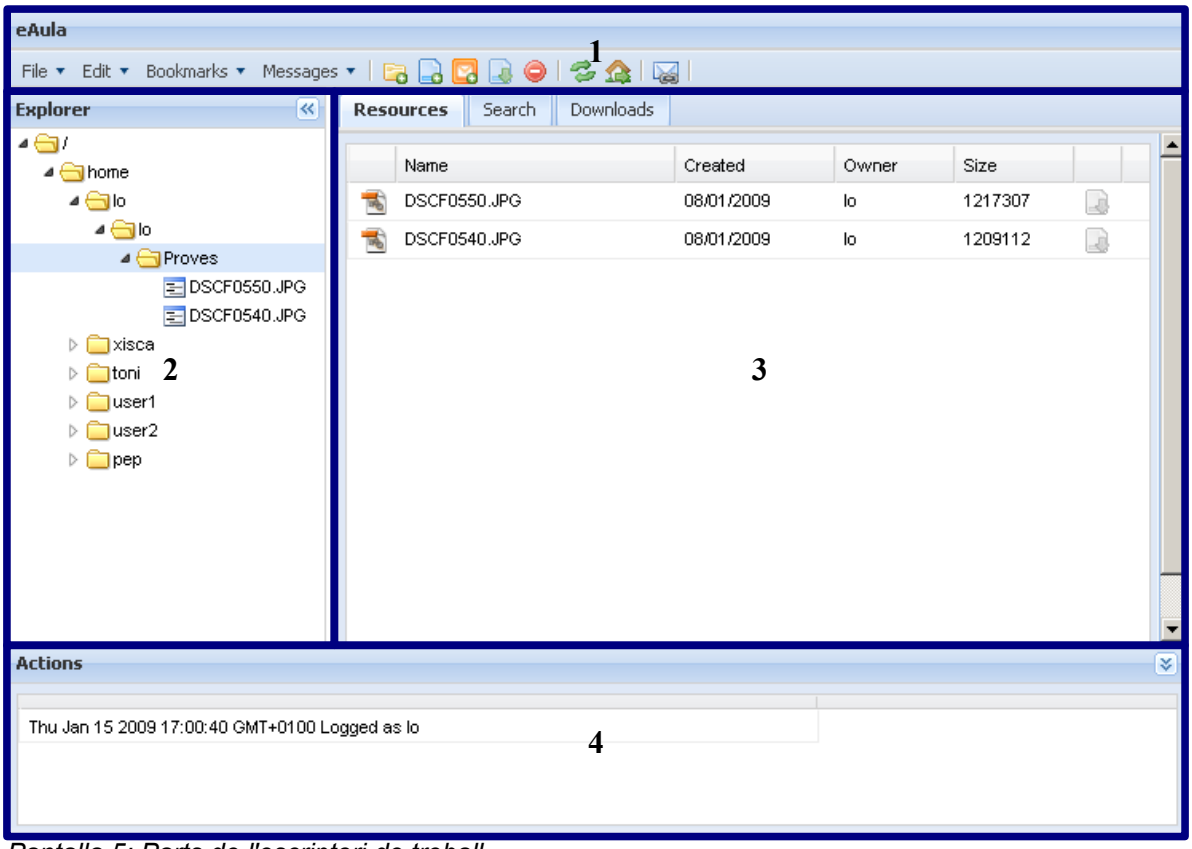
Al formulari d'alta de usuaris cal fixar unes dades bàsiques per identificar d'alguna manera l'usuari. Bàsicament cal inserir el *username*, la paraula clau (password), el nom i els llinatges. Un cop identificar com a usuari o bé registrat de bell nou, ens adrecem al nostre marc de treball: l'escriptori de l'espai compartit.



