Memòria del Treball Final de Carrera:

Disseny i implementació en *Java* d'un sistema segur de descàrrega anònima de fitxers format per aplicacions *web* basades en *XML* i amb autenticació del client per part d'un tercer mitjançant una *PKI*

presentat per:

Ramon Costa i Sala

per optar al títol d'ETIS a la UOC

Lliurada el 16 de gener de 2009
Consultor: Carlos Ares Angulo

RESUM

En aquest TFC s'ha proposat dissenyar i implementar en llenguatge Java un sistema segur de descàrrega anònima de fitxers. Per fer aquesta tasca es proposa un conjunt d'aplicacions web que es s'intercanviaran dades en format XML de forma segura, sempre sota protocol HTTPS i mantenint la integritat, autenticitat i autenticació de les parts amb l'ajut d'un PKI:

El client farà la petició a un proveïdor de serveis de forma anònima, qui li retornarà un document XML de petició i la signatura digital d'aquest amb certificat per mantenir-ne la integritat. En ell, hi constarà l'instant en què s'ha realitzat la petició, que ha de caure dins uns límits establerts.

Per a què se li autoritzi l'accés al recurs, el client s'haurà d'autenticar enfront un tercer i demostrar que és ell qui ha fet la petició dins l'horari establert. Per això, signarà digitalment amb el seu propi certificat el document de petició rebut i trametrà per un canal segur a un tercer, el proveïdor d'identitat, la petició i les dues signatures encapsulades dins un missatge SOAP.

Quan el servei del proveïdor d'identitat rebi em missatge, comprovarà que les dues signatures corresponen a la mateixa petició i que els dos certificats són correctes i no revocats: d'ells n'extraurà la identitat del proveïdor de serveis i del client. Amb aquesta informació i llegint de la petició el recurs demanat i l'hora en que fou emesa, podrà comprovar en la seva base de dades si el client acompleix les condicions. En cas afirmatiu, generarà un document XML d'autorització positiva d'accés al recurs que contindrà la petició original, la signarà digitalment i retornarà pel canal SSL al client. Cas contrari, farà constar en l'autorització negativa el motiu de la denegació.

En rebre el resultat signat de l'autorització, el client, sempre dins la mateixa sessió que havia establert amb el proveïdor de serveis per evitar-ne manipulacions, el retornarà a aquest.

En rebre l'autorització signada, el proveïdor de serveis comprovarà la signatura i el certificat del proveïdor d'identitat que inclou i, si tot es correcte, autoritzarà la descàrrega. Cas contrari, la denegarà mostrant el motiu.

El correcte funcionament del sistema també requerirà que s'habiliti un administrador per gestionar la BBDD on el proveïdor d'identitat hi guarda els permisos. Aquesta gestió s'ha de fer de forma remota a través d'un canal HTTPS segur i prèvia autenticació de l'administrador enfront del servidor.

ÍNDEX

CAPÍTOL I : INTRODUCCIÓ

1.1.- Objectius i justificació

L'objectiu principal d'aquest TFC és dissenyar i implementar un sistema que permeti a un client sol·licitar un servei a un proveïdor de forma anònima, i que només se li autoritzi l'accés al recurs si pot acreditar davant un tercer (el proveïdor d'identitat) la seva identitat i que acompleix unes condicions predeterminades. Totes les comunicacions necessàries es faran a través de la xarxa oberta *Internet*, pel què caldrà assegurar la integritat i privacitat de les dades, i en certs casos autenticar els actors.

El client sol·licitarà el recurs al proveïdor remot amb un navegador *web* a través d'un canal segur: el servidor s'haurà d'autenticar davant el client, mentre que aquest darrer mantindrà el seu anonimat. El servidor remetrà al client un document de petició

L'autorització de l'accés al recurs es farà mitjançant un tercer, el proveïdor d'identitat: serà l'encarregat d'autenticar la identitat del client, l'autenticitat de la petició al servidor i que la petició correspon al client. Fetes aquestes verificacions, podrà autoritzar la concessió del recurs en base a les dades i condicionants que li hagi indicat el proveïdor del serveis.

Concretament, el proveïdor de serveis tindrà autoritzats certs recursos a determinats clients, que els hauran de sol·licitar dins un rang horari prefixat. El proveïdor d'identitat nomenarà un administrador que serà responsable de gravar i emmagatzemar aquesta informació (mitjançant una aplicació *web* que ha d'executar remotament des d'un navegador) en base a la qual el proveïdor generarà les autoritzacions a les peticions dels clients.

Totes aquestes transaccions s'hauran de dur a terme a través d'un medi insegur com és *Internet*: per assegurar la privacitat caldrà utilitzar canals segurs i xifrat de les dades sensibles transmeses. Caldrà implementar aquestes funcionalitats amb tecnologies *web* que el client pugui accedir amb algun dels navegadors àmpliament coneguts i amb un nivell de coneixements bàsic.

La petició d'autorització del servei es farà de forma gairebé automatitzada des del navegador del client amb l'execució local d'un component que li haurà descarregat el proveïdor de servei. Aquest component s'encarregarà de rebre les dades de la sol·licitud degudament autenticades, generar les dades d'autenticació del client, enviar-ho tot al proveïdor d'identitat i rebre d'aquest la resolució d'autorització, també autenticada. Aquest intercanvi entre client i proveïdor d'identitat caldrà fer-lo mitjançant el protocol SOAP. Finalment, el component permetrà lliurar l'autorització rebuda al proveïdor de servei que, cas que sigui positiva, haurà de permetre l'accés al recurs sol·licitat.

Com a suport per assolir els nivells desitjables de privacitat i una certesa raonable en l'autenticació i autorització de les parts, s'utilitzarà la infraestructura de clau pública (*PKI*): en concret, es basarà en l'ús de certificats digitals X.509v3 emesos per autoritats de certificació de confiança: aquests permetran signar les dades trameses entre els diferents autors. També es contempla la possibilitat d'utilitzar segells de temps mitjançant un prestador TSA d'aquest servei que garanteixi independentment que la sol·licitud del client s'ha fet en l'horari permès. Malauradament, no he tingut temps d'implementar aquest darrer punt al lliurament final

Un requeriment que no consta a l'enunciat del treball però que m'agradaria satisfer és que, amb l'excepció del sistema operatiu *Windows*, la resta de programari extern a emprar fos lliure.

La construcció d'un sistema d'aquesta envergadura és tot un repte per a un estudiant de darrer any d'ETIS. Aquest TFC requereix utilitzar i aplicar amb un objectiu únic una gran quantitat de coneixements i competències apresos de forma dispersa en una àmplia varietat d'assignatures de la carrera.

Però també exigeix buscar i aprendre de forma autònoma noves tecnologies que no s'han vist prèviament i que calen per poder abordar i solucionar els nous problemes que es plantegen al llarg del treball. De fet, és una aproximació prou efectiva a la problemàtica que serà el pa de cada dia d'un professional de la informàtica, objectiu de formació primordial d'aquests estudis.

Personalment, el considero una magnífica pedra de toc per demostrar als professors, però sobretot a mi mateix, que he assolit tant els coneixements i les competències professionals d'un ETIS, així com l'habilitat d'ampliar aquestos mitjançant la cerca, obtenció i ús de nous recursos de forma gairebé autosuficient. També servirà per reforçar la capacitat d'afrontar nous problemes i la generació d'estratègies per resoldre'ls, enriquint la personalitat, l'actitud i l'autoestima. I finalment, totes les noves tecnologies i metodologies que s'hi apliquin serviran per enriquir el bagatge de coneixement i experiència del nou titulat que haurà d'obrir-se el seu propi camí.

1.2.- Punt de partida: experiència prèvia

Professionalment no em dedico a la informàtica, per la qual cosa els meus coneixements i aptituds estan basats en l'experiència com a usuari i en els adquirits en cursar les assignatures d'ETIS. Analitzats els requeriments del sistema a desenvolupar, crec que les experiències prèvies que puc aportar com a punt de partida tecnològic són:

- Fonaments de programació orientada a l'objecte en llenguatge Java: *Fonaments de programació II* (convalidada a *Programació orientada a l'objecte*).
- Bases de disseny de programari orientat a l'objecte en *Enginyeria del programari*.
- Tècniques de transferència d'informació per canals segurs a *Seguretat en xarxes de computadors*.
- Infraestructura de clau pública, certificats i signatures digitals a *Seguretat en xarxes de computadors* i *Criptografia*.
- Generació i verificació de signatures digitals programats amb Java (estàndard *Dsig* sobre documents *XML*) en *Criptografia*.
- Creació, accés i gestió bàsiques d'un sistemes de bases de dades propietari *Informix* amb pont JDBC en *Bases de dades I* i *II*.
- Fonaments de programació de pàgines web estàtiques a *Multimèdia i Comunicació*.

En aquest punt sembla clar que la tasca de busca i aprenentatge més important i que em requerirà més temps i esforç serà la selecció i l'aprenentatge de les tecnologies *web* dinàmiques que permetin acomplir els requisits del TFC en la generació, validació i enviament dels diversos fluxos de dades que hauran d'intercanviar els navegadors del client i l'administrador amb els servidors dels proveïdors.

La configuració i posada en marxa d'aquests servidors també serà un punt novedós, alhora que crític. El disseny d'interfícies d'usuari prou amigables i informatives en serà un valor afegit.

També em caldrà aprendre els fonaments de SOAP per poder implementar i utilitzar aquest protocol en un servidor que l'admeti.

1.3- Enfocament (1): visió global del sistema

Després de meditar a fons com hauria de funcionar el sistema i de llegir els diversos documents penjats a l'aula (molt especialment les consideracions, força detallades), he optat per dividir el sistema en quatre subsistemes, cadascun am funcionalitats ben definides:

- El proveïdor de servei, que està ubicat en un servidor que conté diversos recursos i que és capaç de permetre-hi l'accés segur i selectiu a un client anònim que prèviament s'ha autenticat amb el proveïdor d'identitat. Cal que protegeixi la petició mitjançant una signatura, enviï ambdues al client i finalment rebi i verifiqui l'autorització signada pel proveïdor d'identitat, en base a la qual permetrà l'accés al recurs.

- El client, que bàsicament utilitzarà un navegador web per comunicar-se d'una banda de forma anònima amb el proveïdor de serveis i fer-li la petició. D'altra, executarà un component (basat en un *applet* i *javascript*) que li permetrà rebre petició i signatura del proveïdor, fer la seva pròpia signatura, remetre els tres documents al proveïdor d'identitat i rebre d'aquest l'autorització signada, que haurà de retransmetre al proveïdor de serveis per poder accedir al recurs.
- El proveïdor d'identitat, que conté tota la informació sobre el proveïdor de serveis, els clients, i els diversos permisos d'accés a cada servei que pot tenir cadascun. Bàsicament consisteix en un servidor que és capaç de rebre una petició SOAP d'un client que encapsula la petició al proveïdor de serveis i les signatures que d'aquesta han fet client i proveïdor, verificar les signatures i els permisos d'accés, i retornar una resposta SOAP que contingui una autorització degudament signada amb l'acceptació o denegació del servei.
- Un gestor del proveïdor d'identitat, que permet a un administrador gravar les dades del proveïdor de servei, dels seus clients i dels permisos d'accés que cadascun te assignats, de manera que puguin ser llegides per l'aplicació del proveïdor de servei a l'efecte d'emetre la seva decisió. Fóra bo que aquest subsistema residís en una base de dades a la qual pogués accedir el servidor del proveïdor d'identitat per a què llurs aplicacions *web* hi poguessin fer les consultes pertinents.

1.4.- Enfocament (2): anàlisi dels fluxos d'informació

En aquest punt, analitzo els fluxos d'informació que s'han de trametre entre els diferents subsistemes i els dos actors que he identificat: el client i l'administrador del proveïdor d'identitat.

1.4.1.- Administrador - Gestor de dades

L'administrador ha de poder connectar-se remotament al gestor per poder dipositar les dades del proveïdor de serveis, dels clients i de llurs permisos, per a què puguin ser recuperades pel proveïdor d'identitat en el moment que aquest rebi la petició. La manera ideal de fer aquesta administració és disposar d'un servidor de bases de dades amb accés remot que pugui rebre ordres de transaccions (en format estàndard SQL) a través d'una aplicació web ubicada al servidor del proveïdor d'identitat i un pont JDBC.

L'administrador haurà de poder accedir a aquesta aplicació web de forma remota a través del seu navegador, el que requerirà disposar de mecanismes de seguretat que permetin la doble autenticació i el xifrat de les dades (privacitat): el canal més segur sembla una connexió HTTPS amb doble intercanvi de certificats, malgrat que finalment he optat per autenticació sota SSL.

1.4.2.- Proveïdor d'identitat - Gestor de dades

Les dades introduïdes per l'administrador resideixen en una base de dades remota, a la qual hi haurà de poder accedir l'aplicació *web* del proveïdor d'identitat per verificar els permisos de l'autorització que ha d'emetre. Per tant, a aquesta també li caldrà un pont JDBC per poder accedir a les taules guardades al SGBD.

1.4.3.- Client - Proveïdor de servei.

El client s'ha de comunicar de forma anònima i segura amb el servidor del proveïdor, de qui ha de confirmar la identitat, per fer-li la sol·licitud del servei. El canal adient sembla una connexió HTTPS amb només presentació del certificat per part del servidor. Aquesta connexió ha de consistir en una única sessió, que ha d'estar oberta tot el temps que dura la gestió de la petició i la concessió de permís per part del sistema.

Un cop el servidor s'ha autenticat, el client sol·licita el servei (mitjançant la URL específica) i rep la petició (penso en un document XML), la signatura *detached* del document per part del proveïdor i un segell de temps d'un tercer (que finalment no he pogut implementar). Alhora, se li descarrega un *applet* que s'executarà al navegador del client i serà el responsable de generar la seva signatura i fer els tràmits per obtenir el permís del proveïdor d'identitat.

Un cop el client ha obtingut els permisos (un document XML d'autorització signat pel proveïdor d'identitat), els retorna al proveïdor de serveis mitjançant un formulari *javascript* a través del canal ja establert. El proveïdor de serveis verifica el document d'autorització i, cas que sigui positiva, posa el servei a disposició del client.

1.4.4.- Client - Proveïdor d'identitat

En cop l'*applet* disposa de la petició i les dues signatures, ha de contactar amb el servidor del proveïdor i trametre-li els tres documents. El mecanisme ideal és el protocol SOAP sobre un canal HTTS que permeti enviar al servei corresponent del proveïdor d'identitat un missatge de petició encapsulant els tres documents i rebre'n el missatge de resposta amb l'autorització signada.

1.4.5.- Altres consideracions

Hem de tenir en compte que s'han d'establir altres comunicacions secundàries:

- Entre el proveïdor de serveis i les autoritats de certificació del proveïdor d'identitat per poder descarregar les CRL o bé establir el protocol OCSP.
- Entre el proveïdor d'identitat i les autoritats de certificació del client i del proveïdor de servei per poder descarregar les CRL o bé establir el protocol OCSP.

A més, si s'hagués implementat el segellat de temps per part de la TSA, també caldria considerar:

- Entre el proveïdor de servei i l'autoritat de segells de temps, que ha de segellar la signatura abans d'enviar-la al client.

- Entre el proveïdor d'identitat i el servei de validació extern de l'autoritat de segellat de temps.

Finalment, és important recordar que la tramesa de documents tant sensibles com les signatures a través d'Internet pot portar greus problemes de codificació. Per això, seria convenient que tots els documents XML (incloses les signatures) es transmetin prèviament codificats en Base64.

CAPÍTOL II : FONAMENTS TECNOLÒGICS

Una correcta planificació del projecte implica que prèviament s'han entès i interioritzat tots els seus requeriments, que es te una idea de com s'ha de distribuir el sistema en subsistemes encarregats d'executar blocs de tasques, i que s'albira com s'han de comunicar aquests subsistemes i quina informació s'han d'intercanviar. Aquesta anàlisi s'ha fet al capítol anterior.

A partir d'aquest punt, cal estudiar quines tecnologies poden permetre dur a terme el seu funcionament, triar-ne les més adients en base a diversos motius (capacitats, *performance*, simplicitat, econòmics) i a partir d'aquest punt fer un disseny del sistema previ a la implementació final.

Desenvoluparé doncs en aquest capítol l'anàlisi i tria de les solucions tecnològiques que em semblen les més idònies per dur a terme el projecte. Vull ressaltar que, amb l'excepció de la PKI, les signatures XML i les bases de dades, mai havia vist abans la resta de tecnologies que tractaré, la qual cosa m'ha suposat més de la meitat del temps dedicat al TFC fer les cerques, entendre els conceptes i implementar-los funcionalment. Per això, crec personalment que mereixen un capítol propi, malgrat que estic segur que un coneixement previ d'aquestes i de J2EE m'hagués permès millorar molt el TFC. Anem doncs pas per pas a analitzar els requeriments bàsics:

2.1.- Aplicacions *web* dinàmiques

Abans d'activar tot el sistema, cal que el proveïdor de serveis hagi indicat al proveïdor d'identitat quins clients ha d'atendre, quins recursos poden sol·licitar i en quina franja horària han de fer la sol·licitud. Això requereix que l'administrador del proveïdor d'identitat introdueixi aquestes dades en el servidor propi i les emmagatzemi de forma persistent. L'accés al servidor serà remot, pel què ens cal una aplicació al servidor que generi una interfície gràfica al navegador de l'administrador per a què aquest introdueixi les diverses dades. El codi que arribarà al navegador serà una pàgina *web* amb un formulari, i aquesta haurà de ser generada per una aplicació *web* dinàmica que pugui recollir les dades i gravar-les. He pensat doncs en utilitzar:

- ***Servlets***, aplicacions que s'executen dins un servidor i que generen pàgines *web* dinàmiques en resposta a les peticions dels clients. En Java, aquesta funcionalitat s'implementa en la *Java Servlet API*, que actualment es troba en la versió 2.5. Aquestes llibreries no es troben als *JDK* de les *Standard Edition*, i requereixen utilitzar la plataforma *Java Enterprise Edition*, actualment en la versió 5, que ofereix un *SDK* (*update* 5) que incorpora també el *JDK SE 1.6*. L'execució dels *Servlets* en un servidor requereix que aquest tingui alhora la funcionalitat de *contenidor de Servlets*, i per tant el motor d'execució específic.

- Alternativament, i amb la finalitat de facilitar la creació de codi *web* dinàmic, existeix la possibilitat de recórrer a la tecnologia ***Java Server Pages***, que té exactament les mateixes funcionalitats que els *Servlets* (podem considerar-la un cas espacial d'aquests). Aquesta tecnologia es troba actualment en la versió *JSP 2.*1, i també s'inclou al *JEE 5 SDK*. L'avantatge que ofereix és que en comptes de generar el codi HTML enviant les cadenes de text a la sortida (codi Java *out.println* emprat reiterativament), permet escriure directament codi HTML on s'hi insereixen *scripts* i accions predefinides degudament etiquetats per executar codi Java dins la pàgina. En aquest cas, quan el servidor rep la petició el codi JSP s'interpreta per generar el codi d'un servlet equivalent, que es compila per generar el *bytecode* final que s'executarà per generar l'HTML. Aquesta tecnologia requereix doncs que el servidor disposi d'un motor que compili el *JSP* al *Servlet* (o be que l'interpreti *on-fly*).

Entre les dues opcions, m'ha semblat que els *Servlets* utilitzen bàsicament codificació Java, amb la qual em sento més còmode. Crec que si no s'han de generar grans quantitats de codi HTML, no surt a compte aprendre totes les etiquetes, directives i accions de les *JSP*, pel què en principi vull provar d'implementar les funcionalitats del servidor només amb *Servlets*.

