Localitzador Gràfic d’Adreces IP

Aleix Cerecedo Escriu
ETIS

Consultor: David Carrera Pérez

18 / 06 / 2002

En ple creixement d’internet, l’enmagatzematge d’informació pública requereix programari que faciliti la seva gestió i comprensió. Aquesta és la finalitat del Localitzador Geogràfic d’Adreces IP, facilitar la extracció de tota la informació relacionada amb una adreça ip i les seves adreces intermitjes.

Evidentment, el nom del software indica quina és la peça clau d’aquesta informació a extreure: la seva localització geogràfica. Aquest atribut de cada ordenador, però, no queda reflectit en la informació pública disponible, de moment significativament insuficient. Així doncs, aquest projecte dedica a abordar el problema de la extracció i visualització de forma entenedora de la informació pública més rellevant d’una adreça ip, però no aborda els problemes relacionats amb la informació insuficient, suposant que pertanyen a un àmbit de recerca diferent del pròpi projecte.

En quant a la localització geogràfica d’una adreça, aquesta està codificada a internet a partir d’uns paràmetres tals com la longitud i la latitud. El programa facilitarà la seva interpretació a partir de la seva ubicació en un mapa del globus terraqui però també els seus valors tal com queden codificats a internet.

Tota la informació relacionada amb una adreça ip s’extreu de les bases de dades DNS (Sistema de Noms de Domini), on s’hi troba enmagatzemada. Així doncs el programa té una part que implementa un resolvedor DNS i una part gràfica que visualitza la informació. A més, el programa implementa un tracejador de ruta per a saber quines són les màquines intermitjes. És a causa del tracejador que el programa no és multiplataforma, és a dir, que s’ha hagut de pensar en un sistema operatiu concret, que en aquest cas ha sigut Linux.

Índex

1. Introducció.....4
1.1. Punt de partida i aportació del TFC.....4
1.2. Objectius del TFC.....4
1.3. Enfocament i mètode seguit.....5
1.4. Planificació del projecte.....5
1.5. Productes obtinguts.....6
1.6. Breu descripció dels altres capítols de la memòria.....6

2. Construcció projecte.....7
2.1. Iteració 1: Nucli, funcionalitat bàsica.....7
2.1.1. Recollida de requisits.....7
2.1.2. Anàlisi i Disseny.....8
2.1.3. Implementació.....9
2.1.4. Test.....10
2.2. Iteració 2: Nucli, millora de rendiment i funcionalitat....11
2.2.1. Recollida de requisits.....11
2.2.2. Anàlisi i Disseny.....11
2.2.3. Implementació.....13
2.2.4. Test.....14
2.3. Iteració 3: Interficie.....15
2.3.1. Recollida de requisits.....15
2.3.2. Anàlisi i Disseny.....15
2.3.3. Implementació.....17
2.3.4. Test.....22
2.4. Iteració 4: Compatibilitat amb IPv6.....24
2.4.1. Recollida de requisits.....24

3. Valoració econòmica.....25

4. Conclusions.....26

5. Glossari.....27

6. Bibliografia.....28

7. Annexos.....29
7.1 Annexe 1: Script bin/main.....29

1. Introducció

1.1. Punt de partida i aportació del TFC.

En el moment en que es desenvolupa aquesta aplicació, internet està arribant als limits de la seva capacitat d'ordenadors conectats. Aquesta massificació d'ordenadors, en cap cas sorprenent, ha comportat la revisió d'aspectes que han passat a emergir a primera linia en importància per a la comunitat, tals com la seguretat i la privacitat dels usuaris.

A mida que han sorgit nous problemes o reptes, entitats, agrupacions d'usuaris o inclús individus han aportat idees que molts cops s'han plasmat en noves eines per a la xarxa. Però en la majoria de casos, aquestes idees triguen en arribar en forma de eines útils o conegudes per el usuari convencional. Això és degut, entre d'altres qüestions, a la seva complexitat, que impossibilita que un usuari convencional pugui fer-hi ús de forma instantània.

Així doncs, a mida que internet creix, i amb ella la informació que facilita per als seus usuaris, aquests depenen més d'usuaris o entitats, que per motius diversos, saben i poden construir programes que permetin accedir i entendre aquesta informació abans invisible als usuaris convencionals.

Tenint en compte que justament són els usuaris convencionals qui més insegura consideren internet, a causa, normalment, del seu desconeixement del seu funcionament, es fan indispensables totes les eines que els hi facilitin aquesta informació de forma clara.

Centrant-nos en els objectius d'aquest projecte, el programa està pensat per a poder localitzar els ordenadors en el globus terraqui. Aquesta informació pot ser d'ajuda quan, amb la ajuda d'un firewall, un usuari detecta un intent d'intrussió a la seva màquina i decideix averiguar d'on prové. Les dades recopilades per el localitzador podríen ser útils a la policia en cas de denúncia.

Un altre cas, però, és el de la autenticitat. Moltes pàgines web visitades no ofereixen cap mena de garantia (certificats) respecte l'autenticitat de qui hi ha al darrera. La localització geogràfica de la màquina que conté la pàgina web no té perqué significar alguna cosa, però pot ajudar a decidir en cofiar en un espai web o no.