Pantalla 4: Escriptori espai compartit

Com es pot apreciar a la següent figura, aquesta pantalla principal per a gestionar els recursos compartits es divideix en quatre parts diferenciades:

1. La primera regió (1) es correspon a la zona on es defineixen el títol inicial, els menús i les icones de les eines disponibles per l'usuari

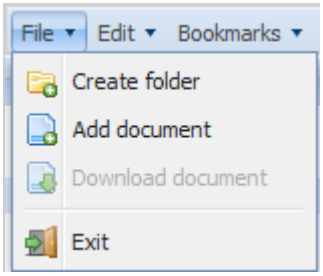


Pantalla 5: Parts de l'escriptori de treball

Els menús i barra d'eines ens permeten executar les tasques per a la gestió d'aquest espai d'intercanvi de documents.

Pantalla 6: Regió 1: Menú i barra d'eines

Les accions principals relacionades amb la gestió de fitxers a l'espai compartit es troben al menú anomenat "File".

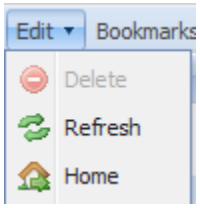


Pantalla 7: Menú 'File'

Sota aquest menú hi podem trobar les següents accions:

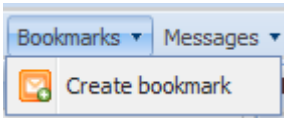
- Crear un directori (*Create folder*): Afegir o crear un directori nou al node seleccionat.
- Afegir un document (*Add document*): Afegir un document nou al directori o node seleccionat.
- Davallar un document (*Download document*): Davallar el document seleccionat i poder visualitzar-ho posteriorment.
- Sortir (*Exit*): Sortir de l'aplicació

Al menú Edit hi trobam funcionalitats per a millorar la gestió dels recursos i l'estructura dels directoris. Les principals funcions d'aquest menú son:

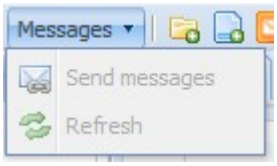


- Esborrar (*Delete*): Esborrar bookmarks.
- Refrescar (*Refresh*): Refrescar, permet actualitzar l'estructura de directoris, fent una petició d'actualització al servidor.
- Home (*Home*): Home, situar-se sobre el node de l'usuari validat.

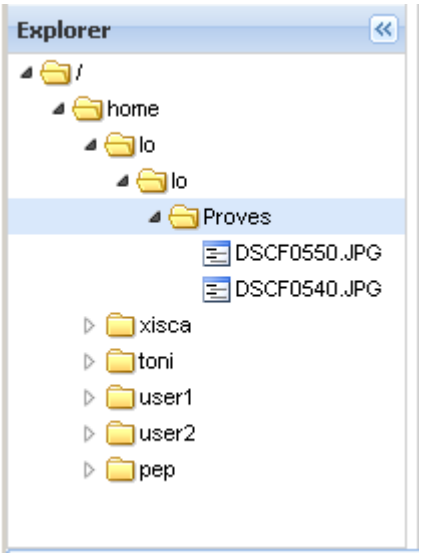
El menú de bookmarks és l'encarregat de mantenir les operacions per a crear els marcador. (*Create bookmark*)



Per últim tenim presència del menú per a enviar i actualitzar els missatges rebuts. Ens permet enviar (*Sende message*) i refrescar els missatges (*Refresch*).