Finalment, indicar en aquest apartat que la generació d'interfícies gràfiques simples que permetin la interacció i el flux de dades client-servidor es poden assolir amb la mateixa tecnologia HTML a partir dels:

- ***Javascript***, que permet realitzar formularis *web* amb etiquetes específiques d'HTML (a partir de l'arrel <*FORM*>) i incloure en les pàgines certes funcionalitats que permetin executar codi Java i també comunicar diversos components. A més, permet crear en el navegador una plantilla que inclogui components gràfics com camps textuals, botons, llistes, ràdios, etc. que el client pot seleccionar i omplir. Amb certes accions, es pot enviar la informació recollida (de l'usuari, dels components) a un *servlet* del servidor per a què generi la resposta adequada. És una tecnologia completament nova per a mi.
- ***Awt* i *swing*** , que són les tecnologies bàsiques per dur a terme interfícies gràfiques per a l'usuari, especialment en els *applets*. És un llenguatge complicat i que he treballat poc, però penso que pot ser de profit per determinades funcionalitats que requereixin una part vistosa, com la que agrairà el client en el seu *applet*.

Aquestes tecnologies semblen suficients per permetre la comunicació bàsica i la transferència de dades del client amb el proveïdor de serveis i de l'administrador amb el proveïdor d'identitat. En tot cas, recordar que els canals emprats han d'assegurar la privacitat i permetre l'autenticació d'un o ambdós extrems.

2.2.- Tecnologies *XML* i *SOAP*

Segons els requeriments del TFC, cal que la petició del recurs per part del client quedi recollida en un document *XML* generat pel proveïdor de serveis. Aquest document, juntament amb els ítems necessaris per obtenir l'autorització, les ha d'enviar el client al proveïdor d'identitat encapsulades dins una petició *SOAP*. Un cop presa la decisió, aquest haurà de generar un document *XML* amb la resposta i les dades d'autenticació, que haurà de tornar al client en forma de resposta *SOAP*. Per això, necessitem emprar:

- ***XML***, la sintaxi *Extensible Markup Language*, que permet estructurar la informació jeràrquicament en documents de text pla que poden ser transmesos sense problemes. Mitjançant l'etiquetatge, permet aportar informació sobre la informació que conté (metadades), i això, juntament amb altres tecnologies satèl·lit (*DTD, Schema, XSLT, XPath, XQuery*, etc) li dóna una versatilitat pràcticament infinita. La capacitat programàtica per crear, editar, manipular i transformar entre diversos formats documents *XML* es troba implementada en un gran nombre de llibreries: en Java, les més importants i esteses semblen la *SAX* (*Simple API for XML*) i la *XMLDom* (*Document Object Model*, suportada pel *W3C*), que segueixen els models seqüencial i aleatori per accedir al document. En principi, la manipulació dels documents (sempre que càpiguen a memòria) sembla força més fàcil amb el model *DOM*, malgrat que la llibreria *SAX* n'és un complement que s'empra per aprofitar els seus *parsers* en les operacions d'entrada-sortida amb fitxers. Ambdues es troben incorporades al *Java 6 SE JDK*.
- ***SOAP*** (*Simple Object Access* Protocol), es pot considerar un subconjunt de *XML* (com ho són *XHTML* i la major part de *HTML*). Específicament, és un protocol escrit en sintaxi *XML* que permet transmetre missatges en un entorn de Serveis *Web*: això permet generar missatges de petició a un servidor especialitzat, i rebre'n els missatges de resposta. En els missatges de petició es poden fer crides a aplicacions residents en un altre node (*RPC*) i passar-les les dades necessàries per generar la resposta. El canal normal de transport dels missatges és *HTTP*, que permet gestionar els missatges de petició i resposta amb mètodes *GET* i *POST*. L'especificació actual de *SOAP* és la 1.2, i es troba suportada al *Java 6 SE JDK*, malgrat que certes llibreries de la fundació *Apache* (*Apache SOAP API 2.2* o *Axis2*) semblen ser les més emprades i ben establertes. Feta la reflexió que el node client (un navegador usual) és qui haurà d'enviar les peticions i rebre les respostes, m'ha semblat que utilitzar aquestes llibreries externes en els servidors podria requerir la instal·lació de programari addicional en el node client, per la qual cosa intentaré ajustar-me a les *API* de *Sun*. Així i tot, les transaccions *SOAP* requeriran que el servidor les suporti en el seu motor, per la qual cosa no descarto haver de recórrer a les llibreries *Axis*.

2.3.- Tecnologies de comunicació segura

Aquestes tecnologies haurien de ser el nucli central del TFC (malgrat que en el meu cas queden eclipsades per l'ombra del meu desconeixement de les tecnologies de les aplicacions *web*). Anem doncs a explorar com ens poden ajudar, començant per mantenir la privacitat de les comunicacions, és a dir, el xifrat. En aquest punt, i si tenim en compte que el transport de dades el confiem al protocol HTTP que ha d'entendre el navegador del client, la tecnologia requerida és ben clara:

- Hem d'interposar una **capa *SSL*** (*Secure Sockets Layer*) entre els *sockets* reals del sistema operatiu i l'aplicació *web* (servidor o navegador) que genera i rep les dades. Aquesta capa permet que els dos extrems negociïn una clau simètrica compartida però secreta: qualsevol que escolti la negociació no ha de poder inferir la clau. Això s'aconsegueix amb les metodologies de Diffie-Hellman o amb certificats, basats en claus asimètriques: cada extrem té un fragment de la clau simètrica, que xifra amb la clau pública de l'altre extrem i li envia el xifrat. Alhora, rep el fragment que li falta xifrat per l'altre part amb la clau pública pròpia. Així, cada part disposa dels dos fragments, un de conegut i l'altre xifrat amb la seva clau pública: el darrer el desxifra amb la seva clau privada i reconstrueix la clau original. L'altre extrem fa el mateix: al final del protocol, tots dos tenen els fragments per reconstruir la clau simètrica compartida, però per la xarxa només han viatjat els fragments xifrats. A partir d'aquest punt, el xifrat simètric dóna la rapidesa suficient per xifrar i desxifrar en temps raonables els paquets de dades que s'intercanvien amb protocol *HTML*. La capa *SSL* fa aquestes operacions de forma transparent per ambdós usuaris, i només cal especificar el port del servidor que s'ha configurat per suportar-ho: normalment és el *TCP*-443, al qual el navegador s'hi connecta quan s'especifica el protocol *HTTP* segur (*HTTPS*) davant la *URL*. Les versions modernes dels navegadors més usuals ja incorporen aquesta capacitat, pel què només cal assegurar que el navegador es pugui configurar per escoltar al port *HTTPS*: és possible que per assolir aquesta funcionalitat en entorns Java calgui instal·lar al servidor *web* l'extensió *JSSE* (*Java Secure Sockets Extensions*).

2.4.- Tecnologies d'autenticació i integritat

Com l'anterior, l'autenticació es basa en la criptografia de clau asimètrica: la part pública d'una clau asimètrica és coneguda per tothom, però la privada només la coneix el propietari, i per tant n'és un signe d'identitat. Així i tot, a cada part privada li correspon una única part pública, i per tant la clau pública també n'és un signe d'identitat únic. Aquest principi és la base de dues tecnologies que permeten tant l'autenticació com la integritat de qualsevol document electrònic:

- Les **signatures electròniques** es basen en què qualsevol document electrònic és serialitzable com una cadena d'octets, de la qual es pot generar un resum a base d'aplicar-hi seqüencialment una funció *hash* unidireccional: el resultat és una cadena d'octets d'una longitud fixada i curta, de la qual no es pot inferir l'entrada original. Tanmateix, la funció s'ha dissenyat per a què una ínfima variació en el document original doni lloc a un resum prou diferent. Per validar aquest document, l'usuari xifra el resum amb la seva clau privada: el resultat xifrat es coneix com a signatura, i s'uneix al document. Qualsevol persona pot comprovar si el document l'ha validat el signatari: només cal agafar el document a validar, calcular-ne el resum i comparar que és el mateix que l'obtingut en desxifrar la signatura amb la clau pública del signatari. El procediment també ens ha assegurat la integritat del document, doncs qualsevol manipulació del document validat ens portaria a un resum diferent de la signatura desxifrada.

Si tenim en compte que els documents que haurem d'autenticar són textos de sintaxi *XML*, fóra bo utilitzar l'estàndard que el *W3C* ha previst per signar aquests documents: *XMLDsig*. Aquest estàndard permet definir tres formes de signatura totes elles en format *XML*: *envelopping* (la signatura incorpora el document original en un node), *envelopped* (el document *XML* incorpora la signatura en un dels seus nodes) i *detached* (la signatura és separada i independent del document). Pels nostres propòsits, utilitzarem la tercera modalitat, atès que el mateix document haurà de ser signat per diversos signataris. En entorns Java, el *Java 6 SE JDK* conté les *API* per poder dur a terme la generació i verificació de signatures amb aquest estàndard, així com la gestió i manipulació dels principals tipus de claus asimètriques.

El problema d'aquest procediment és assegurar que la clau pública que tenim correspon vertaderament al signatari. Si no fos així, bé mai validaríem les signatures, bé un tercer podria falsificar les signatures amb una clau privada que fos la parella de la pública que atribuïm equivocadament al signatari. Si trobem una forma d'associar de manera inseparable el nom de l'usuari i la part pública de la seva clau, i posar el conjunt en un repositori públic, sempre podrem reconèixer sense dubtes i davant tercers si un usuari ha xifrat personalment un document amb la seva clau privada. Aquesta idea és la base de la segona tecnologia que permeten assegurar l'autenticitat i la integritat de documents electrònics:

- La **infraestructura de clau pública** (*PKI*) es basa en què tothom que la utilitza té plena confiança en uns pocs nodes anomenats autoritats de certificació, que posseeixen una parella de claus asimètriques. Aquestes *CA* es dediquen a unir les dades d'un usuari i la seva clau pública en un document, generar un resum del conjunt amb una funció *hash* unidireccional i públicament coneguda, xifrar el resum amb la clau privada de la *CA* i unir el xifrat al document. Al xifrat del resum amb una clau privada se l'anomena **signatura**, i el propietari de la clau privada és el signant. El document resultant que conté les dades de l'usuari, la seva clau pública i el resum xifrat per la *CA*, se l'anomena certificat de l'usuari, i es diposita en un repositori públic. De la mateixa manera, la *CA* penja al repositori el certificat propi on el resum està xifrat amb la clau privada de la

CA (*auto signat*). Qualsevol persona que rebi un document signat per un usuari de la *PKI*, pot comprovar davant tercers si la signatura l'ha feta l'usuari: només cal anar al repositori, agafar la clau pública del certificat de l'usuari i comprovar que quan es desxifra la signatura s'obté el mateix resum que quan s'aplica la funció *hash* al document signat. Alhora, estem garantint la integritat del document, doncs qualsevol modificació posterior a la signatura n'alteraria el resum i no coincidiria amb el desxifrat. Això també garanteix la integritat dels certificats, ja que estan signats amb la clau privada de la CA, que exposa la clau pública en el seu certificat. Aquesta metodologia permet que un cop s'ha emès un certificat a un usuari, aquest pot emprar la clau privada per signar altres certificats. Es poden generar així cadenes de certificats on el certificat de l'emissor permet verificar el certificat emès. L'excepció són els certificats arrel auto signats: per creure en la *CA* només podem tenir-hi confiança, i acceptar cegament el seu certificat. Per aquests motius, les *CA* són també responsables d'anul·lar els certificats emesos a usuaris que no mereixen la confiança o han perdut el secret de la seva clau privada: han de fer públiques en un repositori les llistes de certificats revocats (*CRL*).

La utilització de certificats com a document d'autenticació es pot estendre a l'establiment de les comunicacions client-proveidor HTTPS: la capa *SSL* admet un protocol d'autenticació d'un extrem (el servidor del proveïdor de serveis) o dels dos (el servidor del proveïdor d'identitat i l'administrador que hi carrega les dades) mitjançant l'intercanvi de certificats, amb el què cada part rep una signatura que pot desxifrar amb la clau pública del certificat del signant. Tan sols cal comprovar que el servidor té les *SSL* instal·lades (extensió *JSSE*) i que s'ha configurat el port HTTPS per exigir l'autenticació.

En aquest punt cal prendre una decisió sobre quina *CA* ha de signar els certificats que emprarem per provar el nostre sistema. L'obtenció de certificats de *CA* privades requereix temps i diners. Les administracions públiques estatal i catalana permeten que els ciutadans sol·licitem certificats personals gratuïtament (a través de la FNMT i el sistema *idCat* respectivament), però en el primer cas l'accés a les CRL és de pagament, i en el segon poden haver-hi problemes per accedir el dispositiu extern on es guarden les claus. L'accés al certificat del nou *DNIe* té una problemàtica similar. Per tot això, generaré certificats auto signats de CA per fer les proves, a partir dels quals generaré els certificats dels proveïdors. Al cas del certificat del client, també s'haurà de generar una CRL per a què es pugui validar amb més rigor.

Java 6 SE JDK incorpora les *API* per poder treballar amb el model de certificats emprat en les *PKI*: els X.509v3. Amb l'ajut de l'eina *Keytool* es poden generar certificats i contenidors per a ells que s'integren perfectament a la plataforma. Així i tot, també existeix l'eina *OpenSSL* que té les mateixes funcionalitats, i que és la que prioritàriament penso emprar degut a l'experiència prèvia adquirida en altres assignatures.

2.5.- Tecnologies d'execució local de programes al navegador

Per facilitar la tasca de clients poc experts, s'ha decidit que la signatura per part d'aquest de la petició generada pel proveïdor de serveis, la generació de la petició *SOAP* de l'autorització al proveïdor d'identitat i l'autenticació i tramitació de la resposta al primer, s'executin de forma local i el màxim d'automatitzada en el navegador del client. Per aconseguir-ho, un cop ha fet la sol·licitud al proveïdor ha de rebre una pàgina *web* que contingui un programa compilat que en executar-se al navegador li vagi presentant pantalles d'estat, li demani les dades mínimes per dur a terme les tasques predeterminades (ubicació de fitxers i certificats, clau privada i mot de pas per fer la signatura), i li permeti executar cada nova acció amb una acció intuïtiva (prémer un botó amb el ratolí, etc). Aquesta interacció i funcionalitat les permeten certs components *web* anomenats:

- **Applets**, són aplicacions que s'executen en el context d'un altre programa amfitrió o contenidor, típicament dins una pàgina web on es mostren com una finestra. A través d'altres tecnologies com *Java Swing* o *AWT* que els doten de funcionalitat, poden oferir informació gràfica i interaccionar amb l'usuari demanant-li dades i/o entrades que desencadenen accions. En el fons són classes Java que estenen la classe *Applet* o la seva subclasse *JApplet* (ambdues són *Components* de *java.awt*). Les classes i interfícies d'aquest paquet ofereixen els mètodes i components genèrics necessaris per executar totes aquestes funcions. El programa resideix de fet en el servidor, i el seu *bytecode* és cridat per la mateixa pàgina *web* que s'ha descarregat al navegador. Pel què he llegit, *Java 6 SE JDK* incorpora les llibreries que suporten aquests components.

Com a consideracions de seguretat, cal que l'*applet* manipuli dades sensibles del client i executi i verifiqui signatures: això requereix que el client tingui plena confiança en el proveïdor de serveis que incrustarà el component al seu navegador, i que en pugui garantir l'autenticitat: cal que l'*applet* estigui signat. Aquesta exigència, té importants repercussions en l'execució del programa:

- La **signatura d'*applets*** implica la compressió prèvia a un arxiu *jar* i la ulterior signatura d'aquest amb el certificat del servidor, cosa que es pot fer amb l'eina *jarsigner*. Si un navegador rep un *applet* sense signar, l'executa en un entorn estanc on el *JRE* no li permet fer operacions sensibles (escriure o llegir del disc, establir noves connexions, etc). En canvi, si un *applet* és signat i es pot verificar la confiança del certificat, el *security manager* de Java consulta els fitxers *java.policy* per veure els permisos que s'han d'atribuir als executables signats per un determinat signatari. Amb aquesta tecnologia, el client pot garantir que els *applets* signats pel proveïdor de serveis tinguin els privilegis suficients per executar les funcionalitats requerides. En el cas de l'execució en un navegador, alguns *plug-in* identifiquen si l'*applet* descarregat és signat, i abans d'executar-lo informen a l'usuari sobre qui és el signatari, i demanen si se li volen donar tots els permisos.

2.6.- Tecnologies de persistència de les dades

Un requeriment mínim per a la persistència de les dades que l'administrador incorpora al servidor del proveïdor d'identitat és que s'escriguin al disc de manera que siguin accessibles tant per l'administrador com per l'aplicació d'emissió d'autoritzacions. La forma més simple de fer-ho és posar-les en un fitxer de format prefixat. La manera més professional i segura consisteix a fer-ho amb un SGBD que treballi amb SQL:

- Ja que *Informix* és un programari propietari, el programari *MySQL* sembla una bona opció lliure per implementar i gestionar la persistència: permet a través de ponts JDBC-ODBC l'accés a les dades tant des de l'aplicació d'autorització com des de l'aplicació d'administració. Així i tot, he ponderat que la configuració i posta en marxa del servei SGBD, així com la programació del gestor de disc que generi les ordres *SQL* dins l'entorn Java, poden requerir un temps excessiu. Ja que el punt crític d'aquest TFC és la seguretat, i davant la previsió d'haver de dedicar molt de temps a les tecnologies web, em reservo la decisió d'implementar aquest subsistema per a la darrera etapa d'implementació. La fundació *Apache* ofereix *Derby*, un SGBD relacional implementat en *Java* i que és força més simple de configurar i posar en marxa que l'anterior. Finalment, aquesta darrera ha estat l'opció triada.

2.7.- Tecnologies de servidor

Finalment, cal escollir un servidor que sigui capaç de suportar totes les tecnologies *web* exposades anteriorment (bàsicament *Servlets*, *JSP*, *SOAP* i connexions *SSL* amb autenticació mitjançant certificats). He trobat diverses opcions, malgrat que les més esteses en el món comercial són *Weblogic 10* de BEA (una companyia relacionada amb Oracle) i *WebSphere 7.0* de IBM. Ambdues són de codi propietari, i en ocasions requereixen d'extensions externes per assolir les funcionalitats requerides, i no suporten completament tots els estàndards de Sun. En canvi, hi ha una solució de codi lliure que pràcticament ho porta tot incorporat:

- **El servidor *Tomcat***, de la fundació *Apache*. Els tutorials indiquen que és prou lleuger i que instal·lar-lo no és en excés complicat, malgrat que sembla que la configuració (en especial si es volen incloure les *SSL*) no és en absolut trivial. Disposa d'un motor de *servlets* (de fet el servidor es defineix com un contenidor d'aquests), un motor *JSP* i un motor de *SOAP*. No diré que sigui la única opció possible, però sí sembla que és la idònia per suportar les aplicacions i protocols que haurem d'utilitzar en aquest TFC. A més, aquest servidor és altament configurable i es troba àmpliament documentat a internet.

2.8.- Tecnologies de desenvolupament

Vistes les possibles tecnologies que caldrà emprar, és imprescindible treballar amb el *Java2 EE SDK*, i és possible que per ajudar a fer la implementació també hagi de recórrer a la versió especialitzada de l'IDE *Eclipse*, la *Eclipse IDE for Java EE Developers*.

Al final, m'he trobat que la versió *J2EE* d'*Eclipse* era massa complexa pels meus pocs coneixements de tecnologies web, i que em complicava més la feina que no pas me la resolia. Tampoc he tingut temps d'estudiar aquesta eina, per la qual cosa m'he espavilat amb la *IDE* normal.

CAPÍTOL III : PLANIFICACIÓ I DISSENY

Una correcta planificació seguida d'un bon disseny han de fer més fàcil la implementació del sistema. Malgrat això, durant la fase d'implementació sempre sorgeixen problemes, inconvenients i imprevistos que requeriran cíclicament una replanificació de la feina i un redisseny del sistema.