Finalment, en el àmbit acadèmic, el programa podria ajudar a el.laborar un mapa de les linies de comunicació que constitueixen internet.

1.2. Objectius del TFC.

Així doncs, ordenats per preferència, els objectius bàsics del projecte són:

- Contruir el programari que permeti visualitzar el punt en el mapa de la ubicació de tots els nodes en la ruta amb la IP destí.
- Recollir el màxim d'informació útil sobre cada IP tractada.
- El rendiment sigui acceptable.
- Que sigui compatible amb IPv6.

1.3. Enfocament i mètode seguit.

Tenint en compte que el projecte consisteix a extreure informació d'internet, tractar-la (calcular la ubicació de una IP en un mapa) i mostrar-la, el projecte s'ha dividit en dues parts: el nucli i la interfície.

El nucli s'encarregarà d'extreure la informació necessaria d'internet. Aquesta part inclou el motor de cerca de la aplicació.

La interfície tractarà les dades i la informació necessaria per a localitzar un punt en el mapa del globus, a part d'implementar la funcionalitat del aplicatiu.

El procediment s'ha basat en la recerca d'informació necessaria per dur-lo a terme. Això implica buscar documentació i eines conjuntament amb la realització del disseny.

Així doncs, s'ha optat per la construcció del programari amb un cicle de vida iteratiu i incremental, concretament amb el cicle de vida del Rational Unified Process. Aquesta tècnica s'adapta molt bé als casos en que es desconeixen requisits, i en aquest cas concret ens evitarà la dependència que té la part visual de totes les dades que es poden arribar a mostrar. De fet, s'espera acabar la primera fase amb un motor de cerca senzill que permeti trobar alguna informació bàsica (juntament amb la localització geogràfica) i mostrar-la per pantalla. Així, quan es comenci a implementar la part visual, anirem ampliant el contenedor d'informació a mida que es vagin assolint els objectius més bàsics (funcionalitat, rendiment de la aplicació, etc).

Com a part fonamental de l'enfocament inicial, s'ha intentat desde el primer moment realitzar el projecte amb la tecnologia Java, bàsicament per qüestions de familiaritat i facilitat d'ús per part del desenvolupador.

1.4. Planificació del projecte.

Com s'acaba de comentar, tant per al projecte global com per al desenvolupament de cada fase utilitzarem el cicle de vida del Rational Unified Process. Aquest cicle de vida consta de iteracions independents que a la vegada consten de quatre fases diferenciades: recollida de requisits, anàlisi i disseny, realització, i prova.

Aquestes subfases ens facilitaran la el.laboració de cada objectiu (fases nucli i interficie), però a més ens permetran aplicar noves iteracions a cada objectiu fins a arribar als objectius finals. Així doncs ens donen marge a trencar amb la planificació inicial de dues fases (nucli i interfície) podent incorporar noves iteracions complementaries.

En el cas de trobar eines o llibreries que ens facilitin la feina aquests es determinaran en la fase d'anàlisi de requisits. Així, tot i que la tasca de recerca d'informació serà sempre present en qualsevol subfase de tot el procés de construcció del programari, serà especialment important en aquesta subfase. L'objectiu bàsic és delimitar el domini de la tasca, així que si s'escau realitzarem els diagrames necessaris que ens permetin definir millor el problema.

En la fase d'anàlisi i disseny es tindrà en compte el domini de la tasca i les eines escollides de suport a la seva construcció. A partir de l'anàlisi d'aquesta informació s'el.laborarà el disseny que defineix clarament com realitzar la següent tasca: la implementació. El disseny es realitzarà amb el/s diagrama/es UML que millor s'escaigui/n al problema.

La fase d'implementació es realitzarà i s'anotaran els comentaris escaients per a la seva fàcil comprensió. En aquesta documentació es referenciaran els fitxers que formin part d'aquesta subfase.

Finalment es realitzarà la subfase de test, que ens permetrà evaluar el rendiment i la correcta funcionalitat de la tasca realitzada en la iteració. Aquesta fase aportarà els jocs de proves necessaris.

1.5. Productes obtinguts.

Per a la implementació del resolvedor DNS s'ha usat la llibreria lliure dnsjava (www.xbill.org/dnsjava). Aquesta llibreria implementa exhaustivament totes les funcionalitats del protocol.

1.6. Breu descripció dels altres capítols de la memòria.

Tenint en compte la metodologia a seguir (cicle de vida del Rational Unified Process), s'ha pensat que el millor és que cada punt del capítol *Construcció projecte* correspondrà a una iteració, i aquesta s'estructurarà en les subfases del model de construcció.

Així aconseguirem que es pugui visualitzar clarament tot el procés de construcció del localitzador, i alhora aconseguirem estructurar la informació de forma clara i senzilla.

2. Construcció projecte

2.1. Iteració 1: Nucli, funcionalitat bàsica.

En aquesta iteració volem construir el motor de cerca d'informació. Aquesta s'haurà d'encarregar de buscar la localització geogràfica de totes les IP's de la ruta amb la IP destí introduida per l'usuari.

De moment no tindrem en compte la comptabilitat amb IPv6.