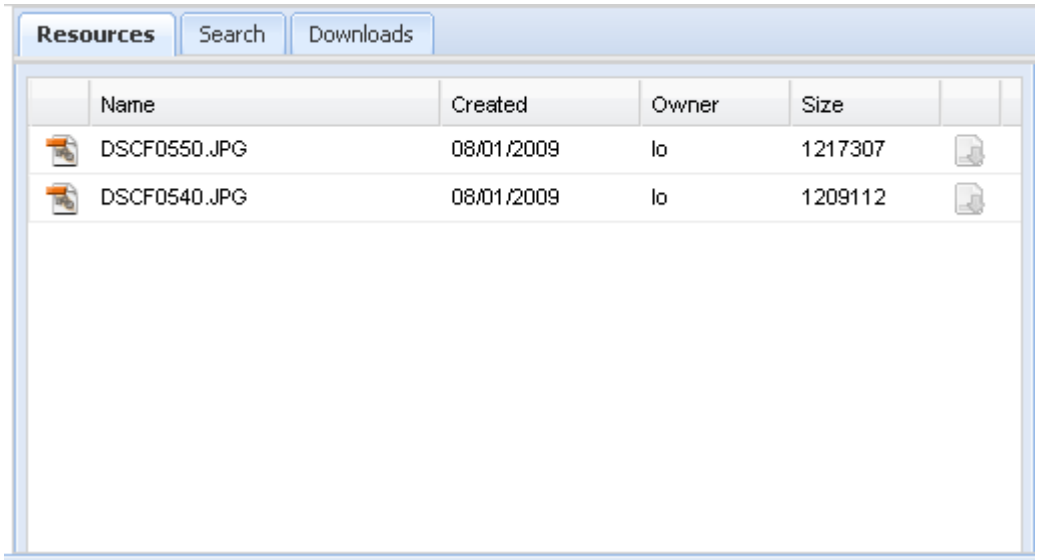


2. La segona regió (2) serà l'encarregada de visualitzar l'estructura de directoris dels nostres recursos compartits. Es visualitzarà en forma d'estructura d'arbre a partir del node o directori arrel (/) on es connectaran els usuaris. Quan ens validam l'aplicació es situa sobre el directori *home* de l'usuari en concret.



Pantalla 8: Regió 2: Explorador

3. La tercera regió (3) es correspon a la zona central de l'escriptori



Pantalla 9: Regió 3: Gestió de recursos

Aquesta zona central de l'escriptori presenta una estructura en forma de pestanyes.

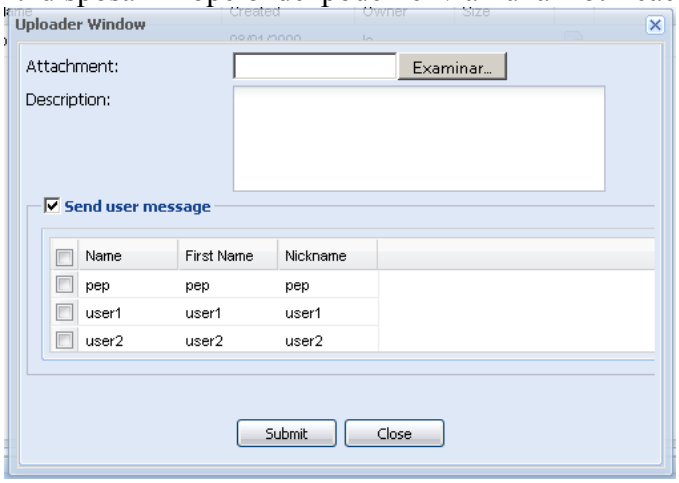
- La primera (seleccionada per defecte) i anomenada “Resources” és la graella on es visualitzen els recursos del directori on estam situats, informant-ne del seu nom, una icona identificadora del tipus de recurs que és (pdf,executable,..) basada en el content-type, la data de creació, l'usuari propietari d'aquest arxiu, la mida del fitxer i un botó indicant si es pot davallar (cas d'estar ubicar a un altre node) o no (cas que ja estigui al node on ens hem connectat) .
- La segona pestanya anomenada “Search” és un panell on hi ha implementat el formulari de recerca de documents i una graella on es visualitzen els documents trobats mitjançant aquesta cerca.
- La tercera pestanya és l'anomenada “Download” i és per indicar l'estat de cada fitxers que hem donat a l'ordre de davallar. Ens mostra de cada recurs la mida total i la mida davallada.

4. La quarta regió està situada al peu de l'escriptori i serveix per a enregistrar qualsevol operació que s'ha dut a terme. Així quan cream un directori nou, enviam un missatge, es connecta un usuari nou a l'aplicació,etc, totes aquestes accions queden impreses a aquest “log” d'operacions.