Malgrat això, cal tenir una base ferma que en orienti com una brúixola per aquests tortuosos camins que haurem de seguir i es permeti no perdre de vista l'objectiu final. Per això, aquesta fase és una de les més importants, encara que potser el producte final sigui ben diferent del previst. Anem doncs a pensar com hauríem d'implementar idealment el sistema.

3.1.- Arquitectura

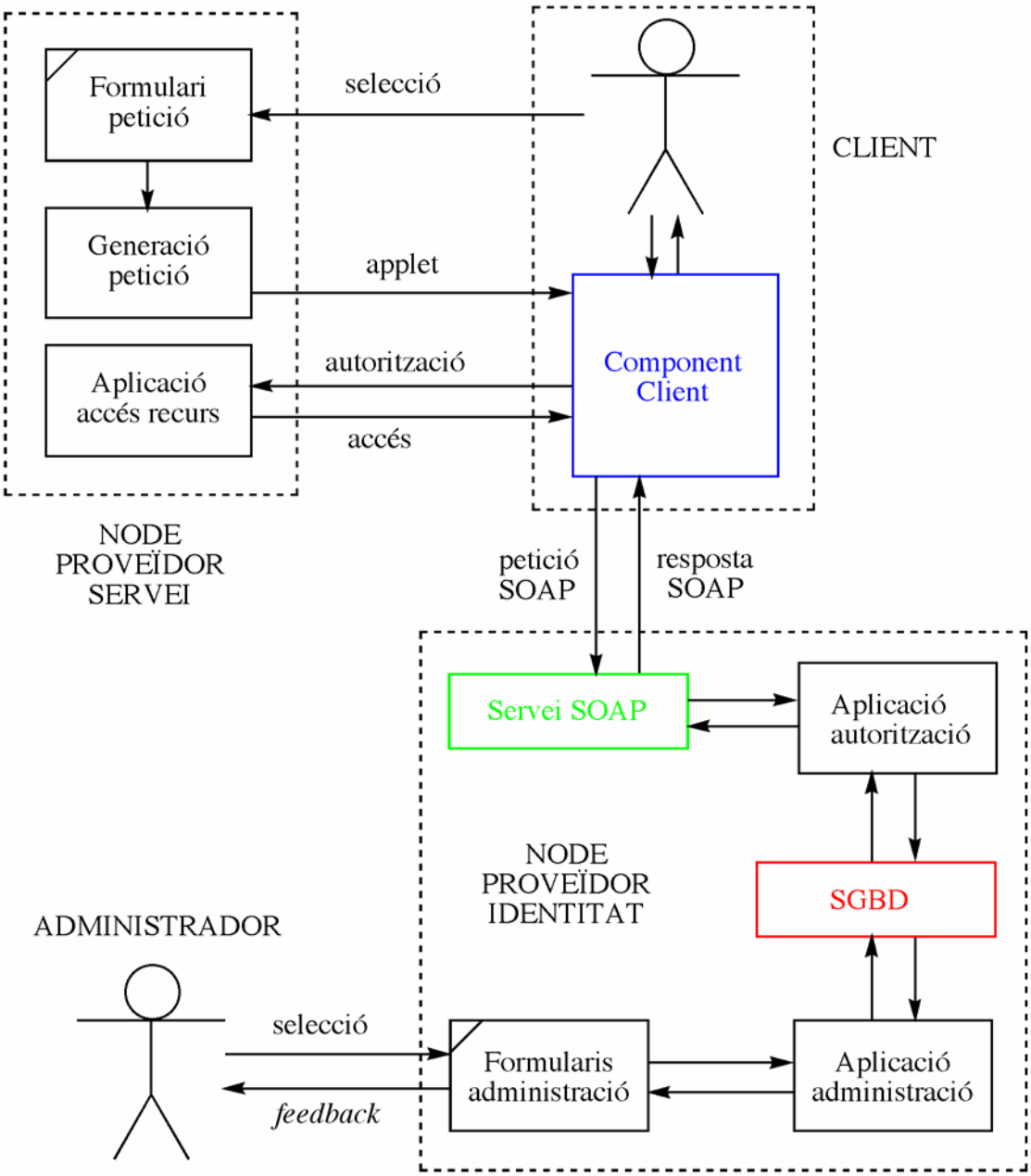
El sistema hauria de constar de dos nodes:

- **El proveïdor de serveis**, amb un servidor encarregat de servir pàgines web amb formularis. Un subsistema hauria de generar la petició i llur signatura i enviar l'*applet* incrustat al client. L'altre ha de rebre d'aquest l'autorització del PI i, si s'escau, oferir el recurs.
- **El proveïdor d'identitat**, amb un SGBD i un servidor on hi corren dues aplicacions web i un servei SOAP. L' aplicació web d'administració ha de permetre a l'administrador l'accés en lectura i escriptura a la BBDD dels permisos per gestionar-la a través del navegador, via aplicacions gràfiques o formularis web (l'opció triada). L'aplicació web d'autorització ha de rebre petició i signatures, verificar-les, esbrinar la identitat del client, consultar els seus permisos i emetre la resolució d'autorització degudament signada. El servei, basat en un motor *SOAP*, serà l'encarregat de rebre els missatges amb les dades que li enviï el client i respondre'ls.

Aquests nodes, han d'estar disponibles en xarxa per acceptar connexions tant HTTP con HTTPS, que seran establertes independentment i mitjançant navegadors per dos actors:

- **El client:** ha de connectar de forma segura amb el proveïdor de serveis per fer la sol·licitud del recurs, i rebre'n un document de petició i la signatura. Ha de signar ell mateix el document i trametre'l de forma segura al proveïdor d'identitat, juntament amb les dues signatures, per rebre'n l'autorització signada. Finalment, i verificada aquesta, ha de lliurar-la al proveïdor de serveis dins la mateixa sessió per a què se li concedeixi el recurs.

- **L'administrador:** s'ha de connectar amb l'aplicació web de gestió al servidor del proveïdor d'identitat mitjançant un canal segur i autenticar-se. Ha d'introduir (i eventualment poder consultar, modificar i esborrar) les dades que li hagin passat independentment el proveïdor de serveis i el client.



Remarcar finalment que és imperatiu que **totes** les comunicacions entre nodes es facin de forma segura per intercanvi de dades forçat sobre protocol *HTTPS*, i que en el cas de l'administrador cal a més un protocol d'autenticació que l'identifiqui davant el servidor.

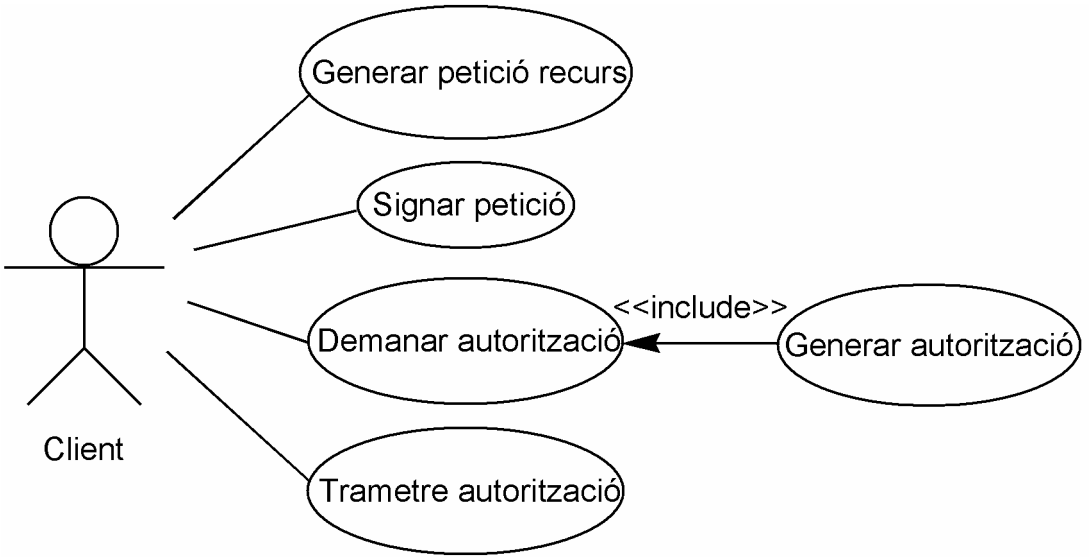
3.2.- Model de domini i de negoci

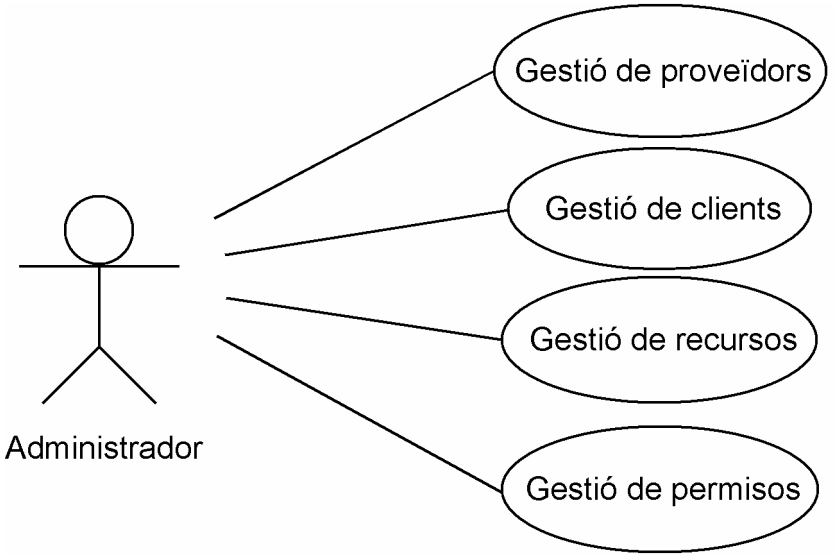
De les discussions i consideracions exposades a la PAC2, podem fer una primera identificació d'objectes, entitats o classes, i alguns de llurs atributs:

- Petició (Proveïdor de servei, ID petició, recurs, hora de petició)
- Signatura (Certificat)
- Permís (Proveïdor de servei, recurs, client, interval horari)
- Missatges SOAP (una sol·licitud i una resposta)
- Autorització (Petició)
- Gestor de persistència
- Formularis web i applet

Com a actors, havíem identificat el client i l'administrador del proveïdor d'identitat. Penso que segons el seu comportament, l'applet que s'executa al navegador del client també es podria considerar un actor, doncs envia i rep els missatges SOAP al proveïdor d'identitat: així i tot, crec que només envia la petició SOAP quan el client autoritza la signatura, i per això consideraré que en aquest cas l'actor és realment el client.

Feta la reflexió, pensem en els casos d'ús:





3.3.- Guions dels casos d'ús

Anem a aprofundir en les funcionalitats que implica cada cas d'ús

3.3.1.- Cas d'ús "Generar petició recurs"

Actor: client

Casos d'ús relacionats: "Signar Petició"

Precondició: No existeix la petició del recurs per part del client. El servidor del proveïdor de serveis té accés al magatzem PKCS12 que conté la seva clau privada i als mots de pas que hi permeten l'accés.

Postcondició: S'ha generat una petició en format XML i la seva signatura per part del proveïdor de serveis (amb segell de temps per part d'un tercer). S'ha descarregat un *applet* al client que li permet visualitzar la petició i accedir a la signatura.

El client es connecta al servidor del proveïdor de serveis a través d'un canal segur (protocol HTTPS) i descarrega un formulari amb els serveis oferts. En selecciona un i valida l'elecció. El servidor genera la petició i la signatura segellada, i les descarrega juntament amb (mitjançant) l'*applet* signat al navegador del client. El client autoritza l'execució de l'*applet* confiant en el certificat inclòs: un cop fet pot visualitzar la petició i accedir tant a aquesta com a la signatura.

Opcions de millora: afegir el segellat de temps per part de la TSA

3.3.2.- Cas d'ús "Signar petició"

Actor: client

Casos d'ús relacionats: "Generar petició recurs", "Demanar autorització"

Precondició: El navegador del client està executant l'*applet* que té accés a la petició i a la signatura segellada del PS.

Postcondició: S'ha generat una segona signatura de la petició XML amb la clau privada del client.

L'*applet* ofereix al client la possibilitat de signar la petició amb la seva *keystore*, pel què li demana la ubicació d'aquesta al disc i el mot de pas per accedir la clau privada. El client proporciona les dades i en confirmar es genera la segona signatura.

Opcions de millora: l'applet podria oferir triar entre els *keystores* que el client tingui a la ubicació estàndard de Java

3.3.3.- Cas d'ús " Demanar autorització "

Actor: client

Casos d'ús relacionats: "Generar petició recurs", "Signar petició", "Trametre autorització"

Precondició: El navegador del client està executant l'*applet* i té accés a la petició, a la signatura segellada i a la signatura pròpia. També té accés al certificat de la CA del proveïdor d'identitat.

Postcondició: S'ha obtingut una autorització del proveïdor d'identitat juntament amb la signatura d'aquest.

L'*applet* ofereix al client la possibilitat de trametre els tres documents anteriors al proveïdor d'identitat a fi d'obtenir l'autorització. Quan assenteix es genera una petició *SOAP* que els conté així com la *URL* del servidor del proveïdor d'identitat i el nom del servei que cal cridar per obtenir l'autorització. Es demana al client que autoritzi la tramesa de la petició. En confirmar, s'envia la petició i s'espera rebre la resposta. Quan aquesta arriba, es verifica la signatura del proveïdor d'identitat i es mostra el contingut de l'autorització.

Opció de millora aplicada a la implementació: finalment, he optat per fer aquest pas s'executi automàticament a continuació de l'anterior i sense intervenció del client, ja que no té sentit que aquest faci la signatura i decideixi no fer l'enviament al PI.

3.3.4.- Cas d'ús " Trametre autorització"

Actor: client

Casos d'ús relacionats: "Demanar autorització"

Precondició: El navegador del client està executant l'*applet* i té accés a l'autorització del proveïdor d'identitat i a la signatura d'aquest.

Postcondició: S'ha obtingut o denegat l'accés al recurs sol·licitat.

L'*applet* ofereix al client la visualització de la resposta del proveïdor d'identitat i la possibilitat de trametre l'autorització i la signatura associada al proveïdor de serveis. En confirmar, s'envien els documents al proveïdor de serveis per a què, un cop verificats, s'enviï al navegador del client una pàgina *web* que li permeti l'accés al recurs o l'informi que li ha estat denegat

Opció de millora aplicada a la implementació: si l'autorització és positiva s'ha implementat directament la descàrrega del recurs?

3.3.5.- Cas d'ús "Generar autorització" <<*included*>>

Actor: client (indirectament)

Casos d'ús relacionats: "Demanar autorització"

Precondició: El navegador del client està executant l'*applet*, s'ha generat una petició *SOAP* que conté la sol·licitud XML i les dues signatures, i aquesta s'ha enviat al servidor del proveïdor d'identitat. El servidor té accés als certificats i CRL de les CA del proveïdor de serveis i del client, així com al magatzem PKCS12 de la seva clau privada i els mots de pas que li permeten l'accés.

Postcondició: El servidor del proveïdor d'identitat ha generat una resposta *SOAP* que conté l'autorització en format XML degudament signada, i l'ha enviada al navegador del client.

El servidor del proveïdor d'identitat ha rebut una petició de servei *SOAP* que tramet a l'aplicació corresponent. Aquesta extrau la petició XML i les signatures i verifica ambdues, incloent la CRL de la CA del certificat del client. Si tot és correcte, extrau les dades de la petició i les contrasta amb les que té emmagatzemades. Si s'acompleixen totes les condicions, es genera un document XML amb l'autorització positiva; cas contrari, se'n genera una de negativa. Es genera la signatura *detached* del document i ambdós s'encapsulen en una resposta *SOAP* que es retorna al navegador del client.

Opcions de millora: obtenir l'instant de la petició del segell de temps de la signatura del PS per part de la TSA.

3.3.6.- Cas d'ús "Gestió de clients"

Actor: administrador

Funcionalitat: introdueix, esborra i/o consulta un client a la BBDD.

Precondició: El client no és a la base de dades quan es vol crear. Hi ha de ser en la resta de casos.

Postcondició: El client s'ha incorporat a la base de dades en cas de creació; en la resta de casos s'efectua l'operació. S'emet un missatge de confirmació o d'error.

En tots els casos s'introdueix l'identificador del client i es permet l'operació.

Opcions de millora: oferir un desplegable amb els clients; si el client introduït no existeix oferir crear-lo.

3.3.7.- Cas d'ús "Gestió de proveïdors"

Actor: administrador

Funcionalitat: introdueix, esborra i/o consulta un proveïdor a la base de dades.

Precondició: El proveïdor no és a la base de dades quan es vol crear. Hi ha de ser en la resta de casos, juntament amb la seva URL.

Postcondició: El proveïdor s'ha incorporat a la base de dades en cas de creació; en la resta de casos s'efectua l'operació. S'emet un missatge de confirmació o d'error.

En tots els casos s'introdueix l'identificador del proveïdor i la seva URL.

Opcions de millora: oferir un desplegable amb els proveïdors; si el proveïdor introduït no existeix oferir crear-lo.

3.3.8.- Cas d'ús "Gestió de recursos"

Actor: administrador

Funcionalitat: introdueix, esborra i/o consulta un recurs d'un proveïdor a la base de dades.

Precondició: El proveïdor del recurs és a la base de dades, però el recurs no hi és quan es vol crear. Hi han de ser ambdós associats en la resta de casos.

Postcondició: El recurs s'ha incorporat a la base de dades i associat al proveïdor en cas de creació; en la resta de casos s'efectua l'operació. S'emet un missatge de confirmació o d'error.

Opcions de millora: oferir un desplegable amb els recursos; si el recurs introduït no existeix oferir crear-lo.

3.3.9.- Cas d'ús "Gestió de permisos"

Actor: administrador

Funcionalitat: introdueix, modifica, esborra i/o consulta un permís associat a un recurs a la base de dades.

Precondició: El recurs i el client a què fan referència el permís és a la base de dades, però el permís no hi és quan es vol crear. Hi han d'haver el permís amb l'interval horari en la resta de casos.

Postcondició: El permís s'ha incorporat a la base de dades i associat al recurs i al client en cas de creació; en la resta de casos s'efectua l'operació. S'emet un missatge de confirmació o d'error.

En tots els casos s'introdueixen els identificadors del recurs i del client que el vol accedir. Si es vol crear, a més cal indicar l'interval horari.

Opcions de millora: oferir un desplegable amb els permisos; si no es troba, s'ofereix crear-lo, demanant alhora la franja horària permesa. En la resta de casos s'accedeix al permís i es permet l'operació.

3.4.- Gestió de dades

En aquest punt estudiaré les dades que cal intercanviar a fi de definir millor els atributs imprescindibles per a cada classe i l'estructura de la base de dades.

3.4.1.- Estructura de les dades

El punt de partida és la petició, que incorpora com a informacions un identificador propi, la URL base del proveïdor, un identificador del recurs sol·licitat i l'instant en què es generà.