2.1.1. Recollida de requisits.

En aquesta fase s'ha trobat el RFC 1876 (A Means for Expressing Location Information in the Domain Name System), on es parla de com s'enmagatzema i transmet la informació relativa a la localització geogràfica d'una IP o domini.

Això ha demostrat que almenys, la informació a extreure d'una IP per part del programa estarà als servidors DNS. Aquest fet ha dut a trobar la llibreria lliure dnsjava (www.xbill.org/dnsjava), que implementarà un resolvedor DNS java i totes les estructures de dades que corresponen a aquest protocol.

Però per conèixer aquest protocol s'ha hagut de buscar els documents oficials que el descriuen: RFC 1034 (Domain names - concepts and facilities), i RFC 1035 (Domain names - implementation and specification).

Per altra banda, com que sabem que aquest motor de cerca s'ha d'encarregar de extreure la localització geogràfica de totes les IP's de la ruta amb la IP destí introduida per l'usuari, sabem que necessitarem una especie de tracejador de ruta que ens permeti saber quines són aquestes màquines.

Aquest nou aspecte ens ha portat a buscar el funcionament dels tracejadors de ruta coneguts (traceroute, per exemple), i ens ha fet veure que necessitem accedir als paquets d'error ICMP que utilitza internet. Això és degut a que les tècniques per esbrinar quines màquines hi ha en una ruta parteixen de provocar aquest tipus d'errors al mig de la comunicació amb la IP destí, podent esbrinar així quines són aquestes màquines intermitjes.

El funcionament és com segueix:

- S'envia un paquet IP qualsevol a la IP destí però amb un màxim de salts més petit del suficient per que arribi a destí.
- La màquina enrutadora del mig de la ruta que reb el paquet i veu que el màxim nombre de salts no li permet reenviar-lo endavant, retorna un paquet ICMP a la IP orígen informant del error TIME_EXCEEDED.
- Al recuperar aquest paquet d'error podem saber quina IP té aquesta màquina intermitja.

Aquest procés doncs, es basa en anar incrementant el nombre de salts d'un paquet qualsevol, de forma que cada cop coneixem una màquina més propera al destí. Finalment, com que aquest paquet qualsevol va dirigit al port d'*echo*, la màquina destí ens retorna una resposta d'*echo*, que ens indica el final de la nostra cerca.

Com que Java no permet l'accés a paquets ICMP de la xarxa, la implementació d'aquesta part s'ha de fer amb una altra tecnologia. Però com que no volem refer tot el projecte en aquest nou llenguatge, usarem JNI (Java Native

Interface) per a comunicar Java amb la funció que tractarà els paquets ICMP. Així el tractament dels paquets ICMP el farem amb C, i les especificacions del protocol ICMP les traurem del estàndard RFC 792 (Internet Control Message Protocol).

Finalment, sabent que en les qüestions de rendiment el coll d'ampolla serà internet, i sabent que haurem d'esperar moltes respostes, el disseny haurà de tenir en compte la execució concurrent de les consultes.

Així doncs, en aquesta fase haurem d'implementar una petita aplicació Java que serà en motor de cerca. Aquesta, a través de la execució concurrent, de la llibreria lliure dnsjava i la llibreria d'accés al protocol ICMP, cercarà la localització geogràfica de totes les IP's de la ruta amb la IP destí que la tinguin informada en les bases de dades DNS. També haurem de tenir en compte la possibilitat de que la ruta no ens permeti conèixer màquines intermitjes, cas en que no mostrarem informació.

2.1.2. Anàlisi i Disseny.

Primer de tot em de tenir en compte que utilitzarem JNI per a cridar a la llibreria feta amb C (que preguntarà per les màquines intermitjes en la ruta). JNI ens permet definir la signatura de la funció nativa (la funció C) en Java. Després, amb la execució de la comanda javah, java ens crearà un fitxer header de C, el qual implementarà la nostra llibreria C.

Així doncs, la execució concurrent la podrem fer amb Java a través de la classe Thread si definim bé aquesta funció. Per a fer-ho la funció de la llibreria nativa només podrà preguntar per una màquina, de forma que podrem llençar desde Java cada pregunta com fil d'execució independent o no (al definir-ho així donarem a Java la llibertat de definir els fils d'execució i podrem analitzar en la fase test el rendiment i veure quina és la millor opció).

Així doncs la signatura en Java de la funció nativa de C serà:

```
getHostOfRoute(String ipDesti, int nombreSaltsMax): String
```

Aquesta funció ens retornarà la IP de la màquina situada en el salt=nombreSaltsMax en forma de cadena de text. En cas de que nombreSaltsMax sigui major que el nombre de salts necessari per arribar a la IP destí ens retornarà la IP de la màquina destí.

Per altra banda tindrem el motor de cerca, que s'encarregarà de preguntar la localització geogràfica per cada IP intermitja que hi hagi en la ruta. Aquest motor haurà de ser capaç de definir i controlar la execució concurrent de les preguntes. Començarem amb una implementació fàcil, és a dir, un sól fil d'execució per a totes les preguntes juntes, de forma que la concurrència no sigui important. El que si és important és que les preguntes en sí s'executin independentment del programa principal, ja que l'objectiu és que a mida que es sàpiguen els resultats de la cerca es vagin mostrant.