11.1 Creació/compartició de recursos

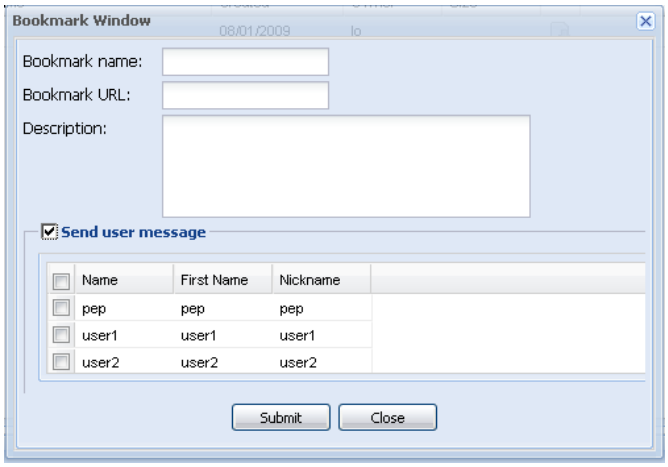
La finestra per a la creació/compartició de fitxers presenta una estructura com la de la figura següent. En ella es pot veure que el primer que hem d'indicar és la ubicació del recurs, podent executar una recerca al disc dur local de l'ordinador de l'usuari . A continuació hi ha la possibilitat de fixar una descripció per aquest recurs (aquesta descripció serà la utilitzada per realitzar les recerques).

Finalment disposam l'opció de poder enviar una notificació als usuaris registrats de la



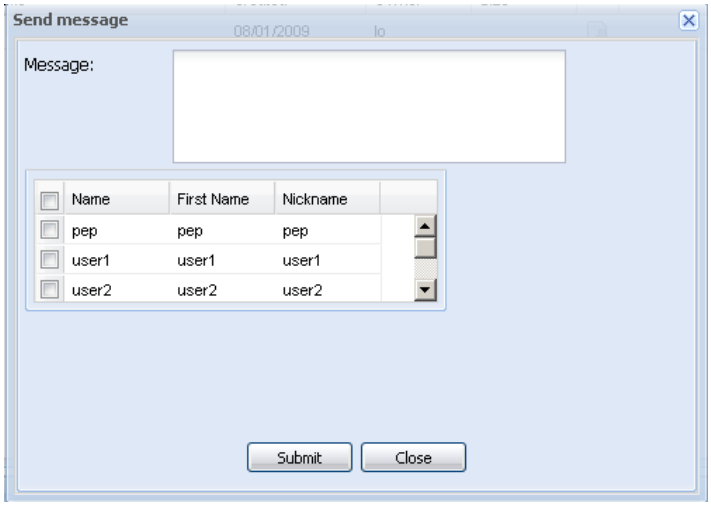
nostra xarxa p2p, indicant el fet que s'ha pujat o compartit un recurs nou.

La pantalla de creació de nous marcadors és similar a l'anterior. Bàsicament hem d'indicar el nom del marcador així com la URL on apunta el marcador. També podem indicar un petita descripció d'aquest bookmark, que serà utilitzada també a l'hora de recercar informació.



Hi ha la possibilitat també de poder enviar notificacions als altres usuaris cada cop que es crea un marcador nou.

L'estructura del formulari per enviar missatges o notificacions als altres usuaris també és molt semblant a les pantalles per crear *bookmarks* o la pantalla per compartir arxius. S'ha d'assenyalar els usuaris destinataris del missatge i escriure el cos del missatge.



Els missatges rebuts sorgeixen de manera síncrona des de dalt de la pantalla tal i com es mostra a la pantalla següent.

12 Conclusions.

Planificar i intentar implementar una aplicació per a la gestió de documents adreçada a un espaï d'intercanvi de tipus "*aula"* és un repte important ja que pot ser un punt de partida de projectes més ambiciosos . I a més a més fer-ho d'una manera totalment descentralitzada sobre una xarxa p2p encara és una ingredient més important que ha fet de eAula un projecte prou peculiar. La motivació inicial d'aquest treball neix d'intentar solucionar el problema de l'arquitectura client/servidor del entorns col·laboratius tipus BSCW.