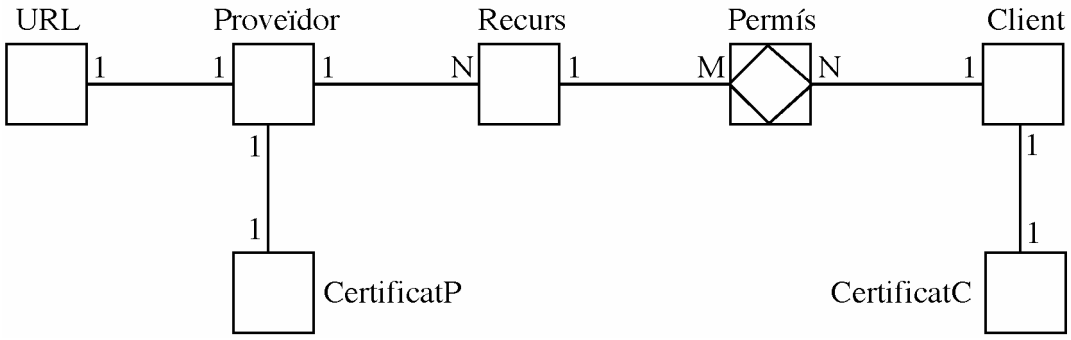
El punt clau és la gestió que tots els nodes han de fer dels certificats (en signar i verificar) per la qual cosa el client haurà d'estar identificat pel camp *Subject* de llurs certificats, que s'inclouen a les signatures que faci. De fet, aquest serà el principal indicatiu de la identitat del client que arribarà al proveïdor d'identitat (mitjançant el certificat inclòs a la signatura). En aquest TFC s'ha considerat per simplificar que el *distinguished name* de cada client és únic i l'identifica unívocament.

Quant al proveïdor de serveis, se li ha aplicat la mateixa suposició que al client, i a més se suposa que té una única *URL* base que li és exclusiva i també l'identifica.

He suposat que cada recurs també disposa d'un identificador únic que l'associa a un determinat recurs d'un proveïdor (dos recursos mai poden tenir un mateix identificador, malgrat que siguin de proveïdors diferents).

Finalment, un permís tindrà associats un únic client, un únic recurs i l'interval horari que hi permet l'accés.

En base a tot això, proposo la següent estructura relacional de dades que poso en forma gràfica d'un diagrama E-R:



He plantejat d'entrada que els identificadors principals de les entitats Client i Proveïdor siguin els seus CN. Les URL dels Proveïdors de serveis haurien de ser claus secundàries.

La relació Recurs necessita un identificador propi i la clau forana envers el Proveïdor que l'ofereix.

L'entitat associativa Permís requereix les claus foranes a Recurs i Client, i a més els atributs hh1, mm1, hh2 i mm2 que delimiten el període temporal.

Tot l'exposat es podria traduir en l'ús de les següents taules:

- Client: (CNc)
- Proveïdor: (CNp, *URL*)
- Recurs: (IdRecurs, URL)
- Permís: (IdRecurs, CNc, hh1, mm1, hh2, mm2)

Els subíndexs p i c indiquen la pertinença del CN a la taula Proveïdor o Client. He marcat en subratllat les claus primàries i en itàlica les secundàries. Les claus foranes són coincidents lexicològicament amb les primàries o secundàries a què apunten.

Un correcte disseny de la base de dades que reculli com a obligatòries aquestes relacions i provoqui esborrats en cascada quan una clau primària estigui apuntada per una forana mantindrà la congruència del sistema.

3.4.2.- Flux de gestió

En aquest punt estudiaré el flux de les dades que cal intercanviar i gestionar a fi de definir millor els mètodes i atributs de les classes.

En el moment que, dins la URL del Proveïdor, el Client selecciona un recurs, s'activa un servlet que genera un fitxer XML d'una estructura similar a:

```
<Peticio Id=XXX >
          <URL>URLbase</URL>
          <Recurs> IdRecurs </Recurs>
          <Temps>
                    <Hora>HH</Hora>
                    <Minuts>MM</Minuts>
          </Temps>
</Peticio>
```

L'identificador del recurs provindrà de la selecció del Client, mentre que l'identificador únic de la petició s'extraurà com el temps del sistema en mil·lisegons des de 1970. Del mateix temps s'extraurà l'instant de la petició.

NOTA IMPORTANT: A la fase d'implementació, ha calgut embolcallar la petició dins un node arrel `Document` per evitar certes interferències dels *parsers* d'XML, que en ocasions i de forma incontrolada afegien un *namespace* buit (`xmlns=""`) al node arrel, el qual interferia les verificacions de signatura. A hores d'ara encara no he trobat la causa d'aquest comportament.

Tot seguit el mateix servlet haurà de generar la signatura *detached* del document, enviar-la pel *timestamp* si s'escau, i retornar document i signatura al Client juntament amb un applet signat que s'executarà al seu navegador.

Després de demanar les dades pertinents al Client, l'applet generarà una nova signatura *detached* del document, inclourà les signatures juntament amb el document en un missatge SOAP i enviarà la petició RPC al servidor del proveïdor d'identitat.

El servlet del proveïdor d'identitat obrirà el missatge i n'extraurà el document XML original i les dues signatures, verificant ambdues contra els certificats de CA de la seva *keystore*, la CRL de la CA del Client i contra l'autoritat de segellat temporal si s'escau.

Tot seguit fóra convenient verificar mitjançant consulta a la base de dades que el CN del certificat de la signatura del proveïdor es correspon a la URL del proveïdor i que el Recurs demanat es correspon a la URL del proveïdor. Aquesta tasca no s'ha implementat i es deixa com a possible millora.

A partir d'aquest punt, cal identificar el Client a partir del certificat de la seva signatura. S'extreu doncs el X500Principal d'aquest i s'obté el CN del Client.

Amb el CNc del Client i l'ID del Recurs s'accedeix a la taula de Permisos i s'obtenen els marges horaris que decidiran l'autorització, que es confronta amb el temps en què es generà la Petició i per comprovar que caigui dins els marges horaris.

Si totes les fases anteriors s'han desenvolupat correctament, es passa a generar un document XML d'autorització amb valor "*cert*". Si algun dels procediments anteriors donés error, es generaria amb valor "*false*" i una petita explicació de la causa de la denegació:

```
<Autoritzacio>
          <Peticio IdPeticio=XXX >
                    <URL>URLbase</URL>
                    <Recurs> IdRecurs </Recurs>
                    <Temps>
                              <Hora> HH </Hora>
                              <Minuts> MM </Minuts>
                    </Temps>
          </Peticio>
          <Resultat> true/false+causa </Resultat>
</Autoritzacio>
```

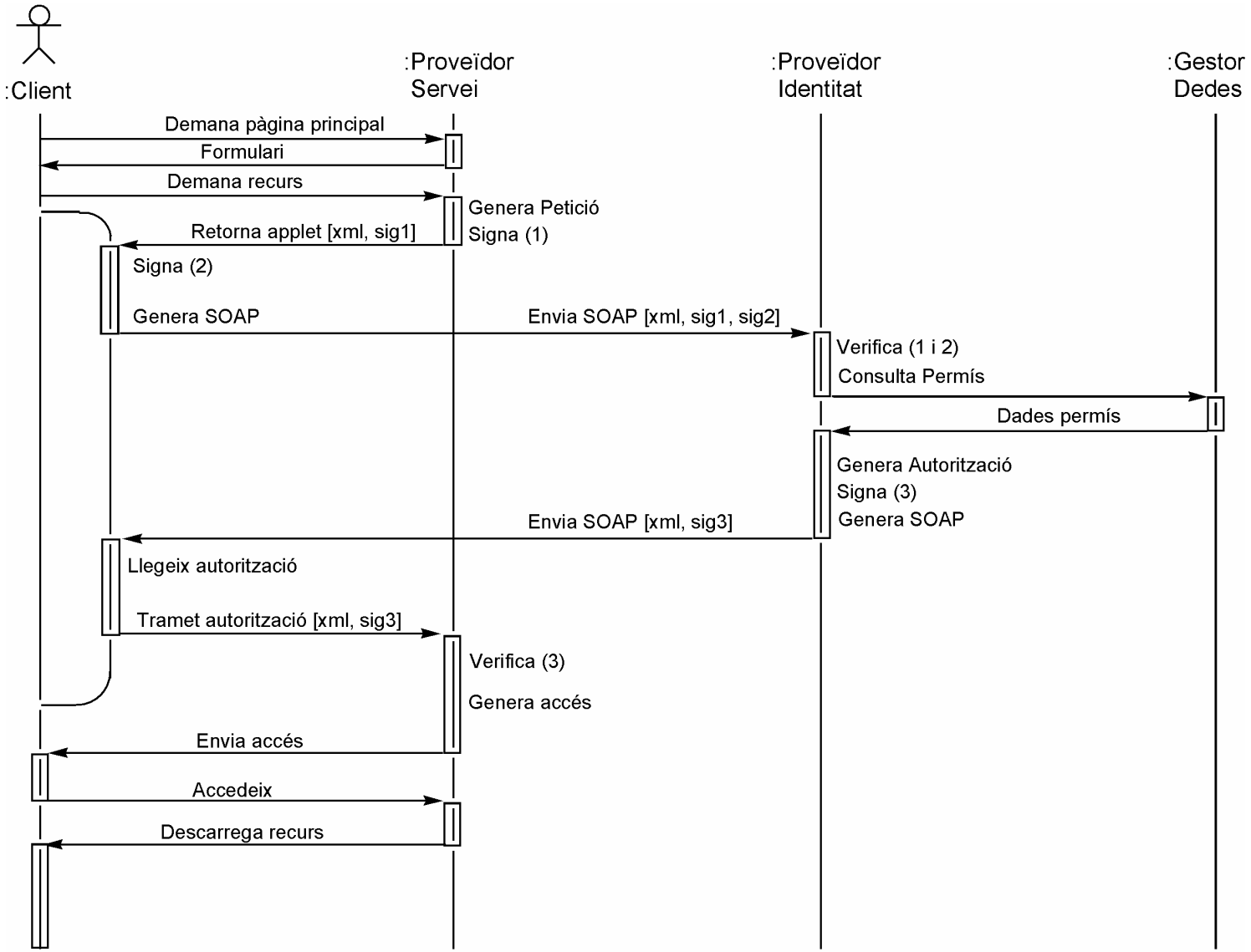
He pensat que el millor és que l'Autorització inclogui la Petició com a node, i així podem aprofitar les dades originals que conté.

Tot seguit el proveïdor d'identitat genera la signatura *enveloping* del document, l'encapsula en la resposta SOAP i la remet al navegador del Client.

L'applet que s'executa al navegador rep el missatge amb la signatura, n'extreu el document d'autorització i el mostra al Client, tot demanant-li permís per trametre-la al proveïdor de serveis.

Quan ho accepta, el servlet del proveïdor verifica la signatura, extrau el camp resultat i, si és afirmatiu, extreu l'identificador del recurs i hi dóna accés.

Tot l'esmentat es pot recollir en un diagrama de seqüència, que degut al seu format mostro a la pàgina següent:


:Client
:Proveïdor Servei
:Proveïdor Identitat
:Gestor Dedes
Demana pàgina principal
Formulari
Demana recurs
Genera Petició
Signa (1)
Retorna applet [xml, sig1]
Signa (2)
Genera SOAP
Envia SOAP [xml, sig1, sig2]
Verifica (1 i 2)
Consulta Permís
Dades permís
Genera Autorització
Signa (3)
Genera SOAP
Envia SOAP [xml, sig3]
Llegeix autorització
Tramet autorització [xml, sig3]
Verifica (3)
Genera accés
Envia accés
Accedeix
Descarrega recurs

3.5.- Disseny de les aplicacions de seguretat

Com hem vist a l'apartat d'arquitectura, dins els nodes hi conviuen diverses aplicacions web, cadascuna de les quals té funcionalitats ben definides. Aquestes actuen com a subsistemes en els quals entra una informació i en surt una altra. Per tant, anem a fer un disseny bàsic de classes de cada aplicació per veure com poden dur a terme aquestes transformacions.

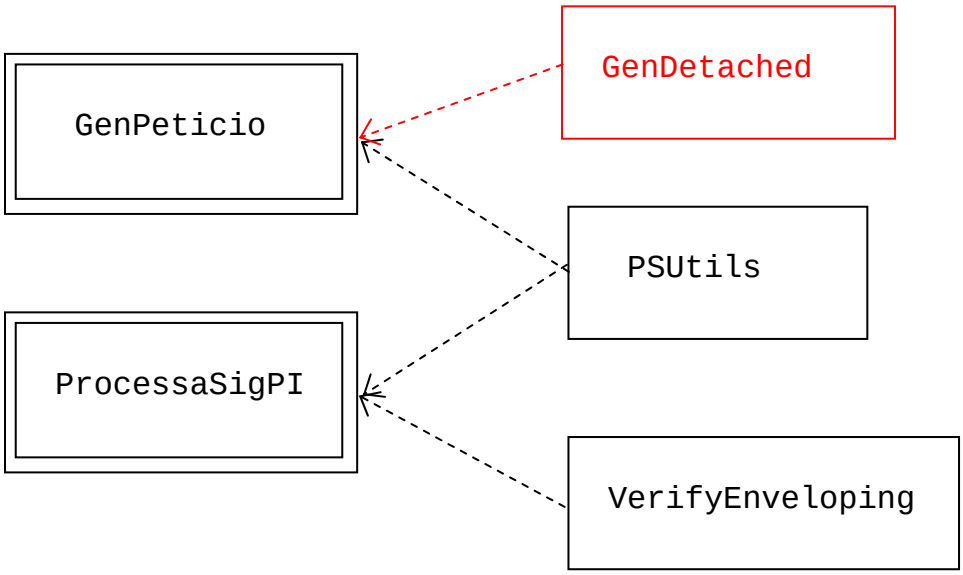
3.5.1.- Aplicació del Proveïdor de Serveis

La composarien bàsicament dos servlets (que als esquemes marco amb doble línia):

- L'encarregat de generar la petició XML a partir de les dades passades pel formulari de petició, signar-la i alhora de descarregar l'*applet* i els seus paràmetres (*Strings* en Base64) al navegador del client.
- L' encarregat de rebre l'autorització signada del proveïdor d'identitat de l'*applet* (també un *Strings* en Base64), verificar-la i concedir si s'escau l'accés al recurs al client, tot dins la mateixa sessió.

Els servlets requeririen una classe auxiliar per fer la signatura *detached*, una altra per verificar l'*enveloping* del PI, i una tercera amb les utilitats per codificar i descodificar en Base64.

L'estructura ideal seria encapsular les cinc en un *package* com:



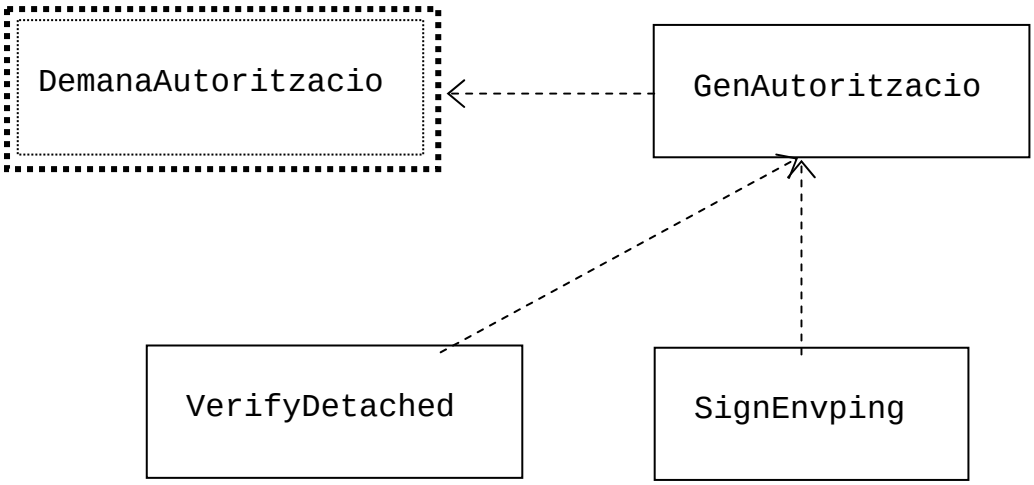
Degut als problemes que he tingut durant el procés d'implementació no he tingut temps de separar la classe *GenDetached* de l'aplicació *GenPeticio*, per la qual cosa el codi hi està integrat. Queda doncs com una millora a fer.

3.5.2.- Aplicació del Proveïdor d'Identitat

La composarien bàsicament:

- El servei SOAP (marcat amb línia doble de punts) encarregat de rebre els missatges de l'*applet* amb petició i signatures *detached* codificades en base64, trametre-les a les classes que generen l'autorització per a què els lliurin aquesta ja codificada en base64, i retornar l'*String* a l'applet encapsulat a la resposta SOAP.
- La classe cridada pel servei que en base als paràmetres rebuts genera l'autorització prèvia verificació de signatures i consulta a la BBDD.
- Dues classes auxiliars per verificar les signatures *detached* (que també consulten si els certificats han estat revocats a la CRL de la CA) i, un cop generat el document, el signen forma *enveloping*.

En aquest cas, no he inclòs les classes en un *package* ja que no sé com reaccionaria el servei empaquetat en un *aar*. És una millora que hauria d'estudiar amb més temps.



3.5.3.- Aplicació del Component del Client

La base és un *applet* que rep com a paràmetres la petició i signatura del PS codificades en base64, descodifica la petició, la signa amb el certificat i mot de pas que li indica el client, i tramet els tres documents codificats al servei SOAP.

Quan rep d'aquest l'autorització en base64, la descodifica, n'extrau el document de la signatura i el mostra al client. Quan el client l'accepta, l'envia al PS per poder accedir al recurs.

Un problema trobat en aquesta etapa ha estat que les codificacions i descodificacions en base 64 dins els servidors les havia implementat amb classes internes de Sun (`sun.misc.BASE64Encoder` i `BASE64Decoder`) que sota aquestes condicions treballen sense problemes. Quan les vaig posar al codi de l'*applet*, feien saltar l'*AccessController*, pel què vaig haver de recórrer a la classe externa *Base64.java* oferida per *Sourceforge* per fer aquesta tasca.

L'esquema ideal seria doncs:

Com abans, la cronologia de la implementació ha fet que el codi de generació quedi dins del de l'*applet*. Queda doncs segregar la classe com una millora.

Les dues classes finals s'empaquetaran dins un *jar* que ulteriorment se signarà per permetre la lectura a disc del *keystore* indicat pel client.

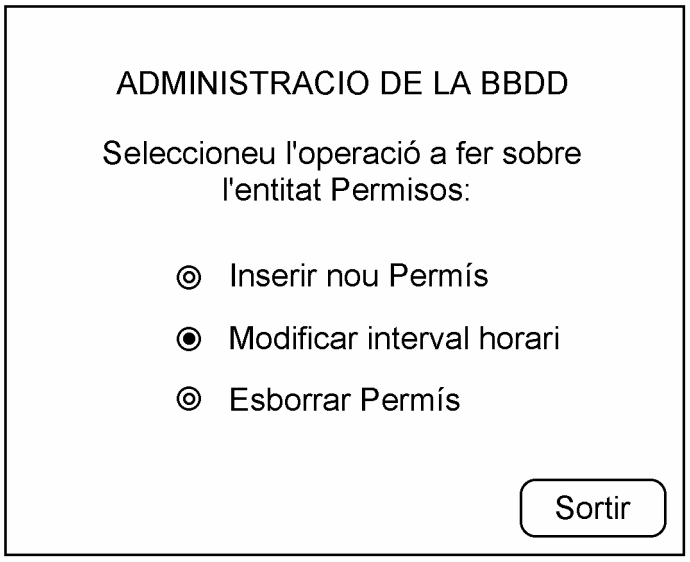
3.6.- Disseny de l'aplicació d'Administració de la BBDD

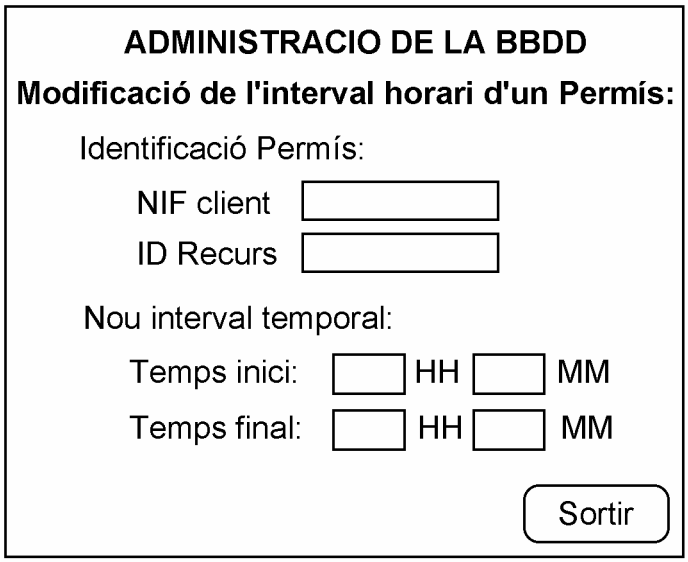
Finalment, a l'aplicació de gestió de l'administrador també hi caldrà una sèrie de *GUI*s amb menús que permetin dur a terme les diverses operacions amb la base de dades. Inicialment, he pensat en un menú principal (tipus formulari amb botons) que cridi altres menús secundaris encarregats de dur a terme cada cas d'ús específic, que també poden ser formularis amb botons de ràdio que ofereixin les diverses opcions de cada cas d'ús, així com la possibilitat de tornar al menú principal. Es disposarà així d'un arbre de formularis amb arrel al menú inicial i que es van cridant consecutivament en funció de l'opció triada fins arribar a les fulles finals, on es demanen les dades i s'envia al servlet la transacció a executar.