Cada pregunta haurà de averiguar quina és la seva IP a preguntar (ho farà a través de la llibreria nativa), i un cop la conegui preguntar a través de la llibreria lliure dnsjava quina és la seva localització geogràfica.

Així doncs el disseny UML del motor Java queda així:

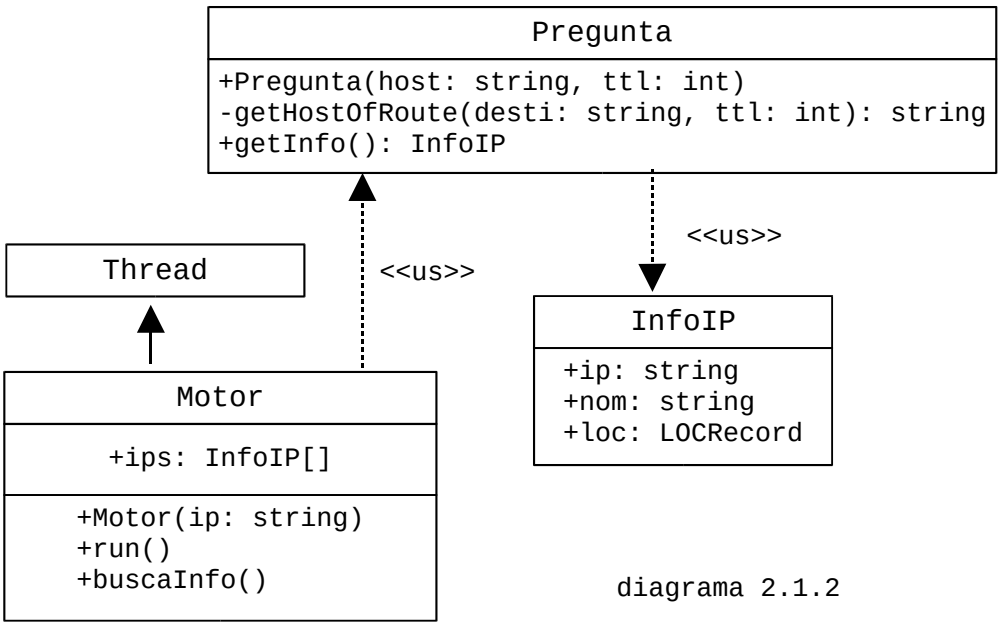


diagrama 2.1.2

2.1.3. Implementació.

La implementació forma part del paquet anomenat *iteracio1* i s'ubica dins la carpeta *src*. El resultat són els fitxers de la carpeta *src/iteracio1*: *InfoIP.java, Pregunta.java, i Motor.java*.

D'aquests fitxers és *Pregunta.java* qui accedeix a la llibreria dnsjava. De totes formes la llibreria implementa la estructura de dades *LOCRecord*, que enmagatzema la informació de la localització geogràfica. Així doncs, la classe *Motor.java* també accedirà a aquesta estructura. Un exemple de com s'utilitza la llibreria per a fer una consulta és el següent codi:

```
try {
        String hostname = “yahoo.com”;

        // Busquem localització geogràfica
        Lookup lookup = new Lookup(hostname, Type.LOC);
        Record[] records = lookup.run();
        if (records.length>0)
        {
                LOCRecord loc = (LOCRecord)records[0];
                System.out.println("Alt.: "+loc.getAltitude());
        }
}
catch (TextParseException ex) {}
```

Abans de compilar s'haurà d'instalar la llibreria lliure dnsjava. Com que estem desenvolupant això ho farem copiant la estructura de fonts de la llibreria a a la carpeta *src*. Aquests pengen de la carpeta *org*. Un cop tenim els fonts de la llibreria ben ubicats compilem els nous a la carpeta *class* amb la comanda següent:

```
javac -d bin src/iteracio1/*.java
```

Aquesta comanda ens genera tots els fitxers class necessaris en la carpeta *bin*.

Després generem la capçalera JNI de la classe *bin/iteracio1/Pregunta.class* executant la comanda:

```
javah -classpath bin -o src/iteracio1/Motor.h iteracio1.Motor
```

Com es pot comprobar s'ha generat el fitxer capçalera (*Motor.h*) en la carpeta *src/iteracio1*. Així doncs, un cop s'ha generat la capçalera s'implementa la llibreria C amb el fitxer *src/iteracio1/trace.c*.

El següent pas és generar la llibreria dinàmica *libtrace.os* perque es pugui executar el motor de cerca. Aquesta llibreria és cridada desde Java a través de JNI. Per a fer-ho em d'executar les comandes:

```
export JAVA_PATH=/usr/java/j2sdk1.4.2_03
gcc -w -c -I$JAVA_PATH/include \
-I$JAVA_PATH/include/linux \
src/iteracio1/trace.c
ld -G trace.o -lm -lc -o bin/iteracio1/libtrace.so
rm trace.o
```

On *$JAVA_PATH* és una variable que indica el directori on tenim instal.lat Java. En el meu cas és la carpeta */usr/java/j2sdk1.4.2_03*. La comanda ens genera el fitxer *bin/iteracio1/libtrace.so*.

Un cop tenim el motor de cerca, la llibreria lliure dnsjava, i la llibreria dinàmica libtrace.os ja podem executar el motor de cerca.