Es tenia clar des del primer moment que es volia crear una aplicació totalment distribuïda sobre una xarxa peer-to-peer. Per això s'ha vist que les llibreries EasyPastry ens ajuden en molts de casos ja que ens abstreuen de funcionalitats més complexes (Pastry, Bushine,..) i les presenten de manera senzilla per al seu ús. D'aquesta manera EasyPastry ens facilita molt la tasca d'accedir a les DHT, obtenir-ne valors o posar-ne. També ens presenta la part del Cast de manera que ens simplifica l'acció d'enviament i recepció de missatges.
També es tenia clar que per ser una aplicació web ben estructurada s'havia de dividir en capes. S'ha vist que la distribució òptima per a eAula era una estructura en tres capes. Això ha fet més senzilla tan la tasca d'implementació de la lògica de negoci així com la de presentació i ho propiciat una metodologia de treball peculiar on, com s'ha vist, primer s'ha dissenyat l'estructura de la interfície d'usuari i després els mètodes de serveis i classes de negoci necessàries.

Un punt important desenvolupat al llarg d'aquest procés és el fet de tenir ben definides les classes de negoci que determinen el comportament de la lògica. També s'ha experimentat la problemàtica de certs tipus de comportaments, per exemple referències creuades d'objectes (clonació d'objectes) o el problemes derivats d'utilitzar classes que no permetien la serialització. Per tant, aquests dos punts, serialització i possiblitat de clonació, han determinat en cert grau les classes destinades a ser emmagatzemades en DHTs.

En conclusió es pot dir que ha estat un projecte engrescador on s'ha pogut experimentar la problemàtica i els beneficis de les xarxes distribuïdes sobre una KBR, la problemàtica de les DHT i les possibilitats que ens ofereix com a mecanisme de persistència.

13 Glossari

Ajax (*Asynchronous JavaScript And XML*): és una tècnica de desenvolupament web per a crear aplicacions interactives RIA (Rich Internet Applications).

Ant:eina utilitzada a l'hora de crear projectes java per a la realització de tasques mecàniques i repetitives, normalment durant la fase de compilació i construcció de l'aplicació.

Anycast: es una forma de adreçament en la qual la informació es enrutada al millor destí des del punt de vista de la topologia de la xarxa.

Bookmark (marcador): és una localització emmagatzemada d'una pàgina web expressada en forma de URL de forma que pot ser revisada posteriorment.

BSCW(Basic Support for Cooperative Work): és un paquet de software col·laboratiu per a la col·laboració en la web desenvolupat per la Franhofer Society. Suporta pujades de fitxers, notificació d'events i gestió de grups de treball,

Caching (caché o cau): conjunt de dades duplicades d'unes altres originals amb la propietat de que les dades originals son costoses d'accedir, normalment en temps, respecte a la copia cau. Els accessos següents es realitzen a la copia millorar el temps d'accés.
Client-servidor: Arquitectura del programari consistent en que un programa (client) realitza peticions a un altre programa (servidor) que li serveix la resposta.

Content-Type: És una capçalera enviada per un servidor de algun recurs cap al client que descriu el tipus d'informació que el servidor enviarà.

CSS (*Cascading Style Sheets*): És un llenguatge formal utilitzat per la presentació d'un document estructurat escrit en HTML o XML.

Descriptor de desplegament : És un component en les aplicacions J2EE que descriu cóm s'ha de desplegar o implantar una aplicació web.

DHT (Distributed Hash Table) :Mecanismes distribuïts que permeten la localització eficient de dades en sistemes de gran escala a través d'un índex descentralitzat i uniformement repartit entre tots els nodes del sistema.

Diagrames Gannt: és una popular eina gràfica en la gestió de projectes el principal objectiu de la qual és mostrar el temps de dedicació previst de les diferents tasques o activitats al llarg d'un temps total determinat.