Aquests formularis finals hauran de tenir un botó per validar l'operació i un per cancel·lar-la, que portarà al menú inicial.

En validar, el formulari enviarà al servlet de l'aplicació d'administració els codis de l'operació sol·licitada i la taula sobre la què s'ha d'efectuar, a més de les dades necessàries per dur a terme la transacció. Amb elles, el servlet generarà el codi SQL i mitjançant el pont JDBC l'enviarà al SGBD per a què l'executi i en retorni el resultat. S'haurà doncs de retornar de retornar una pantalla amb un text on es pugui veure el resultat de l'operació (si s'ha fet amb èxit o si hi ha hagut algun problema).

Un possible disseny dels formularis seria:





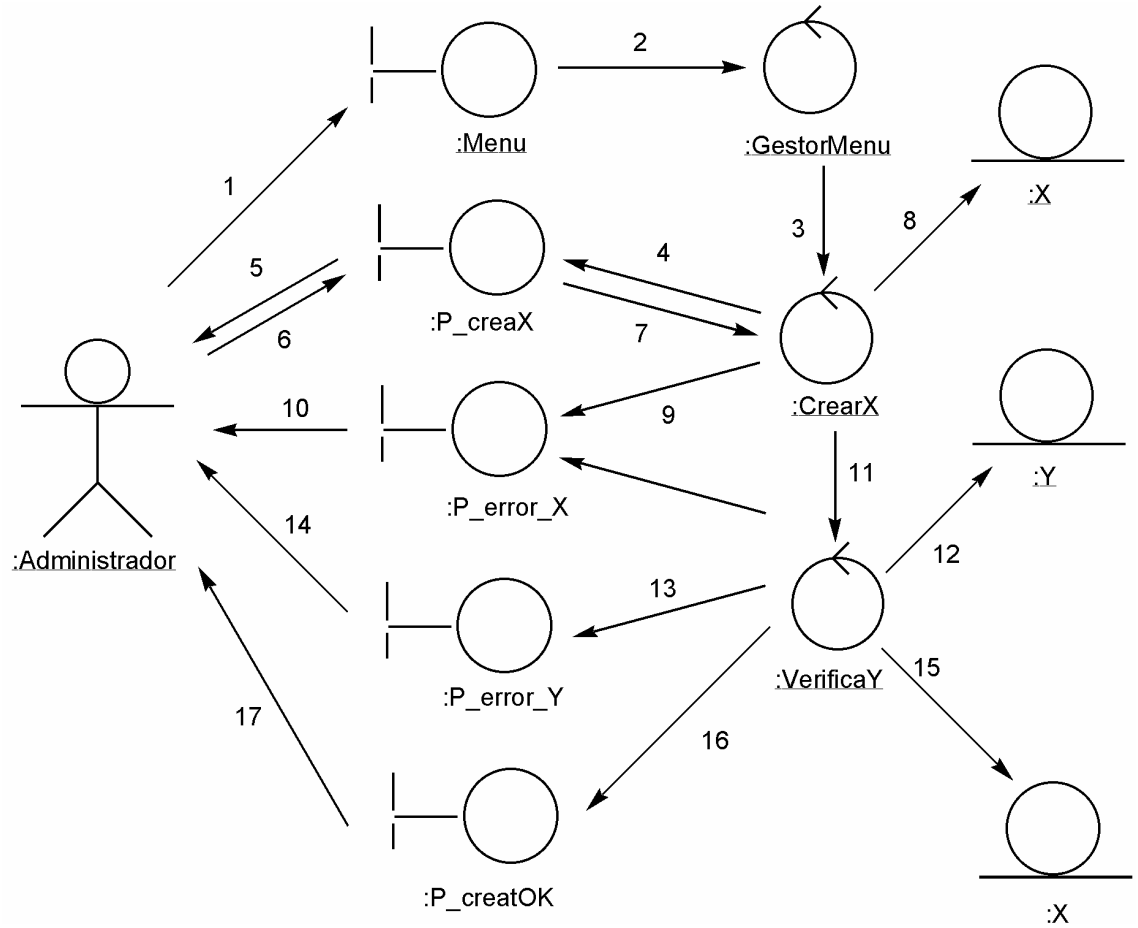
La implementació d'aquestes aplicacions de fet implica tres operacions bàsiques d'*update* (afegir, modificar i esborrar) de les quals es pot fer un diagrama comú d'interfície amb l'usuari vàlid per a qualsevol entitat.

Quant a l'operació de consulta, he decidit oferir dues possibilitats:

- Una consulta individual, on en base de la clau primària i/o secundària de l'entitat es consulta si existeix i es donen tots els seus atributs.
- Una consulta completa de tota la taula, on es llisten totes les entitats, juntament amb llurs atributs.

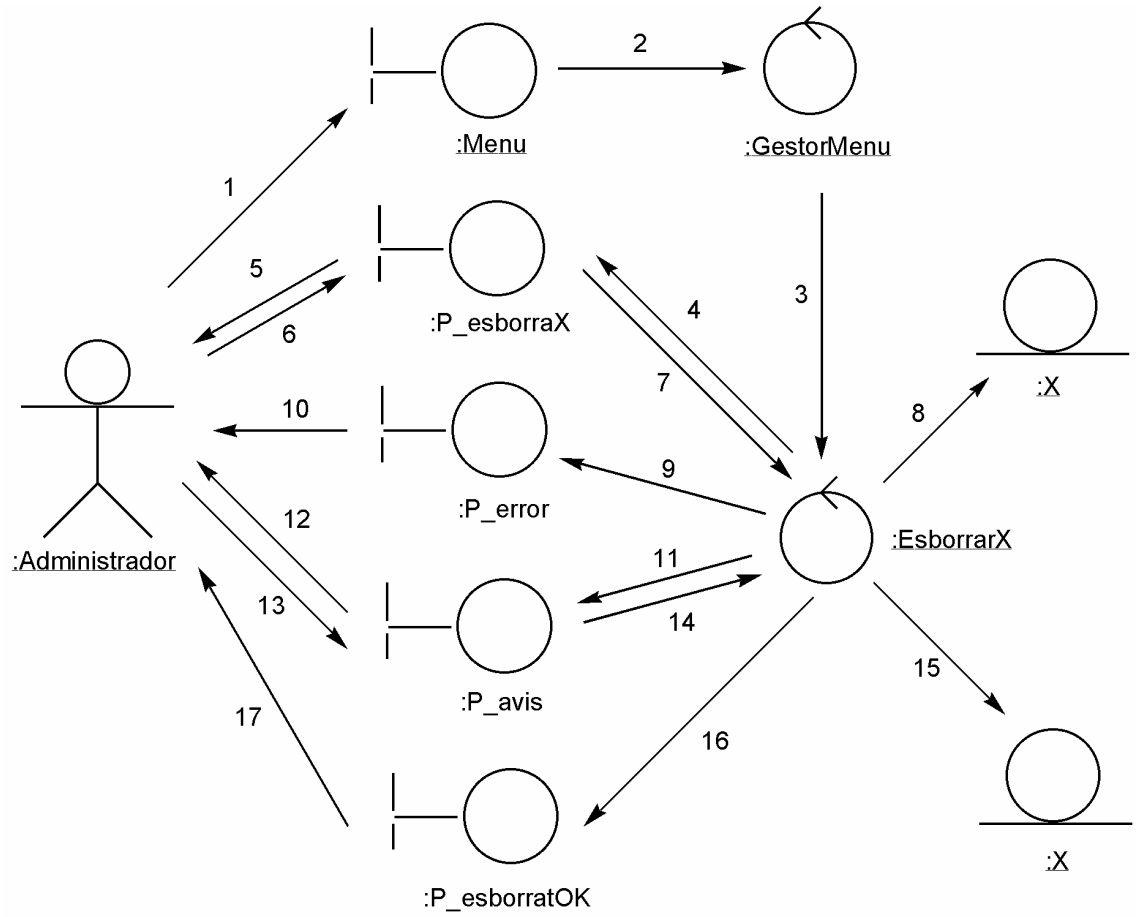
Podem pensar una mica com serien els diagrames de les interfícies d'aquestes operacions, en especial les de modificació de dades.

Pel cas de l'operació afegir (que es pot aplicar a qualsevol entitat X de les quatre considerades) proposo una seqüència que tingui en compte si a l'hora de crear una entrada X en una taula s'ha de comprovar que ja existeixi una referència en una taula Y:



1, 2, 3, 4: Crea X
5: Demana dades
6, 7: Dóna dades
8: Consulta
9: Error (existeix X)
10: Informa
11: Crea X
12: Consulta Y
13: Error (no existeix Y)
14: Informa
15: Crea X
16: Creat
17: Informa

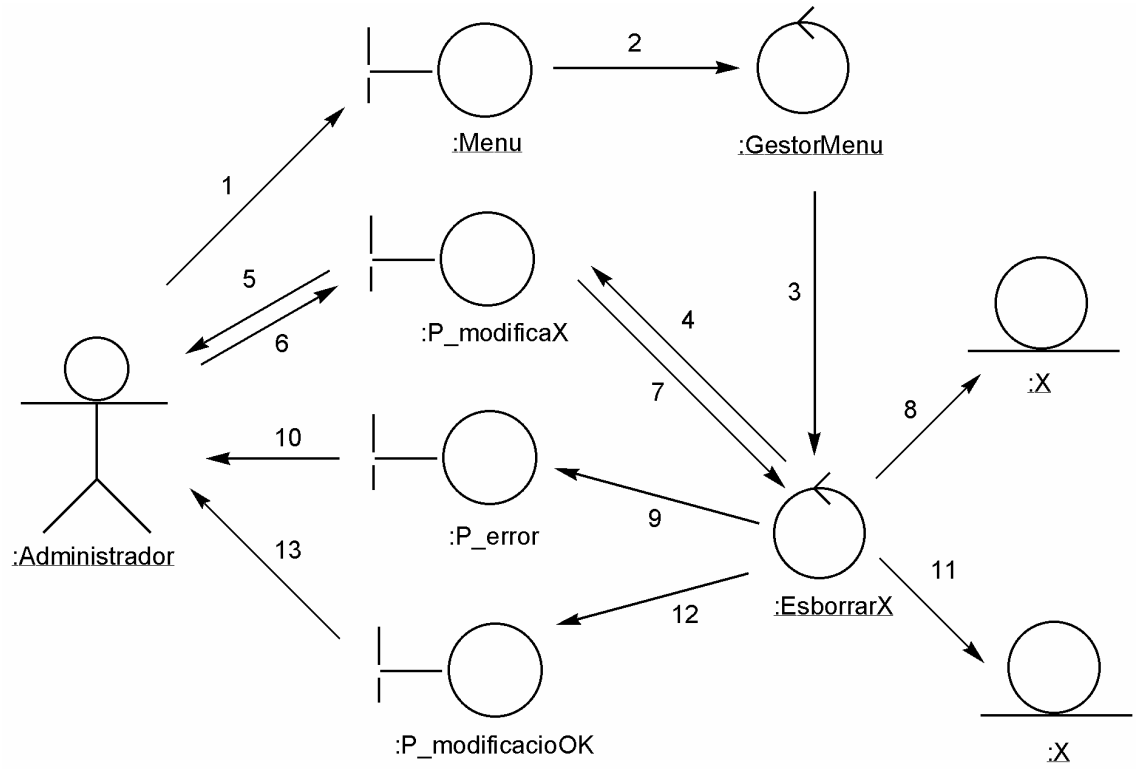
En el cas de l'esborrat tindríem:



1, 2, 3, 4: Esborra X	9: Error (no existeix X)	15: Esborra
5: Demana dades	10: Informa	16: Esborrat
6, 7: Dóna dades	11, 12: Confirmar esborrat	17: Informa
8: Consulta	13, 14: Confirmació esborrat	

Per simplificar la programació, es pot fer que en els casos d'esborrat la BBDD actuï en mode *cascade*, és a dir, que si s'esborra una entrada en una taula s'esborrin també totes les entrades a d'altres taules que la referenciïn.

Quant a la modificació, que només seria d'aplicació a l'interval temporal de l'entitat Permís, el diagrama seria:



1, 2, 3, 4: Modificar X	8: Consulta	11: Modifica X
5: Demana dades	9: Error (no existeix X)	12: Modificat X
6, 7: Dóna dades	10: Informa	13: Informa

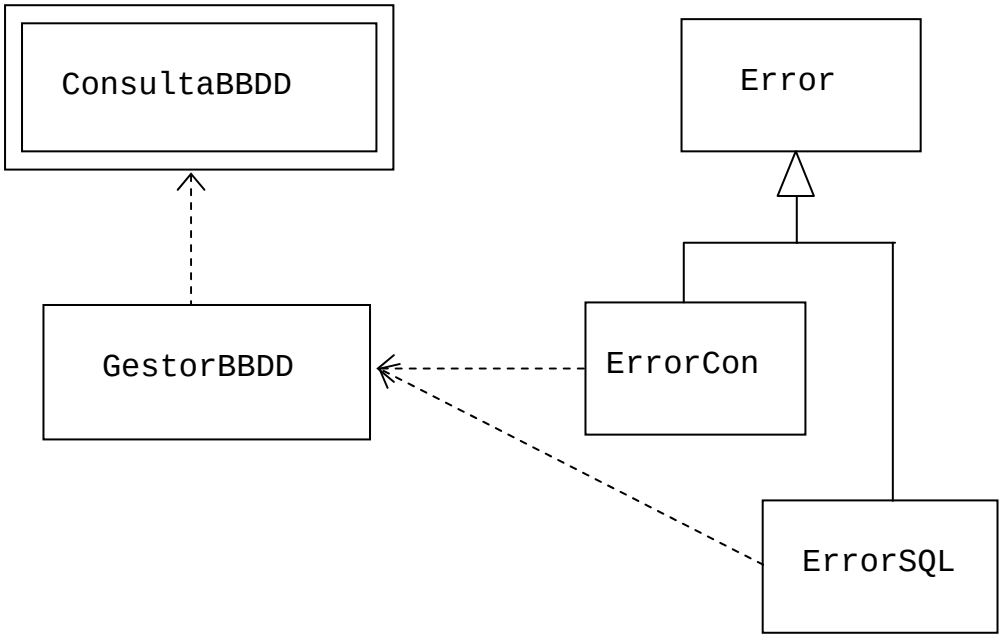
Les operacions de consulta/llistat implicarien només una petició SELECT a la base de dades i mostrar els *ResultSet* resultants a l'administrador, pel què el seu diagrama seria elemental.

Finalment, he optat per NO implementar les consultes prèvies a la BBDD abans de les operacions d'inserció, esborrat i modificació. He configurat la base de dades de manera que el mateix sistema en sigui el responsable de la integritat i la coherència de la informació que conté: recull com a obligatòries les relacions prèviament esmentades entre les claus de les diferents taules i provoca esborrats en cascada quan la clau primària o secundària de l'objecte a esborrar estigui apuntada per una.

Quant a la transferència d'informació, he pensat de responsabilitzar-ne d'aquesta tasca a les següents classes:

- Un servlet que rep com a paràmetres els codis d'operació i taula efectuada, així com els necessaris per localitzar les entrades afectades, ho transforma en una petició SQL i la passa a una classe auxiliar que gestiona la consulta amb l'SGBD. En rebre la resposta, la mostra al navegador.
- Una classe gestora que rep la petició, s'encarrega de trametre-la a la BBDD, rebre'n la resposta i retornar-la. Per evitar problemes en aquesta comunicació, he generat dues classes hereves d'*Error* que gestionen els problemes de connexió i transacció i eviten la caiguda del sistema.

Ens quedaria doncs un *package* com:



CAPÍTOL IV : IMPLEMENTACIÓ

Com he esmentat anteriorment, en el procés d'implementació he procurat utilitzar sempre les classes java de *Sun*, evitant fer ús d'*Axis*: crec que això permet que el client no hagi d'importar llibreries noves per poder executar l'applet, i que en tingui prou amb el paquet estàndard *SE 1.6* per poder dur a terme tota la funcionalitat d'aquest.

En les classes residents al servidor aquest problema no és tan crític, però he intentat seguir la mateixa metodologia, encara que no puc assegurar si *Tomcat* treballa internament fent crides a llibreries *Axis* en algun moment.

Quant a l'infraestructura PKI, he utilitzat exclusivament *OpenSSL* per generar l'arbre de certificats i la CRL de la CA, l'eina *keytool* de *Sun* per generar el *keystore* pel certificat SSL del servidor, i *jarsigner* per signar l'*applet*.

Dit això, esmentaré tot seguit els punts que he cregut més interessants dels diversos codis que adjunto a aquesta memòria dins el directori `src`.

4.1.- Aplicació web ProServ

Resideix en el servidor del Proveïdor de Serveis i és l'encarregada de generar la petició, servir l'*applet* signat, recollir-ne l'autorització del PI, i cas que sigui positiva servir la descàrrega del recurs demanat. Per fer-ho, consta d'un formulari web que recull l'identificador del fitxer sol·licitat (`inici.html`) i d'un *package* anomenat `tfc.provserv` que conté les següents classes:

4.1.1.- Classe *GenPeticio.java*

Aquest *servlet* rep del formulari de petició el paràmetre identificador del recurs sol·licitat i genera la petició XML. Demana la *Date* actual del sistema per generar l'identificador de la petició (atribut `Id` del node `peticio`) i omplir els camps de temps. La resta de dades són pròpies del PS.

Tot seguit signa de forma *enveloped* la petició, prenent com a *Reference* l'identificador de la petició, pel què només se signarà aquest node (i també permetrà reunificar petició i la signatura *detached* que se li extraurà en una *enveloped* que es podrà verificar).

Per signar extrau del subdirectori `pki_PS` el PKCS12 del PS i el mot de pas de la clau privada. A continuació, grava la signatura completa al directori `xml_PS` per tenir-ne constància. Tot seguit, n'extrau el node `signature` en forma de document, que serà la signatura *detached*.

Transforma petició i signatura en cadenes base64 i les inclou al codi HTML de sortida com a paràmetres de l'applet signat que es carregarà al navegador del client.

Aquest codi és una *FORM* que inclou l'activació de l'*applet* signat i que, un cop aquest s'ha executat, permetrà cridar a petició del client el mètode *doPost* del servlet *ProcessaSigPI* al qual, mitjançant la funció *passaValor()*, li enviarà l'autorització codificada en base64 que ha retornat l'*applet* per al seu procés.

4.1.2.- Classe *ProcessaSigPI.java*

Aquest *servlet* rep del formulari generat a la classe anterior la cadena base64 amb l'autorització del PI signada *enveloping*. La transforma en document i la grava al directori `xml_PS` per tenir-ne constància.