2.1.4. Test.

Per executar la aplicació em implementat una petita classe anomenada *Test.java* (ubicada com tots els fonts a la carpeta *src/iteracio1*) que s'encarrega de cridar a la classe *Motor*.

Abans de executar-la, però, caldrà entrar en mode superusuari degut als permisos que es requereixen per a executar la llibreria *libtrace.os*.

Seguidament caldrà indicar a Java quina és el path de llibreries. Això ho resol la comanda següent:

```
export LD_LIBRARY_PATH=bin/iteracio1
```

Ara la execució es farà amb la següent comanda:

```
$JAVA_PATH/bin/java -cp bin iteracio1.Test1 yahoo.com
```

(Nota 1: cal notar que en tot el procés s'han declarat les rutes completes, però es podíen haver posat aquestes carpetes al path del sistema.)

(Nota 2: Per a facilitar el desenvolupament del tot el projecte s'ha creat un script anomenat *main* i ubicat a la carpeta *bin* que s'encarrega de compilar i executar qualsevol de les iteracions implementades. Aquest s'ha d'executar desde la carpeta pare, és a dir, cridant-lo així: *bin/main ...* El seu funcionament queda detallat al punt *7.1 Annexe 1: Script bin/main*)

Com a resultat de les primeres execucions s'ha detectat un relatiu baix rendiment a causa de les crides a internet (per cada host intermig em d'esperar a una resposta d'error, i després preguntar amb el resolvedor DNS per el seu registre LOC). Això ens conduirà, doncs, a una iteració en el desenvolupament del projecte que s'encarregarà d'evaluar si es pot reduir l'efecte d'espera definint un fil d'execució per cada pregunta.

També s'ha detectat que el motor no funciona pels noms de domini que no tenen una ip relacionada, fet que s'espera que la segona iteració resolgui.

Un altre fet és que en un 0% dels casos probats no troba el registre LOC que conté la ubicació geogràfica de la IP, per tant, una altra tasca a realitzar en la següent iteració serà incorporar alguna tècnica de rastreig per trobar la ubicació.

2.2. Iteració 2: Nucli, millora de rendiment i funcionalitat.

En aquesta iteració partim del nucli anterior. Volem millorar el tractament de la concurrència i millorar l'algoritme de cerca del registre LOC relacionat amb una IP. També volem recuperar tota la informació interessant dels servidors DNS.

De moment no tindrem en compte la comptabilitat amb IPv6.

2.2.1. Recollida de requisits.

Com que en aquesta fase partim del codi font del nucli anterior per a millorar-lo, els únics requisits que s'ha hagut de recollir són els referents a la informació a extreure dels servidors DNS que ens pugui ser útil.

Després de consultar el RFC 1035 (Domain names - implementation and specification) s'han pogut observar diferents camps que poden resultar ser d'interés (registres SOA, TXT, SRV, NS).

Amb l'ajuda de la aplicació *host* de linux s'han pogut fer proves de cerca d'aquests registres. El resultat ha sigut que molts servidors de noms no informen alguns d'aquests registres (inclós el registre LOC). Per tant, només recuperarem la adreça de correu electrònic de l'administrador de la màquina (ubicat dins el registre SOA) a part de la localització geogràfica.

En qualsevol cas s'ha d'utilitzar alguna tècnica de cerca avançada per a vincular la poca informació pública que hi ha a la IP o domini sol.licitat. Una d'aquestes tècniques serà la cerca per subdomini. La idea parteix que molts servidors DNS informen el SOA o el LOC només per el nom de domini privat superior, estalviant-se així, replicar la informació per a cada màquina que hi tinguin donada de alta.

La segona tènica de cerca és la cerca per subxarxa definida en el RFC 1876 (A Means for Expressing Location Information in the Domain Name System). Aquest sistema parteix de la premisa que si una màquina pertany a una xarxa i preguntant per la màquina no trobem la informació que volem, la informació vinculada a la adreça de xarxa ens servirà per igual. Un exemple seria una xarxa que físicament està situada a les mateixes coordenades (la mateixa habitació, per exemple) i l'administrador decideix informar el registre LOC només un cop, és a dir, informar el LOC només per la adreça de xarxa i així no informar-lo per cada màquina que estigui donada d'alta al DNS.

Com es pot comprobar, amb les dues tècniques es fa indispensable informar d'on s'ha extret finalment la informació a mostrar.

Finalment definim els objectius de la iteració:

1. Reduir el temps de resposta del motor de cerca. Aquest temps de resposta és molt elevat a causa de que els retards causats per una petició a internet repercuteixen a totes les peticions posteriors.
2. Permetre que es pugui buscar la localització geogràfica d'un domini tot i que no tingui una IP relacionada.
3. Millorar els algoritmes de cerca.

2.2.2. Anàlisi i Disseny.

El primer que s'observa és que els objectius 2 i 3 no afecten el disseny definit en la iteració anterior. Els canvis que provocaran només afectaran al codi de les classes ja implementades.

En canvi, l'objectiu 1 implicarà bastants canvis en el disseny inicial. El problema que volem sol.lucionar és el del temps de resposta, i sabem que aquest es podria millorar si les peticions a internet no esperèssin a que les anteriors estiguèssin satisfetes.