DNS: És una base de dades distribuïda i jeràrquica que emmagatzema la informació associada als noms de domini en xarxes d'Internet.

DOM (Document Object Model): és essencialment un model computacional a través del qual els programes o scripts poden accedir i modificar dinàmicament el contingut, estructura i estils dels documents XML i HTML.
EasyPastry: llibreries java destinades a ajudar i facilitar la gestió del mecanismes bàsics de les xarxa distribuïdes peer-to-peer.

Enrutar (**enrutament**): És tracta de la funció de cercar un camí entre tots els possibles en una xarxa de paquets la topologia de la qual presenta una gran

Entitat: És una representació abstracta i conceptual d'una dada i la seva estructura.

Escalabilitat: És la propietat d'un sistema, xarxa o procés, que indica l'habilitat per menejar el creixement continuu del treball de manera fluida i estar en preparació per esdevenir més gran sense pèrdua de qualitats en els serveis ofertats.

Ext Js: son una serie de llibreries JavaScript per a construir aplicacions web interactives utilitzant tècniques som Ajax, DHTML i DOM.

framework: és una estructura de suport definida a través de la qual un altre projecte de software pot ser organitzat i desenvolupat. Sol incloure suport de programes, biblioteques, llibreries per ajudar a desenvolupar i ajuntar els diferents components d'un projecte.

getter i setters: són mètodes d'accés a les classes, la qual cosa significa que generalment són una interfície pública per a canviar membres privats de les classes. S'utilitzen per definir propietats de les classes o objectes.

herència: és la propietat que permet als objectes ser creats a partir d'altres ja existents, obtenint característiques (mètodes i propietats) similars als ja existents. És la relació entre una classe general i una altre classe més específica.

heterogeneïtat ***:***Propietat d'un sistema distribuït que indica que està format per una varietat de diferents xarxes, sistemes operatius, llenguatges de programació o maquinari de l'ordinador o del dispositiu.

indexació: acció de registrar ordenadament informació per elaborar un índex amb la finalitat d'obtenir resultats de forma més ràpida i rellevant en el moment de realitzar una consulta.

javascritp: És un llenguatge de programació interpretat, és a dir no requereix compilació, utilitzat principalment en planes web per a dotar-la d'interactivitat. La seva sintaxis és semblant al llenguatge C i al llenguatge Java.

jetty: És un servidor HTTP i contenidor de Servlets escrit en Java. Se publica com un projecte de software lliure baix la llicència de Apache 2.0. Degut al seva petita mida, Jetty se complementa per oferir serveis web en una aplicació Java incrustada.

JSON(JavaScript Object Notation): és un format lleuger per l'intercanvi de dades. JSON és un subconjunt de la notació literal d'objectes JavaScript que no requereix l'ús de XML.

JSP(Java Server Pages): és una tecnologia de Java que permet generar contingut dinàmic per a web, en forma de documents HTML, XML o d'altres tipus. Permet la utilització de codi Java mitjançant scripts. A més a més es possible utilitzar algunes accions JSP predefinides mitjançant etiquetes (TagLibs).

JSTL (**JavaServer Pages Standard Tag Library**): és un component de Java EE. Extén les conegudes JSP proporcionant quatre llibreries d'etiquetes (TagLibs) utilitzades en el desenvolupament de pàgines web dinàmiques.

Junit: És un conjunt de classes i llibreries (framework) que permet realitzar l'execució de classes i programes Java de manera controlada per a poder avaluar si el funcionament de cada un del mètodes de la classe es comporta com s'espera (prove unitàries).
KBR (**Key based routing**): És un mètode per a cercar el host més pròxim per intercanviar dades, d'acord a alguna mètrica definida.

lògica de negoci: és la part d'un sistema (en arquitectura de software) que s'encarrega de les tasques relacionades amb els processos de negoci i tota casta de processament que es realitza darrera de l'aplicació visible a l'usuari.
marcador: Veure **bookmark**.