Tot seguit, la verifica amb l'ajut de la classe *VerifyEnveloping* i n'extrau els nodes `resultat` i `recurs`. Si el primer val `cert`, inicia la descàrrega del recurs sol·licitat, guardat al directori `rec_PS`. Cas contrari, mostra una plana HTML amb el motiu de la denegació de la sol·licitud.

4.1.3.- Classe *VerifyEnveloping.java*

Aquesta classe auxiliar ofereix el mètode:

- Verifica(*String* cacertsDir, *Document* sigAuth): *String* cn

on cacertsDir es el subdirectori on el PS guarda el certificat de la CA del PI (al nostre cas és `/pki_PS/trusted`), i sigAuth l'autorització *enveloping* del PI, a partir de la qual n'extrau el CN del signatari, verifica la signatura i retorna el CN llegit si la verificació és correcta, o bé la cadena "`error`" en cas contrari.

Per dur a terme la seva tasca, utilitza una classe estàtica interna anomenada *KeySelector* encarregada de buscar el certificat de la CA emissora en el directori que s'ha donat com a paràmetre gràcies al mètode:

- buscarCert(*String* iss, *String* trustedDir): *X509Certificate* caCert

on trustedDir es el subdirectori on el PS guarda el certificat de la CA del PI (al nostre cas és `/pki_PS/trusted`), i iss és el nom del *X500Principal* del signatari del certificat de la signatura. Aquest certificat es valida contra el de CA i es contrasta amb la *CRL* de la CA que he emprat per estendre tots els certificats, i que he dipositat al directori `/webapps/CA_RCOSTAS` de Tomcat.

4.1.4.- Classe *PSUtils.java*

Ofereix a les anteriors els mètodes de codificació i descodificació per passar de documents XML a cadenes base64 i a l'inversa:

- *b64toDoc*(*String* cadenaBase64) : *Document* docXML
- *docToB64*(*Document* docXML) : *String* cadenaBase64

Utilitza els descodificadors interns de *Sun*, i per tant no es pot executar dins un *applet* ni tan sols si és signat.

4.2.- Component del Client

Resideix en el directori `ProServ` de l'aplicació web del Proveïdor de Serveis i consisteix en un fitxer `jar` comprimit i signat pel PS (`SignedApplet.jar`), que conté l'*applet* format perla classe hereva de *JApplet* anomenada *IntextApplet* i per la classe auxiliar lliure *Base64*, oferta per *Sourceforge* com a programari lliure.

Cal recordar que quan un *applet* s'executa al navegador no pot accedir als recursos del sistema, a menys que estigui signat i la JVM reconegui el certificat o bé l'usuari l'accepti explícitament.

La inclusió de la classe auxiliar és deguda, com s'ha comentat abans, al fet que els codificadors interns de *Sun* no es poden invocar dins un *applet*, i per això cal incloure-la en el `jar` signat.

4.2.1.- Classe *IntextApplet.java*

Quan es carrega aquest *applet* al navegador rep com a paràmetres la petició i la signatura del PS codificats en base64.

El primer que fa és descodificar ambdós document i mostrar la petició en un camp de text, on el client la pot consultar. A l'esquerra d'aquesta, apareixen dos camps de text on el client hi ha d'introduir la ruta absoluta del seu *keystore* PKCS12 i el mot de pas de la seva clau privada. Quan ho ha fet, s'activa un botó que es torna vermell i que permet al client endegar el procés de petició de l'autorització al PI.

En pitjar-lo, el codi crida el seu mètode:

- signa(*Document* peticio, *String* ubicacioP12, *String* password) : *String* sigCLI64

Que de forma similar a la implementació de la classe *GenPeticio* obté la signatura *detached* de la petició extraient-la de l'*enveloped* que genera, i tot seguit la codifica en base64.

Tot seguit, i després d'emetre un avís a la finestra, s'invoca el mètode:

- envia(*String* peticio64, *String* sigPS64, *String* sigCLI64) : *String* auth64

que remet petició i signatures codificades en base64 al servei del PI i en rep com a retorn l'autorització també codificada.

Per fer aquesta funcionalitat, el mètode *envia* genera un les capçaleres d'un missatge SOAP i en pobla el cos amb un node que conté la invocació al mètode *autoritza* del servei *DemanaAutoritzacio* del PI i la URL d'aquest. Del node hi penja tres nodes de text que contenen les cadenes codificades de petició i signatures, que actuaran com a paràmetres del mètode. Després envia el missatge cridant pel port *HTTPS* el servei i n'espera la resposta. Quan aquesta arriba, n'extreu la cadena amb l'autorització signada *enveloping* codificada.

En el retorn d'*envia* l'*applet* copia la cadena en l'atribut intern `sigAuth64`, la descodifica al document signatura i d'aquest n'extreu el node autorització, que mostra a la finestra.

L'*applet* també ofereix el mètode:

- *getAuth64*(): *String* ath64

per permetre passar la cadena codificada com a paràmetre al *servlet ProcessaSigPI* del proveïdor de serveis.

Finalment, dir que al codi de l'*applet* hi he inclòs els mètodes auxiliars:

- docToB64(*Document* doc): *String* docB64
- b64ToDoc(*String* docB64): *Document* docB64

que s'encarreguen de fer les codificacions i descodificacions amb l'ajut dels mètodes oferts per la classe de programari lliure *Base64*, així com el mètode:

- docToFormattedString(*Document* doc): *String* textImprimible

que permet que el contingut del document XML es pugui visualitzar a la finestra de text ben indentat.

4.3.- El servei DemanaAutoritzacio

Resideix en el servidor del Proveïdor d'Identitat i és l'encarregat de rebre el missatge SOAP amb la petició i les dues signatures i retornar una resposta amb la codificació base64 de la signatura *enveloping* de l'autorització. Per satisfer aquesta funcionalitat, he implementat les següents classes que he empaquetat en el fitxer `DemanaAutoritzacio.aar` juntament amb el descriptor del servei:

4.3.1.- Classe *DemanaAutoritzacio.java*

És el punt d'entrada i sortida del servei, i es limita a rebre els tres paràmetres encapsulats dins la petició SOAP, trametre-la al mètode *autoritza* de la classe *GenAutoritzacio* per a que els processi, rebre el retorn d'aquest amb la codificació base64 de la signatura *enveloping* de l'autorització i reenviar-lo al peticionari del servei dins una resposta SOAP.

4.3.2.- Classe *GenAutoritzacio.java*

Ofereix el mètode:

- autoritza(*String* peticio64, *String* sigPS64, *String* sigCL64): *String* auth64

Que en rebre la petició i les dues signatures les descodifica a *Document* i les verifica amb una crida al mètode *verifica* de la classe *VerifyDetached,* que inclou el directori on ha de buscar el certificat de la CA. (`pki_PI/trusted`).

Si la verificació és correcta, rep com a retorn el CN de cada signatari, que li permet identificar el client. Tot seguit extrau de la petició l'identificador d'aquesta, el recurs sol·licitat i l'instant en què es generà, i consulta directament a la BBDD via pont JDBC si existeix aquest permís pel client que ha signat i en quin horari es pot fer la sol·licitud.

Tot seguit, compara les dades retornades i, si tot és correcte, genera l'autorització amb el text "`cert`" dins el node "`resultat`". Cas contrari, hi escriu "`fals`" i la causa de la negació.

Un cop s'ha generat el document, el signa en forma *enveloping* amb la crida a l mètode *signa* de la classe *SingEnvping* que inclou la ubicació del *keystore* i el mot de pas (ambdós al directori `pki_PI`), desa una còpia de la signatura retornada al directori `xml_PI` per tenir-ne constància, i retorna la signatura que conté l'autorització prèviament codificada en base64.

Dins la classe hi ha implementats els mètodes interns *b64ToDoc* i *docToB64* similars alls que ofereix *PSUtils*, malgrat que per la manca de temps no els he segregat com a classe auxiliar ni he utilitzat la classe externa *Base64*.

4.3.3.- Classe *VerifyDetached.java*

Ofereix el mètode:

- verifica(*String* caCertsDir, *Document* peticio, *Document* sig): *String* cnErr

El mètode extreu el CN del signatari de la signatura *detached* i reconstrueix la signatura *enveloped* original en base als dos fragments. Tot seguit la verifica i, si és correcta, retorna el CN del signatari. Cas contrari, retorna la cadena "`error`".

Com a la classe *VerifyEnveloping*, he disposat la classe interna *X509KeySelector* que verifica el certificat i el contrasta contra la CRL de la CA emissora (que descarrega d'un servidor públic prèviament predefinit), així com el mètode *buscarCert*, que és l'encarregat de trobar el certificat arrel de la CA dins el directori indicat com a paràmetre.

4.3.4.- Classe *SignEnvping.java*

És força similar a la classe :

- signa(*Document* aut, *String* pkcs12File, *String* pass): *Document* sigAuth

el codi del qual és força similar al contingut a l'*applet* i a *GenPeticio* per signar en forma *enveloped*. De fet, amb més temps m'hauria agradat segregar d'aquests codis una classe *Verifica* amb dues classes hereves *SignaEnveloped* i *SignaEnveloping*, però al final ho deixo com a millora a fer.

La diferència principal rau en la *Reference* utilitzada i en la creació del node `Object` que conté l'autorització embolcallada.

4.4.- Aplicació web ProId

Resideix en el directori `webapps` del servidor del Proveïdor d'Identitat, i és l'encarregada de gestionar les consultes i modificacions a la BBDD del PI que li sol·licitarà de forma remota l'administrador des d'un navegador.

D'una banda, consta d'un conjunt de formularis web estructurats en forma d'arbre on, partint del node `menuAdmin.html` es va accedint successivament a una sèrie de submenús que, en arribar a les fulles, recullen les dades necessàries per fer una determinada operació (consultar, inserir o esborrar i, al cas dels permisos, modificar) en una determinada taula (clients, proveïdors, recursos o permisos), o bé simplement llistar tot el contingut de la taula. Quan l'administrador ompli els camps necessaris i premi el botó, aquest formulari cridarà al *servlet ConsultaBBDD* i li passarà com a paràmetres els codis de l'operació sol·licitada i de la taula sobre la qual ha de ser efectuada, a més dels arguments necessaris per realitzar la transacció SQL. En tot instant, cada formulari permet que l'administrador retorni al menú inicial.

D'altra banda, el processat de les dades el duu a terme un *servlet* que requereix l'ajut d'una classe auxiliar i dues classes que gestionen els errors que es puguin derivar de la comunicació amb la BBDD. Les quatre classes es troben empaquetades en el *package* `tfc.proid`. anem a veure-les amb una mica més de detall:

4.4.1.- Classe *ConsultaBBDD.java*

Aquest *servlet* rep els diversos paràmetres abans esmentats del formulari final i a partir dels dos codis i els paràmetres genera la sentència SQL que caldrà enviar a la BBDD.

A partir d'aquest punt, es crea una nova instància de la classe *GestorBBDD* i se li sol·licita que obri la connexió amb la BBDD (que he anomenat `adminDB`).

Si el codi d'operació es correspon a una consulta, es passa la sentència al mètode *executaConsulta*, del qual es rebrà un *ResultSet*. Si és una *update*, es crida el mètode *executaModificacio*, que retorna un enter.

La part final del servlet es dedica a capturar el retorn, formatar-lo i passar el resultat en forma de plana HTML al navegador de l'administrador després d'haver tancat la connexió amb la BBDD.

4.4.2.- Classe *GestorBBDD.java*

Ofereix quatre mètodes, dos dels quals:

- obreConnexio (*String* nomBBDD): void.
- tancaConnexio() : void

simplement criden el pont JDBC per establir (creant els objectes *Connection* i *Statement*) i tancar la connexió amb el SGBD (*Derby*).

Els mètodes.

- executaConsulta(*String* querySQL): *ResultSet*
- executaModificacio(*String* updateSQL): *int*

simplement remeten la consulta o la modificació al SGBD i en retornen el resultat o bé recullen l'error si es produeix.

4.4.3.- Les classes *ErrorCon.java* i *ErrorSQL.java*

Aquestes classes estenen la classe java *Error*, del qual en sobreescriuen el constructor, i simplement s'utilitzen per distingir el tipus d'error que es pugui donar durant la connexió al SGBD i poder-lo visualitzar a consola gràcies al mètode:

- toString ():*String*

4.5.- Infraestructura PKI

Com s'ha esmentat abans, per poder provar el sistema he generat una petita cadena de certificats amb l'ajut de l'eina *OpenSSL*. He creat doncs un certificat arrel de CA (`certCA.crt`, amb CN = `CA_rcostas`) a partir del qual he generat els certificats del proveïdor de serveis (`cert_PS.pem`, amb CN = `PROV_SERV`), del proveïdor d'identitat (`cert_PI.pem`, amb CN = `PROV_PI`), del client (`cert_user.pem`, amb CN = `USER_RCOSTAS`). També he generat un segon certificat pel client (`cert_userBad.pem`, amb CN = `USER_DOLENT`), que he revocat amb la creació d'una *CRL* (`ca.crl`), dipositada al directori de *Tomcat* `/webapps/CA_RCOSTAS/` per permetre'n la descàrrega externa.

Tots aquests certificats s'han dipositat als respectius directoris `pki_XXX` de *Tomcat*, que alhora incorporen un subdirectori `trusted` que conté el certificat arrel de la CA (en casos reals, podrien ser diferents).

Val a dir que per a cada certificat s'ha generat una parella de claus RSA de 1024 bits (en situació real haurien de ser de 2048 i la de CA potser més), una petició PKCS10 (excepte pel certificat arrel, que és auto-signat), així com la *keystore* PKCS12 i el certificat respectius per separat.

Comentar finalment que ha calgut un certificat d'específic per l'intercanvi SSL segur al servidor (`cert_localhost.pem`) amb CN = `localhost`, que en situació real hauria de ser el mateix que el de cada proveïdor i contenir al CN l'adreça web d'aquest (cas contrari, el navegador genera un *warning* per la no coincidència del nom del servidor accedit via HTTPS i el CN del certificat SSL). Per a què aquest certificat SSL de *localhost* funcioni correctament he hagut de traduir el *keystore* a format *JKS* amb l'ajut de l'eina *keytool* (nou nom `localhost.jks`, dipositat a `/pki_SSL`). El certificat de CA també s'ha incorporat al *keystore* `cacerts` a la JRE del client i als navegadors del client i de l'administrador (que en les proves era el mateix) a fi que l'execució de l'*applet* i l'establiment del protocol HTTPS siguin automàtics.

CAPÍTOL V: INSTAL·LACIO I ÚS DEL SISTEMA

En la implementació i proves de funcionament del producte d'aquest TFC s'ha emprat un ordinador portàtil Dell Inspiron amb sistema operatiu *Windows2000 Professional SP4*. Tot el temps he treballat com a usuari administratiu.

La programació s'ha fet en llenguatge Java, utilitzant els darrers paquets oferts directament per *Sun*, concretament el *SDK* v5.05 de *J2EE*.

El servidor emprat ha estat *Tomcat* en la versió 6.0.18 de la fundació *Apache*, que ha requerit el suport suplementari del paquet *Axis2* de la mateixa fundació.

El SGBD triat ha estat *Derby* en versió 10.4.2, un servidor de BBDD lleuger i basat en Java també de la fundació *Apache*.

La PKI s'ha generat amb l'eina *OpenSSL*, malgrat que algunes funcions específiques (traducció i incorporació de certificats a *keystores* i signatura d'*applets*) s'han dut amb l'ajut de les eines *keytool* i *jarsigner* del *SDK* de Java.

Per a les proves s'han emprat els navegadors *Internet Explorer* v6 i *Firefox*. Malauradament, *IE6* presenta problemes en l'execució de l'*applet*, concretament en el pas de paràmetres entre aquest i el *servlet*, que no es presenten en la versió 7 en la qual ho va provar el consultor al seu ordinador. Malauradament, la versió 7 no es troba disponible per W2K.

El navegador *Firefox* no presenta aquests problemes, però en ocasions no interpreta bé certs caràcters no-*ASCII* del codi HTML. No he corregit aquest comportament (cal substituir la codificació) per manca de temps a refer les planes estàtiques i dinàmiques.

5.1.- INSTAL·LACIO

5.1.1.- Preparació del sistema operatiu, l'entorn i el programari

El primer que caldria fer abans d'instal·lar el sistema és configurar les variables d'entorn dels sistemes operatius on residiran els servidors dels proveïdors i el SGBD del proveïdor d'identitat (que al nostre cas són el mateix).

Ja que el sistema incorpora tecnologies de servidor *web*, ha calgut treballar en la part dels proveïdors amb la versió *Enterprise* de Java. Així doncs, cal instal·lar el SDK v5.05 de J2EE, que inclou el JDK v1.6.0_06. Fet això cal definir pel sistema i per l'usuari les variables d'entorn `JAVA_HOME`, `J2EE_HOME`, `JRE_HOME` i `J2RE_HOME` en funció d'on s'ha fet la instal·lació.

Tot seguit cal baixar de la web d'*Apache* el fitxer ZIP *Tomcat v6.0.18* i instal·lar-lo a partir de la descompressió d'aquest. En el fitxer `\bin\catalina.bat` cal incloure la variable d'entorn `JAVA_HOME`, i finalment comprovar que les instruccions *startup* i *shutdown* funcionin bé.

Per facilitar les compilacions durant la implementació, he copiat les llibreries `servlet-api.jar`, `jsp-api.jar`, `el-api.jar` i `tomcat-dbcp.jar` del directori `/lib` de tomcat al directori `jre/lib/ext` del *JDK*. Entenc que això no cal en la instal·lació final, només si es vol modificar el codi.

Tot seguit cal instal·lar el paquet *Axis2* al contenidor *Tomcat* per permetre el funcionament del servei *SOAP* de l'aplicació. Per fer-ho, es descarrega de la web d'*Apache* la distribució *Web Archive* (*war*) en versió 1.4.1, es descomprimeix i s'arrossega sobre el directori *webapps* de *Tomcat*. He vist que en aquest punt és millor reiniciar el servidor (*shutdown* + *startup*), i en general cal fer-ho sempre que s'introdueixen modificacions al servidor per a què els afegits que s'hi faci funcionin correctament.

La instal·lació de *Derby* comença descarregant de la web d'*Apache* el fitxer ZIP que en conté la versió binària 10.4.2, i tot seguit descomprimir-la. A continuació cal definir en la finestra on s'executarà el servidor la variable `DERBY_INSTALL` que apunti a la carpeta arrel de *Derby*. Es pot provar el seu funcionament arrencant-lo en mode *embedded*, que permet accés directe a la base de dades amb comandes. Cal col·locar-se al subdirectori `/bin` i executar l'*script* `setEmbeddedCP.bat`. Es comprova que tot funcioni amb la comanda: `java.org.apache.derby.tools.sysinfo`, la qual ha de mostrar les variables del sistema operatiu i de *Derby*.

Addicionalment, i per a què el sistema reconegui els certificats de la PKI cal instal·lar el fitxer arrel de CA `certCA.crt` a:

- A les carpetes `/trusted` dels subdirectoris `/pki_PS` i `/pki_PI` (on ja hi és dipositat) per verificar els certificats de les signatures.
- Al navegador del client i administrador, per a què admeti com a bons el certificat del servidor en establir el protocol HTTPS.
- A la màquina virtual de Java incorporant-lo al *keystore* `cacerts` que hi ha al subdirectori `/jre/lib/security` de la instal·lació local de Java. Això permetrà que l'*applet* es pugui executar automàticament (*IE6*) o amb la mínima intervenció (*Firefox*).

També caldrà que el client dipositi en algun lloc del seu sistema la carpeta `pki_CLI`, a fi que disposi del *keystore PKCS12* amb el qual haurà de signar la petició abans d'enviar-la al proveïdor d'identitat.

A la carpeta hi ha dipositats un *keystore* amb certificat vàlid (`usuari.p12`, amb mot de pas `tfc_user`) i un de revocat (`usuariBad.p12`, amb mot de pas `tfc_userBad`) per fer-ne proves.