Això es pot aconseguir fent que cada pregunta sigui un fil d'execució independent de les altres preguntes, no com està que cada pregunta forma part d'un sól fil d'execució que defineix la classe Motor (diagrama 2.1.2). Així tindrem que si alguna pregunta concreta rep una resposta amb molt de retard no afecti a les altres, que també estaran esperant resposta.

Cal tenir en compte també que al ser fils d'execució independents, haurà d'haver algun sistema de senyalització de que la cerca ha finalitzat. Per tant hi haurà una relació directa entre la classe Pregunta i la classe Motor.

També redefinirem la finalitat de cada classe:

- Motor: s'encarregarà d'esbrinar quines IP's hi ha a la ruta i per cadascuna d'elles llençarà una instància de Pregunta.
- Pregunta: serà un fil d'execució independent i s'encarregarà de cercar tota la informació relacionada amb una IP de les bases de dades DNS.

La classe contenedora InfoIP quedarà modificada: s'afegirà el camp mail, per guardar la direcció de correu del administrador; i s'afegiran els camps loc_src i mail_src, que enmagatzemaran el domini d'on s'ha extret la localització geogràfica i el domini d'on s'ha extret la direcció de correu de l'administrador respectivament.

Finalment, i amb vistes a la creació de la interficie de la aplicació, crearem una *interface* Java que definirà una *Interficie*. Això ho farem degut a que un cop una pregunta ha acabat una cerca necessitarà informar a la interficie. La interface Interficie definirà un mètode que permetrà afegir informació relacionada amb una IP en qualsevol moment.

Així doncs, el nou disseny UML del motor de cerca queda així:

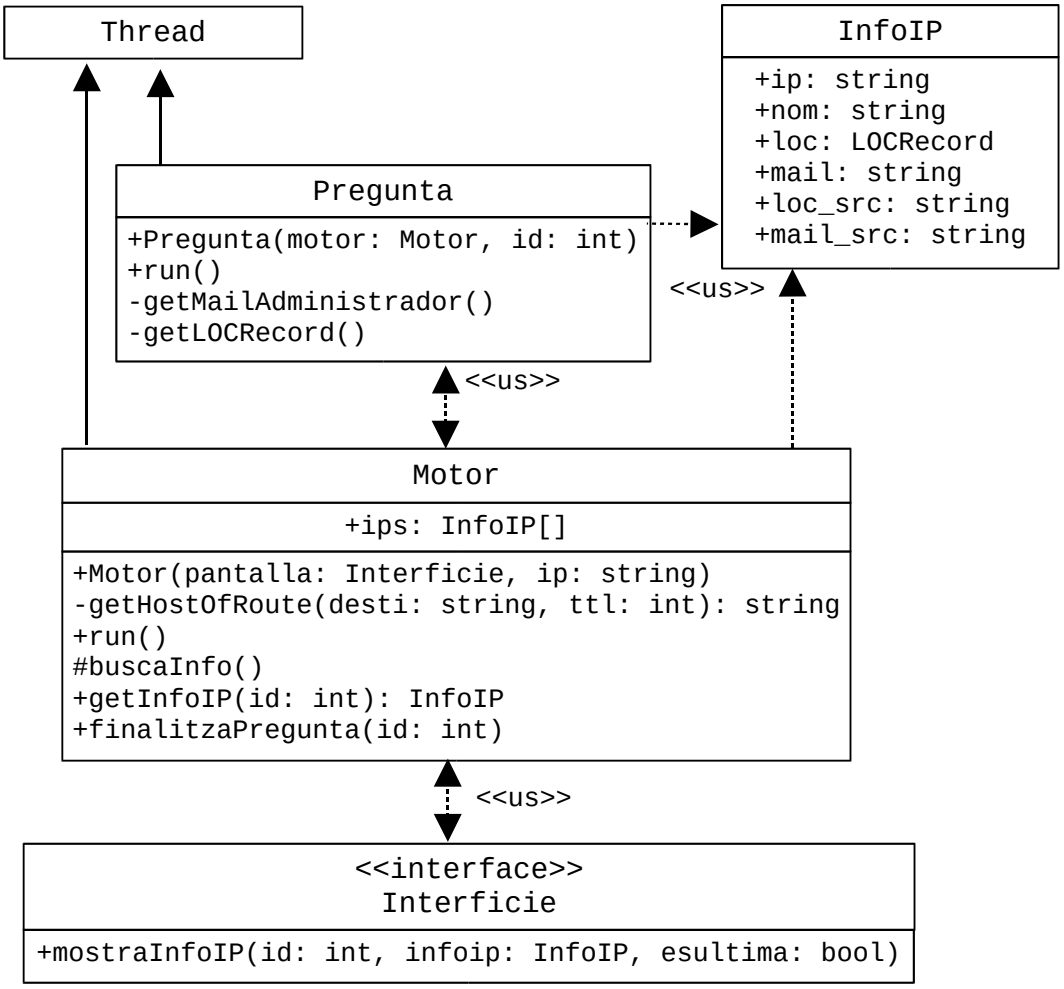


diagrama 2.2.2

On podem veure que la classe Motor gestiona les IP's a partir d'un id (que serà la posició en la llista *ips*) que és el que utilitzarà la classe *Pregunta* per poder accedir al contenedor associat.