Meta-Inf: És un directori que presenten les aplicacions web desenvolupades en Java i incrustades en un WAR, destinat a contenir la meta informació sobre els fitxers continguts a l'aplicació.

middleware: és un software de conectivitat que ofereix un conjunt de serveis que fan possible el funcionament de les aplicacions distribuïdes sobre plataformes heterogènies. Funciona com a una capa d'abstracció de software distribuït que se sitúa entre les capes d'aplicacions i les inferiors.

multicast (multidifusió): és l'enviament de la informació en una xarxa a múltiples destinataris simultàniament, utilitzant l'estratègia més eficient .

ONO: *Plugin* destinat a resoldre el problema d'escollit el millor node per fer la descàrrega en xarxes p2p. ONO fa enginyeria inversa sobre xarxes distribuïdes de continguts per a obtenir una resolució DNS dels nodes que tenen el fitxers o les parts, ordenar per latència les llistes i arriben a millorar un rendiment fins a un 30%.

overlay network : Xarxa a nivell d'aplicació que formen els nodes d'uns sistema o aplicació d'igual a igual. Aquesta xarxa funciona sobre la xarxa física que connecta els nodes.

P2P: Veure peer-to-peer

Pastry : és un sistema genèric de localització d'objectes *peer-to-peer* i enrutador descentralitzat basat en enrutament de claus sobre xarxes p2p.

Peer-to-peer : Tipus d'arquitectura distribuïda que es caracteritza pel fet que tots els nodes que formen el sistema o aplicació tenen les mateixes capacitats i responsabilitats i on tota la comunicació és simètrica.

Ping: utilitat que comprova l'estat de la connexió en un o varis equips remots mitjançant paquets de sol·licitud de eco i resposta d'eco per a determinar si un sistema és accessible en una xarxa. Molt de pics s'utilitza per a mesurar el temps de latència entre dos punts remots.

POO (Programació orientada a objectes): És un paradigma de programació que utilitza objectes i classes d'objectes així com les sever interaccions per a dissenyar aplicacions i programes software. Està basada en vàries tècniques com poden ser herència, modularitat, polimorfisme i encapsulat.

RIA (***R**ich **I**nternet **A**pplications*): és un nou tipus d'aplicacions amb més avantatges que les tradicionals aplicacions web caracteritzades per la presencia d'elements típics de les aplicacions d'escriptori.
Scribe *m* : Sistema de comunicació de grup i notificació d'events en una xarxa peer-to-peer que utilitza Pastry per implemetar-ho.

Serialització: procés de codificació d'un objecte e un mitjà d'emmagatzemament amb la finalitat de ser transmés a través d'una conneció de xarxa com un serie de bytes .

Servlet: objectes Java que s'executen dins el context d'un contenidor de servlets i extenen la seva funcionalitat. L'ús més comú dels servlets és generar pàgines web de forma dinàmica a partir dels paràmetres d'entrada.

setter: Veure Getter i Setter

Sistema distribuït: Col·lecció d'ordinadors autònoms enllaçats per una xarxa d'ordinadors i suportats per un programari que fa que la col·lecció actuï com un servei integrat.

SOA (Service Oriented Architecture): és un concepte d'arquitectuda del software que defineix la utilització de serveis per a donar suport als requisits del negoci.

Taula hash: és una estructura de dades que associa claus amb valors. La operació principal que suporta de manera eficient és la cerca: permet l'accés ràpid als elements emmagatzemats a partir d'una clau generada. Funciona transformant la clau a un hash utilitzat per localitzar el valor desitjat.

Topologia: Maneres d'internacionalitzar-se que tenen els components d'un sistema distribuït en funció dels fluxos d'informació que intercanvien.

TIC (**tecnologías de la información y la comunicación**): Són un conjunt de serveis, xarxes, software i dispositius que tenen com a finalitat la millora de la qualitat de la vida de les persones dins un entorn i que se integren a un sistema d'informació interconectat i complementari.

UI (User Interface): És un mitjà amb el qual l'usuari es pot comunicar amb una màquina, un equip o computadora, i compren tots els punts de contacte entre usuari i equip, normalment fàcils d'entendre i accionar.