Finalment, caldrà dipositar la carpeta `CA_RCOSTAS` directament al directori webapps de *Tomcat*, ja que conté la CRL de la CA (`ca.crl`), que hauria de residir en un servidor independent, i amb accés lliure (la carpeta ja es troba al lloc adient del lliurament).

5.1.2.- Configuració del SGBD *Derby*

Cal carregar al servidor l'estructura de la base de dades a utilitzar pel sistema i un contingut mínim. Per això s'obre una finestra de comandes *MS-DOS* i es va fins el subdirectori `/bin`. Allà es defineix la variable `DERBY_INSTALL` i s'arrenca el servidor en mode *embedded* executant l'*script* `setEmbeddedCP.bat` i es crida l'eina `ij`. Des d'ella s'executa l'*script* `crea_adminDB.sql` (que he dipositat al mateix subdirectori `/bin`) amb l'ordre: "`ij> run 'crea_adminDB.sql';`" per crear la BBDD que utilitza el sistema (anomenada `adminDB` i que queda resident al subdirectori `/bin`). Amb la mateixa eina es pot comprovar que tot és correcte fent-hi consultes.

Per a què la BBDD creada sigui accessible des de les aplicacions web del servidor, cal iniciar el servidor en mode *Network Server*. Per això, des d'una finestra de comandes diferent es defineix la variable `DERBY_INSTALL`, es carrega el `CLASSPATH` executant l'*script* `setNetworkServerCP.bat` i a continuació s'engega el servidor amb `startNetworkServer.bat`. Si tot ha anat bé, ens comunica que queda a l'escolta al port 1527.

En aquest mode, la BBDD es pot accedir bé des del pont *JDBC*, bé des del client de xarxa que s'activa amb `setNetworkClientCP.bat` i que amb l'ajut de l'eina `ij` permet comprovar que tot funcioni correctament.

5.1.3.- Configuració del servidor del Proveïdor de Serveis

Cal dipositar les carpetes `pki_PS`, `pki_SSL`, `xml_PS` i `rec_PS` al directori arrel de*Tomcat* (totes elles es troben al lloc adient del lliurament).

En la primera es guarden les claus, els certificats, el magatzem del proveïdor i llurs mots de pas per poder generar i verificar signatures. El segon, conté un certificat especial per l'intercanvi SSL del servidor. De fet, en condicions normal haurien de servir pel mateix propòsit els certificats de la carpeta `pki_PS`, però com en el producte lliurat els dos proveïdors comparteixen servidor i, a més, cadascun hauria de tenir el seu propi CN coincident amb l'adreça internet del servidor (que en el cas present ha de ser localhost) s'ha optat per aquesta solució temporal.

En la tercera carpeta s'emmagatzemaran les signatures de les peticions que envia i de les autoritzacions que rep per tenir-ne constància (en el nom de fitxer es recull l'identificador de petició), i a la quarta es guarden els recursos (fitxers de text) que el client pot sol·licitar.

També caldrà canviar el fitxer de configuració del servidor (`/config/server.xml`) per a què es pugui habilitar el port 8443 com a port *HTTPS* i s'indiqui la ubicació i el mot de pas del *keystore JKS* que permeti la negociació *SSL*. El fitxer modificat s'inclou en el lloc adient del lliurament.

Tota la funcionalitat del PS s'activa copiant la carpeta `ProServ` al directori `webapps` de *Tomcat* i reiniciant-lo. La carpeta ja conté el formulari d'inici per fer la petició (`inici.html`), l'*applet* signat (`signedApplet.jar`) i la carpeta `WEB-INF` on es guarden el descriptor de l'aplicació (`web.xml`) i les classes del *package* `tfc1.provserv` compilades.

Esmentar que en el descriptor, a més de definir els dos *servlets* i llurs *mappings*, aplica el grau de transport `CONFIDENTIAL` a la totalitat de l'aplicació, pel què només s'hi pot accedir pel port *HTTPS* (les peticions al 8080 hi seran redireccionades): això assegura un intercanvi segur i xifrat de les dades.

5.1.4.- Configuració del servidor del Proveïdor d'Identitat

Cal dipositar les carpetes `pki_PI`, `pki_SSL` i `xml_PI` al directori arrel de*Tomcat* (totes elles es troben al lloc adient del lliurament). La primera i segona tenen les mateixes funcions que al cas del proveïdor de serveis, i a la tercera es desaran les autoritzacions signades que s'emetin, en el nom de fitxer de les quals també hi consta quin proveïdor de serveis ha gestionat la petició. També cal permetre la comunicació *HTTPS* modificant de la mateixa manera que el proveïdor de serveis el fitxer `/config/server.xml`.

La càrrega i activació del servei *SOAP* s'aconsegueix fent un *uploading* del fitxer *Axis Archive* `DemanaAutoritzacio.aar`. Aquest, es troba dipositat a la carpeta del mateix nom inclosa al lliurament, i conté comprimits els executables de les classes del servei més el seu descriptor (el fitxer `services.xml` dins la carpeta `META-INF`, que conté el nom del servei i les operacions que ofereix). Per desplegar el servei cal anar a l'eina administrativa d'*Axis2* (http://localhost:8080/axis2/axis2-admin) i, després d'autenticar-se amb *login* i *password* (per defecte `admin` i `axis2` respectivament) clicar a *Upload Service*, introduir la ubicació del fitxer *aar*, pitjar el botó *Upload* i a continuació activar el servei amb el *link Activate Service*. Com sempre, millor reiniciar *Tomcat* un cop fet això.

Quant a l'aplicació d'administració de la BBDD, només cal copiar la carpeta ProId al directori webapps de *Tomcat*. La seva estructura és gairebé idèntica a l'esmentada per proveïdor de serveis, amb l'excepció que a l'arrel tenim tot l'arbre de pàgines *HTML* (amb inici a `menuAdmin.html`) amb formularis que permeten passar els paràmetres adients al *servlet*.

Com abans, al subdirectori `WEB-INF` es troben les classes del *package*, ara `tfc1.proid`, compilades i el descriptor de l'aplicació (`web.xml`) que també força l'accés pel port 8443.

En aquest cas, però, s'afegeix a aquest descriptor un *constraint* d'autenticació addicional (tipus `BASIC`) per a què només els usuaris reconeguts com a administradors de la base de dades (amb el `role-name bdadmin`) siguin els únics que hi poden accedir prèvia autenticació per usuari i mot de pas (que en efectuar-se sota SSL queden perfectament xifrats).

En aquest cas, cal definir quins són els usuaris autoritzats, i per això s'ha de modificar el fitxer `/conf/tomcat-users.xml`. Per això, se li assigna a l'administrador aquest paper, a més d'un nom d'usuari i un mot de pas: `username="adminDB" password="tfc_admin"` i `roles="bdadmin"`. Per poder accedir en mode administratiu al servidor, també s'ha definit un usuari amb les característiques: `username="tomcat" password="s3cret"` i `roles="manager"`

5.1.5.- Configuració del navegador i la JRE de client i administrador

Com s'ha esmentat abans, als navegadors del client i de administrador cal incorporar-hi el certificar de la CA, que permetrà admetre com a vàlid el certificat que li enviarà el servidor quan s'hi connecti pel port HTTPS.

A cada navegador dels provats, el procediment per carregar el fitxer `certCA.crt` és lleugerament diferent:

- En el cas de *IE6* cal seguir `Herramientas> Opciones de Internet> Contenido>Certificados> Entidades Emisoras Raíz> Importar`.
- Al *Firefox*: `Eines> Opcions> Avançat> Xifratge> Visualitza> Entitats> Importa`

Finalment, també cal que el client l'incorpori al seu *keystore* `cacerts` de la màquina virtual de Java que hi ha al subdirectori `/jre/lib/security` de la instal·lació local del client. Això permetrà que l'*applet* que se li descarregarà es pugui executar automàticament (*IE6*) o amb la mínima intervenció (*Firefox*). Per fer la importació cal posar el certificat al mateix directori de la instal·lació local i allà executar la instrucció:

```
keytool –importcert –trust cacerts –alias CArcostas –file certCA.crt –keystore cacerts
```

donant el mot de pas del magatzem (per defecte `changeit`).

5.2.- Instruccions d'utilització

5.2.1.- Administrador

Un cop ha configurat el seu sistema local i el navegador com s'indica al punt *5.1.5*, pot connectar-se de forma segura al proveïdor de serveis a l'adreça:

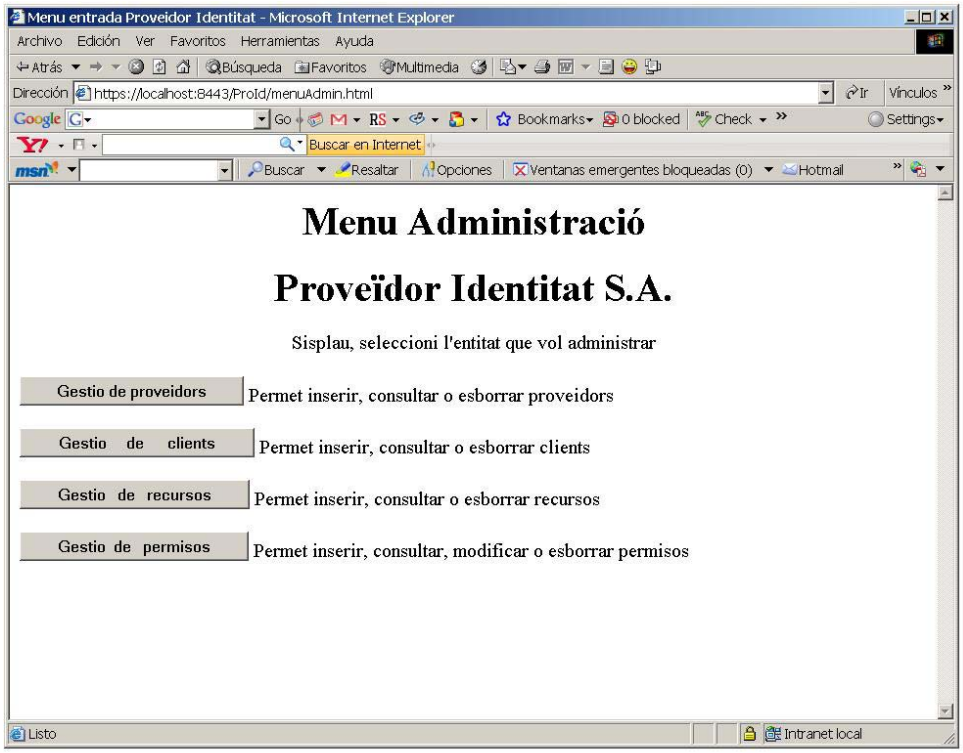
https://localhost:8443/ProId/menuAdministracio.html

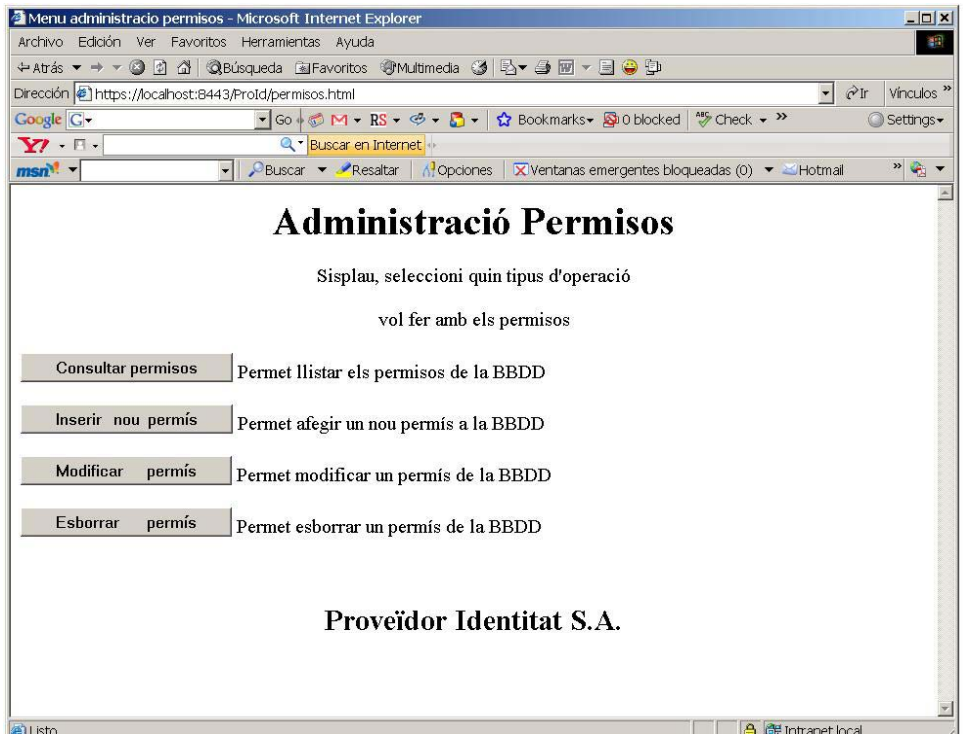
Cas que es vulgui connectar sense capa SSL:

http://localhost:8080/ProId/menuAdministracio.html

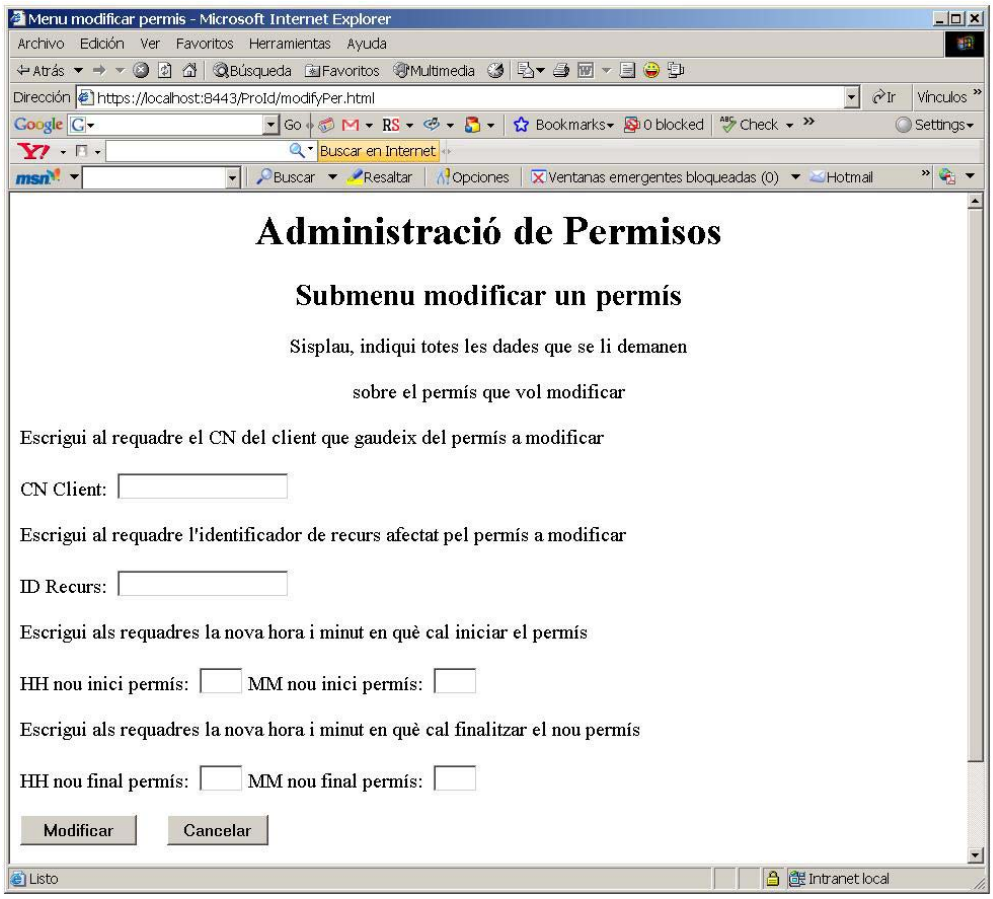
serà redirigit al port segur i la direcció efectiva serà la primera.

En obrir-se la plana, se li oferirà que seleccioni mitjançant botons sobre quina entitat vol actuar, i en prémer se li demanarà el tipus d'operació a fer:





Segons les opcions triades, arribarà a un menú final on se li demanarà les dades necessàries per efectura-la:



En funció de l'operació demanada i l'entitat afectada, se li oferirà un retorn amb la confirmació de l'operació (modificació) o bé un llistat (consulta):

5.2.2.- Client

Un cop ha configurat el seu sistema local i el navegador com s'indica al punt *5.1.5*, pot connectar-se de forma segura al proveïdor de serveis a l'adreça:

https://localhost:8443/ProServ/inici.html

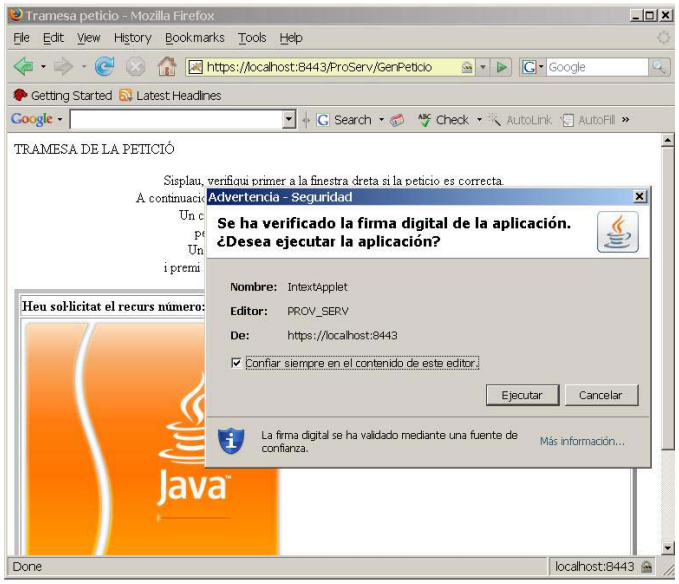
Cas que es vulgui connectar sense capa SSL:

http://localhost:8080/ProServ/inici.html

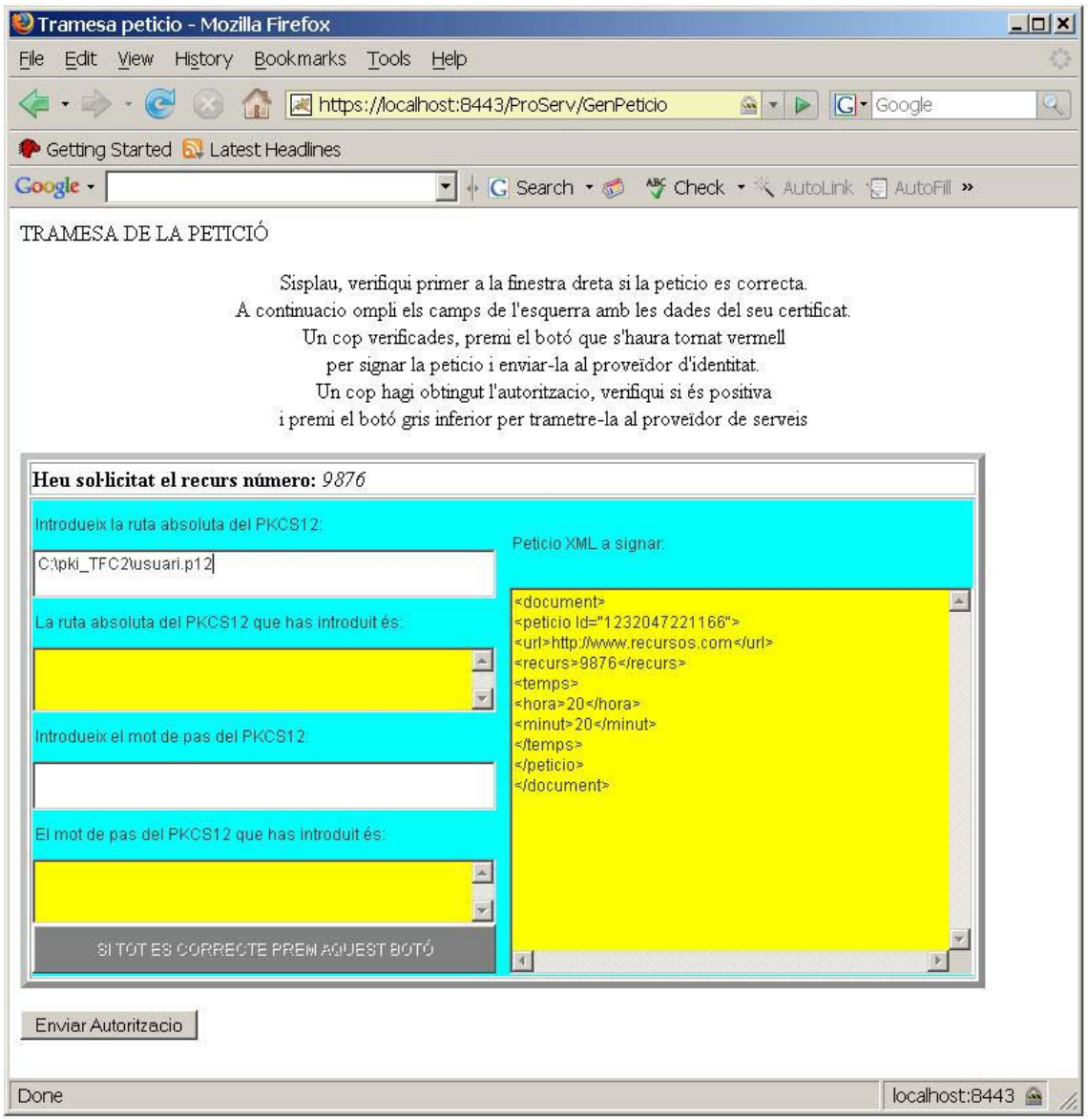
serà redirigit al port segur i la direcció efectiva serà la primera.