També cal remarcar que en el tractament de la concurrència es manté una dependència entre peticions a internet. Aquesta situació apareix en el moment que volem saber quina ruta hi ha amb la IP destí: fins que no rebem resposta sobre quina IP hi ha a tanta distància no podem preguntar per la següent màquina a una unitat més de distància. Això és degut a que no sabem quantes màquines hi ha en una ruta. Per tant, no evitem els retrassos provocats per les peticions d'echo, però no serà gaire significatiu comparat amb els retrassos que provoquen les peticions DNS, que al ser un protocol de capa superior és més lent,

2.2.3. Implementació.

Al aplicar els canvis definits pel nou disseny ens trobem que la classe que utilitzarà JNI per a comunicar-se amb la llibreria nativa *libtrace.so* serà *Motor*. Això fa que només haurem de canviar la signatura JNI de lloc: de la classe *Pregunta* passarà a la classe *Motor*. Aquest fet tindrà alguna repercussió: el fitxer capçalera generat no s'anomenarà *Pregunta.h*, sinó que es dirà *Motor.h*.

També caldrà tenir en compte que al ser la segona iteració les classes formaran part d'un paquet anomenat *iteracio2* i s'ubicaran a una carpeta amb el mateix nom dins la carpeta *src*. Per tant, la signatura de la funció C generada dins la capçalera també canviarà de nom ja que aquest depen del paquet al qual pertany (veure el fitxer generat per javah anomenat *src/iteracio2/Motor.h*). Així doncs, el fitxer *src/iteracio2/trace.c* haurà d'implementar una funció amb nom diferent i enllaçar amb una capçalera diferent que a la primera iteració, tot i que la resta de codi seguirà sent el mateix.

Respecte la classe *src/iteracio2/Test.java* que ens servirà per arrancar el motor implementarà la interface *src/iteracio2/Interficie.java*. Així contindrà el codi necessari per a mostrar la informació rebuda en qualsevol moment per la classe Motor.

La classe *src/iteracio2/Motor.java* quedarà bastant diferent, ja que em canviat la seva finalitat. Ara haurà de gestionar la llista de InfoIP de forma que per cada IP de la ruta que vagi rebent vagi llençant instàncies de la classe Pregunta. També estarà preparada per si la IP destí demanada no és una IP, situació en la que només llençarà una pregunta per al domini sol.licitat.

La altra classe que s'ha de modificar és *src/iteracio2/Pregunta.java* degut a la seva redefinició. Ara partirà d'una InfoIP que contindrà la IP i el hostname, i a partir d'aquests farà les peticions necessaries per recuperar la informació especificada en els objectius de la iteració (veure punt 2.2.1) i omplir així el contenedor. Aquest el recuperarà a partir del id que li passarà la classe Motor al crear la instància de Pregunta i a través de la referència a Motor (que també és un paràmetre). La referència a Motor és necessaria per la classe Pregunta a causa de que ha de poder comunicar-li que ja ha finalitzat la cerca, moment en que aquest avisarà a la Interficie.

Finalment cal parlar dels algoritmes de cerca de la localització geogràfica i de la direcció de correu del administrador que implementa la classe Pregunta.

La primera tènica de cerca que em comentat és la <u>cerca per subxarxa</u> definida en el RFC 1876 (A Means for Expressing Location Information in the Domain Name System). La idea és aconseguir totes les adreces de xarxa a les quals pertany la IP objectiu ja que aquestes poden tenir informat el registre de la IP. El procediment que segueix és el següent:

1. A partir d'una IP es busca la xarxa a la qual pertany en funció de la classe de xarxa que sigui (A, B o C).

2. Es prepara el nom de domini per a una IP, és a dir, s'enganxa la IP (invertint l'ordre dels nombres de la direcció) amb el sufixe IN-ADDR.ARPA.
3. Es busca el registre PTR (registre per a cercar un domini a partir de la seva IP) del domini que tenim. Si el trobem s'afegeix el domini informat al registre PTR a la pila de dominis candidats a tenir informat el registre que cerquem.
4. Es continua amb el domini del punt 2 i es pregunta pel registre A (adreça IP) d'aquest domini. Com que el domini que busquem és una adreça de xarxa, si té aquest registre informat és que correspon a la màscara de la subxarxa. Si trobem aquesta màscara es calcula la nova adreça de xarxa i es passa al punt 2, sinó es deixa de buscar dominis candidats.
5. Finalment, anem preguntant pels dominis candidats fins que trobem el registre que cerquem o s'acaben els dominis candidats.

La segona tècnica és la cerca per subdomini. L'objectiu és buscar el subdomini dins el domini a analitzar que tingui informat el registre que busquem. El procediment que segueix és el següent:

1. Busquem el registre a buscar de la màquina 147.Red-80-58-33.pooles.rima-tde.net i no el trobem.
2. Retallem la primera etiqueta del nom de domini i busquem el registre de la màquina resultant (Red-80-58-33.pooles.rima-tde.net).
3. Si no el trobem tornem al pas 2. Si el trobem deixem de buscar.