UML (Unified Modeling Language): És un llenguatge de modelat de sistemes de software utilitzat de manera gràfica per visualitzar, especificar, construir i documentar un sistema.

URL (*Uniform Resource Locator*): És un localitzador uniforme de recursos. És una seqüència de caràcters, d'acord a una format estàndard, que s'utilitza per a nombrar recursos com documents o imatges a Internet per a la seva localització

URL connection: vincle (petició / resposta) entre el un computador que executa una aplicació i un computador que ofereix un recurs. Aquest vincle s'estableix mitjançant l'ús d'una URL del recurs.

web.xml : Verure descriptor de desplegament

widgets : A l'àmbit de la programació gràfica un widget , també conegut com un control o aparell, és un component gràfic o de control amb el qual l'usuari interactúa com poden ser finestres, barra de eines, capces de texte.

Xdoclet: És un motor de codi obert per el llenguatge de programació Java. La seva funció és generar codi i fitxers de desplegaments (meta fitxers).

XmlHttpRequest: interfície utilitzada per a realitzar peticions HTTP i HTTPS a servidors Web. La interfície es presenta com una classe o objecte, bàsicament en Javascript, de la que una aplicació client pot generar tantes instàncies com necessiti per manejar el diàleg amb el servidor.

14 Bibliografia

Tanenbaum A.; Steen, M. (2007). "*Distributed Systems: Principles and Paradigms"* , 2/E. Prentice Hall.

Justin Gehtland, Ben Galbraith, Dion Almaer: "*Pragmatic Ajax. A Web 2.0 Primer*"

Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides : "*Design Patterns: Elements of Reusable Object-Oriented Software*", Addison-Wesley

Martin Fowler :"*UML Distilled: A Brief Guide to the Standard Object Modeling Language*" 3/E. Addison-Wesley

Martin Fowler: "*Patterns of Enterprise Application Architecture*" Addison-Wesley

Thomas Erl: "*Service-Oriented Architecture (SOA): Concepts, Technology, and Design*",Prentice Hall

Dave Crane, Eric Pascarello, Darren James :"*Ajax in Action*", O'Really

Shea Frederick, Colin Ramsay, Steve 'Cutter' Blades : "*Learning Ext JS*", Paperback

Rubén Mondéjar, Pedro García, Carles Pairot: "B*unshin: DHT para aplicaciones distribuidas***"**

Gregorio Jiménez Valverde, Eva Núñez Cruz, Anna Llitjós Viza: "*Synergeia, un entorno telemático cooperativo en el área de ciencias*"

P.A. Felber, E.W. Biersack, L. Garc´es-Erice, K.W. Ross, G. Urvoy-Keller: "*Data Indexing and Querying in DHT Peer-to-Peer Networks*"

Matthew Harren, Joseph M. Hellerstein, Ryan Huebsch: "C*omplex Queries in DHT-based Peer-to-Peer Networks*"

Luis E. Mendoza : "*Estudio de Tecnologías Middleware para Sistemas Peer-to-Peer*"

Joan Manuel Marquès i Puig, Xavier Vilajosana i Guillén,Pedro A. García López: "*Arquitectures, paradigmes i aplicacions dels sistemes distribuïts*", UOC

Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica: "Wide-area cooperative storage with CFS (Cooperative File System) "

David R. Choffnes, Fabián E. Bustamante:"*Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems*", Proc. of ACM SIGCOMM 2008, August 2008.

EasyPastry Homepage : http://ast-deim.urv.cat/easypastry/
BSCW: http://public.bscw.de/
Gnutella Homepage: http://www.gnutella.com
Napster Homepage: http://www.napster.com
Akamai Homepage: http://www.akamai.com
Groove Homepage: http://www.groove.net
Jquery Homepage: http://jquery.com/
Ext Js Homepage : http://extjs.com/
Azureus Homepage: http://azureus.sourceforge.net/plugin_details.php?plugin=ono