En obrir-se la plana, se li oferirà que seleccioni mitjançant botons de ràdio un dels fitxers oferts:

En prémer el botó "Demanar" es carrega l'applet signat, del qual es verifica automàticament la signatura. Així i tot, el navegador demana que se n'autoritzi l'execució:

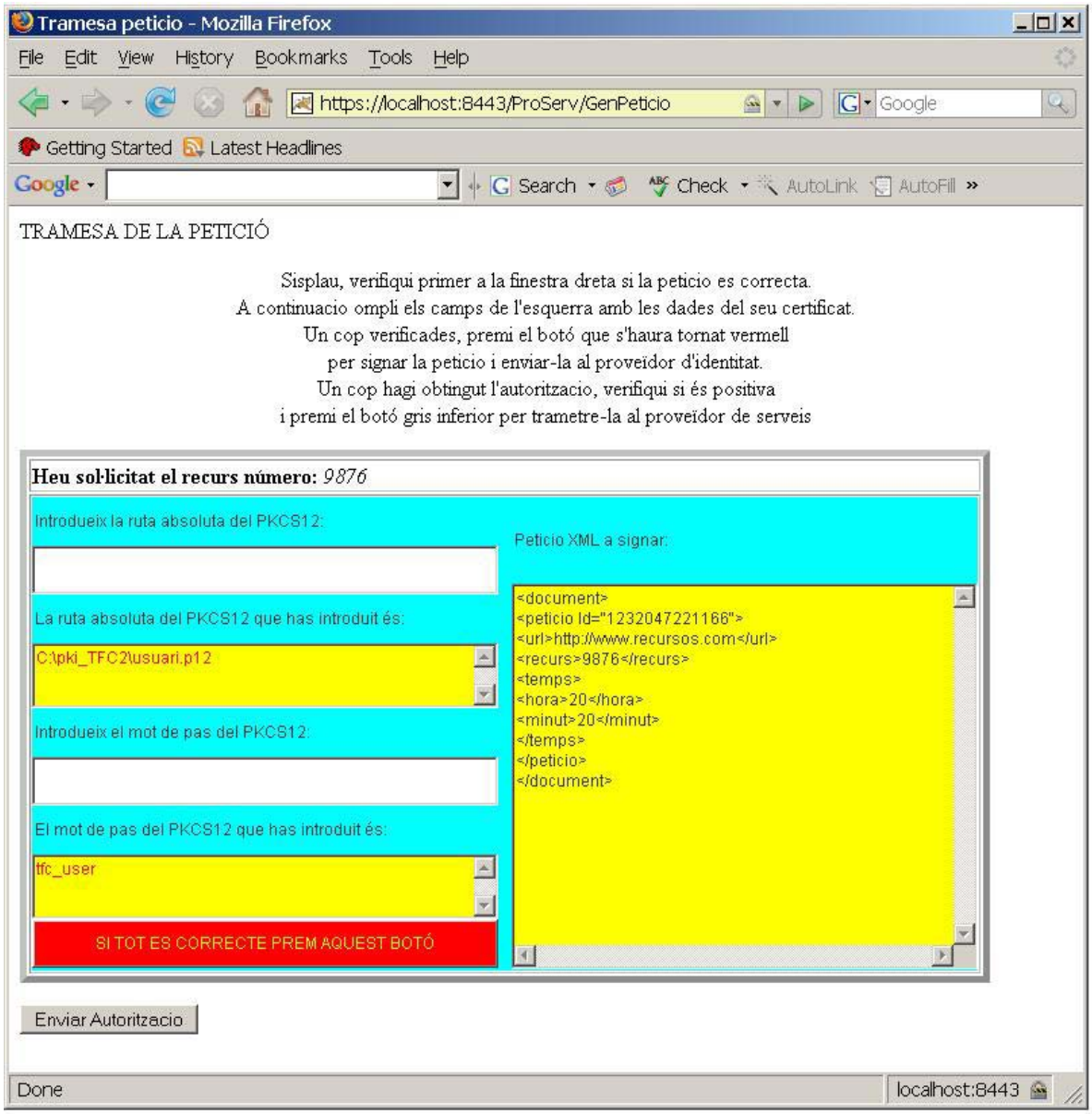


i en fer-ho s'inicia l'applet, que visualitza la petició de la qual ha rebut la signatura detached del proveïdor de serveis.

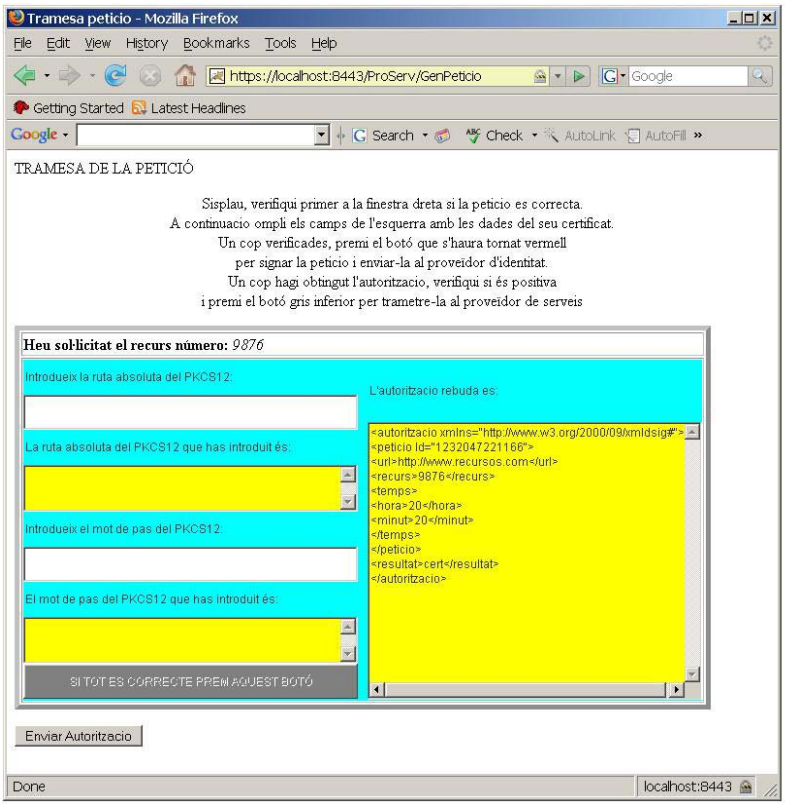


Si el client està d'acord, ha d'omplir els camps amb la ubicació del seu PKCS12 i el mot de pas que hi permet l'accés. Cada cop que omple un camp blanc i prem la tecla *ENTER* la cadena inserida apareix en lletra vermella dins el camp groc inferior.

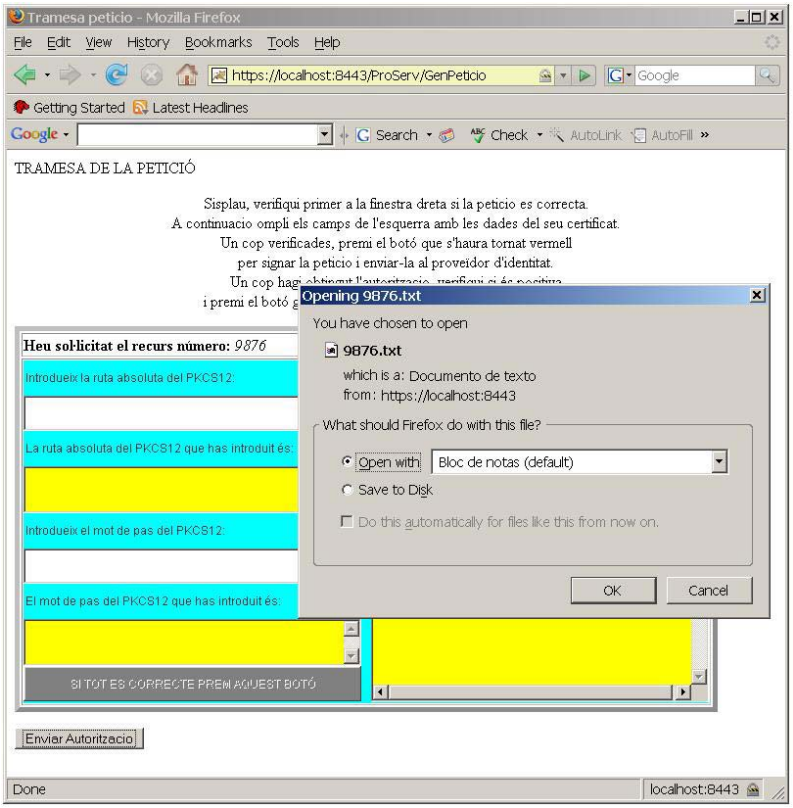
Quan ha omplert els dos camps, s'activa el botó vermell que permet generar la pròpia signatura i enviar-la, juntament amb la del proveïdor de serveis i la petició original, al proveïdor d'identitat:



En prémer el botó vermell es produeix l'intercanvi *SOAP* amb el proveïdor d'identitat, del qual en retorna l'autorització. Aquesta es visualitza a la finestra dreta per a què, veient el resultat, el client decideixi si la vol retornar al proveïdor de serveis per accedir al recurs:



Quan prem el botó “Enviar Autorització” es fa el tràmit i, si és positiu, s’ofereix la descàrrega del recurs:



CONCLUSIONS

Fer aquest treball en un temps tan limitat com són quatre mesos, durant els quals també he hagut de compaginar una feina a temps complet i exigent, i una família amb dos petits però encantadors i ben estimats terratrèmols, ha estat tota una experiència de com arribar a assolir allò que hom creia impossible (de fet, abans de Nadal vaig estar a punt de tirar la tovallola). Per sort en Carlos, el meu consultor, i la Roser, la meva dona, no em van deixar fer, cosa que finalment els he d'agrair sigui quina sigui la qualificació final.

D'una banda, estic descontent de no haver tingut el temps o els coneixements suficients per poder arribar a implementar les millores que m'hagués agradat introduir en el sistema que presento:

- Aconseguir el segellat de temps de les signatures mitjançant una *TSA* que fós la base de captura i processament del temps en què es fa la petició.
- Estructurar millor de classes, segregant els codis comuns de manera que es poguessin reutilitzar més fàcilment i substituir les crides a classes internes de Java per altres d'equivalents.
- Millorar el control i la captura d'excepcions, oferint una millor informació als usuaris.
- Fer una interfície gràfica d'administració més digna i millor estructurada, que mostri en format més agradable i entenedor els resultats de cada transacció.
- Fer que les interfícies gràfiques d'administració i de l'*applet* ofereixin dades prèvies en menús desplegables (els *keystores* presents al sistema del client quan cal que signi la petició, les entitats de la taula que es poden esborrar, les dades actuals d'un permís que es vol modificar, etc)
- Especificar millor les causes d'una denegació d'autorització.
- Bloquejar la possibilitat que el servei *SOAP* sigui accessible externament de forma no segura.
- Consultar una *CRL* amb protocol *OCSP*
- Permetre que el sistema pugui tractar tota una cadena de certificats, en comptes de només un certificat arrel i els seus fills directes.
- Trobar la causa del problema dels *namespaces* i poder treballar amb la petició com a node arrel directament.
- Provar el sistema extensament per polir-ne els defectes.
- Aprendre a treballar amb *Axis2*, *WSDL*, *JSP* i amb la versió *J2EE* d'*Eclipse*.

Però el temps disponible ha estat limitat, i m'hauré de conformar amb:

- Haver aconseguit, amb l'inestimable ajut del meu consultor, en Carlos Ares Angulo, i amb moltes (MOLTES!) hores de cercar a llibres i a Internet, de treball, de dedicació, i també de desesperació, dissenyar, implementar i posar en marxa amb les funcionalitats bàsiques requerides **un sistema segur de descàrrega anònima de fitxers format per aplicacions *web* basades en *XML* i amb autenticació del client per part d'un tercer mitjançant una *PKI*** amb l'ús exclusiu de programari lliure.
- Haver aconseguit, partint de zero, un nivell raonable de coneixements i experiència bàsica en la posta en marxa, configuració i administració d'un servidor *web*, així com els fonaments de com permetre-hi l'accés segur i autenticat.
- Haver aconseguit el nivell de coneixements i l'experiència bàsica necessaris per programar amb les extensions per aplicacions *web* de *J2EE* (*servlets*).
- Haver aconseguit, partint de zero, el nivell de coneixements i l'experiència bàsica necessaris per programar i posar en marxa un servei *SOAP*.
- Haver aconseguit, partint de zero, el nivell de coneixements i l'experiència bàsica necessaris per programar, signar i executar de forma segura un *applet*.
- Haver aconseguit, partint de zero i amb molta ajuda, el nivell de coneixements i l'experiència mínima necessaris per programar punts clau del sistema en *javascript*.
- Haver aconseguit, com era el meu desig quan vaig acabar l'assignatura *Criptografia*, saber treballar amb signatures *XMLDsig detached*.
- Haver aprofundit en el coneixement i les possibilitats d'un *PKI*, i de com tractar-lo programàticament.
- Haver millorat el grau de coneixement i experiència sobre el tractament de documents *XML*.
- Haver aconseguit introdui-me en els fonaments de *awt* i *Swing*.
- Haver comprovat la utilitat d'estudiar les assignatures *Bases de Dades I* i *II* en la posta en marxa i administració d'un *SGBD* de programari lliure.
- Haver comprovat que la formació rebuda durant tots aquests anys d'estudis m'ha capacitat per fer tot l'anterior, buscant autònomament el que no sé i el que he de millorar.
- Corroborar que la majoria de vegades és millor no rendir-se abans que no s'hagi lluitat fins el darrer moment, i que l'ajut i comprensió dels qui t'envolten, així com el suport i els ànims incansables dels bons professors en els moments difícils, treu el millor del què porten dins les persones i són imprescindibles per arribar a bon port.

... que no és poc.

BIBLIOGRAFIA

Llibres

Herrera Joancomartí, J. *Criptografia*, Universitat Oberta de Catalunya, 2006.

Arnold, K.; Gosling, J.; Holmes, D. *El lenguaje de programación JAVA.* Pearson Educación, Madrid, 2001.

Horstmann, C.S.; Cornell, G.; *Core Java 2, Volume I--Fundamentals*

Stallings, W. *Cryptography and Network Security.* Pearson education, USA, 2006.

L'eina OpenSSL. UOC, Febrer 2005.

McLaughlin, B.; Edelson, J.; *Java and XML*. O'Reilly, USA, 2007.

Chopra, V; Li, S.; Genender, J; *Professional Apache Tomcat 6. Wrox Series.* Wiley, USA, 2007.

Perry, B.; *Java Servlet & JSP Cookbook.* O'Reilly, USA, 2004.

Oaks, S.; *Java Security.* O'reilly, USA, 2001.

Enllaços d'Internet

Programació Java

Eckel, B. Thinking in Java, 3rd ed. Revision 4.0. Llibre electrònic: http://www.mindview.net/Books/TIJ/

JDK 6 Documentation.

http://java.sun.com/javase/6/docs/

SDK J2EE 5 Documentation

http://java.sun.com/javaee/5/docs/api/

OpenSSL

http://www.cs.odu.edu/~cs772/fall06/lectures/introduction_openssl.html

http://resin.csoft.net/cgi-bin/man.cgi?section=1&topic=openssl

http://www.scribd.com/doc/915895/OpenSSL-User-Manual-and-Data-Format

Criptografia Java

JavaTM Cryptography Architecture. API Specification & Reference

http://java.sun.com/j2se/1.5.0/docs/guide/security/CryptoSpec.html

Java XML Digital Signatures.

http://www.w3.org/TR/2002/RECxmldsig-core-20020212/

http://java.sun.com/developer/technicalArticles/xml/dig_signatures/

http://java.sun.com/javase/6/docs/technotes/guides/security/xmldsig/XMLDigitalSignature.html#wp511406

http://java.sun.com/developer/technicalArticles/xml/dig_signature_api/

Signatura d'applets

http://java.sun.com/javase/6/docs/technotes/guides/plugin/developer_guide/security.html

http://java.sun.com/j2se/1.5.0/docs/guide/plugin/developer_guide/rsa_deploying.html

http://www.pawlan.com/Monica/signedapps/

JAXML i SOAP

http://java.sun.com/j2ee/1.3/docs/tutorial/doc/IntroWS6.html

http://java.sun.com/j2ee/1.4/docs/tutorial/doc/SAAJ2.html

http://www.javaworld.com/javaworld/jw-09-2003/jw-0912-webservices.html?page=1

Derby

http://db.apache.org/derby/

http://db.apache.org/derby/papers/DerbyTut/index.html

http://www.herongyang.com/jdbc/Java-DB-Installation.html

Axis

http://ws.apache.org/axis2/1_4_1/userguide.html

Tomcat

http://tomcat.apache.org/tomcat-6.0-doc/index.html

http://tomcat.apache.org/tomcat-4.0-doc/ssl-howto.html

http://forum.springframework.org/archive/index.php/t-10181.html

Programació d'applets

http://csis.pace.edu/~bergin/sol/java/gui/

Javascript

http://www.w3schools.com/JS/default.asp

http://www.webestilo.com/javascript/

http://www.wikilearning.com/tutorial/completo_tutorial_de_javascript/4466

http://www.javascript-coder.com/javascript-form/javascript-form-submit.phtml

Timestamping

http://www.java2s.com/Open-Source/Java-Document/Security/Bouncy-Castle/org.bouncycastle.tsp.htm

http://www.java2s.com/Open-Source/Java-Document/Security/Bouncy-Castle/org/bouncycastle/tsp/TimeStampRequestGenerator.java.htm

...i moltes més per a consultes puntuals

A Mataró, el 16 de gener de 2009

Ramon Costa i Sala