2.2.4. Test.

Tant per a la compilació com per a la execució s'ha utilitzat el script *bin/main* comentat en el punt 2.1.4, Nota 2; i detallat al punt *7.1 Annexe 1: Script bin/main*. En aquesta iteració em fet servir la opció -t del script, és a dir, em llençat el joc de proves. S'ha fet amb la següent comanda:

```
bin/main JAVA_PATH -e 2 -t > src/iteracio2/test-iteracio2.out
```

On JAVA_PATH en el meu cas és: */usr/java/j2sdk1.4.2_03*.

Com es pot comprobar per la comanda, el fitxer resultant de la execució del joc de proves és el *src/iteracio2/test-iteracio2.out*.

En la sortida del joc de proves es mostra la hora just abans de cridar el motor, la sortida normal de la classe Test (que ens mostra la informació a mida que reb finalitzacions de cerca), i la hora de finalització de cerca de ruta. Així podem veure qué s'ha trobat per cada domini del joc de proves i quant ha trigat en trobar-ho.

Tenint en compte el primer objectiu (millora de rendiment), i mirant el temps mitjà que es triga en analitzar una ruta (uns 4-5 mins), podem concloure que el temps de resposta continua sent gran, però que és acceptable tenint en compte que treballem amb el temps de resposta d'internet.

Per altra banda, tot i que ara ja podem realitzar cerques per nom de domini no associat a una IP (objectiu 2), continuem constatant que el nombre de registres LOC informats és mínim. Tot i utilitzar els 2 algoritmes de cerca per al registre LOC, els resultats de la execució del joc de proves són els següents:

Nombre de rutes analitzades: 11
Temps de resposta mitjà per ruta analitzada: 49'03'' / 11 = 4'27''
Nombre de dominis diferents analitzats: 138
Nombre de dominis amb LOC: 18
Nombre de dominis amb mail: 92
LOC's informats: 13,04%
Mail administrador informat: 66,66%

(Nota: Els resultats s'han calculat a partir del fitxer *src/iteracio2/test-iteracio2.out* amb un petit programa fora de l'àmbit del projecte.)

Com es pot comprobar, els registres LOC trobats són molt pocs, però si a més tenim en compte que diferents dominis poden haver aconseguit el registre LOC d'un mateix subdomini, realment el resultat no és bo.

En canvi, la direcció de correu del administrador es troba informada en un tant per cent raonable de casos.

Després de observar aquests resultats s'ha revisat el comportament de cada algoritme de cerca. Les conclusions són greus: la cerca per subxarxa no aconsegueix cap resultat. En el 0% dels casos tractats no s'ha trobat cap adreça de xarxa de tipus A, B o C que tingui informat una màscara de subxarxa (veure funcionament de l'algoritme al punt *2.2.3 Implementació*). Per tant, cada cop que executem l'algoritme de cerca per subxarxa especificat al RFC 1876 (A Means for Expressing Location Information in the Domain Name System) només aconseguim un domini candidat. També cal dir que de tots els dominis candidats que han aparegut en les diferents execucions de prova, en cap ocasió s'ha trobat un registre LOC associat.

2.3. Iteració 3: Interficie.

En aquesta iteració volem contruir la Interficie que finalment veurà l'usuari. Aquesta permetrà a l'usuari introduir un domini a buscar, llençarà el motor de cerca pel domini introduit, i finalment visualitzarà els resultats en la pantalla.

2.3.1. Recollida de requisits.

Com que la aplicació que estem construint és un localitzador geogràfic d'adreces IP, la part central de la pantalla serà un mapa mundi que ubiqui els punts localitzats. La resta de parts de la pantalla seran els elements necessaris per a gestionar la informació i interacció de l'usuari.

De fet, el problema en aquesta subfase és el mapa. Així que s'ha trobat una projecció bastant fàcil de gestionar a nivell de coordenades x i y: la projecció de Miller.

Com que el únic mapa que s'ha trobat en llibreries especialitzades és un mapa mundi de la projecció cilíndrica de Miller, no tindrem en compte la projecció equidistant de Miller, que pel nom es pot deduir fàcilment que hauria sigut molt més fàcil de gestionar. De totes formes la projecció cilíndrica de Miller ens facilitarà visualment la comprensió del mapa mundi.

La projecció cilíndrica de Miller projecta la superfície de la Terra en un plà rectangular, on les longituds són linies verticals paral.leles i equidistants. Les latituds segueixen una distribució no uniforme, tot i que es manté la paral.lelitat. Per tant es fa necessaria una fòrmula que resolgui la transformació d'un punt en la esfera a la projecció. Aquesta transformació està explicada en la url http://mathworld.wolfram.com/MillerCylindricalProjection.html.

Per la resta de la interficie només necessitarem la API de Java.

2.3.2. Anàlisi i Disseny.

La interficie s'anomenarà Pantalla, i implementarà la interficie Interficie (veure diagrama 2.2.2), així podrà mostrar la informació que li vagi arribant del motor de cerca.

L'aspecte visual de la pantalla determinarà el seu disseny, així que és el primer punt a tractar. Abans de fer cap diagrama, però, em de llistar tota la informació a mostrar:

- Camp d'entrada de l'usuari.
- Llistat de IP's de la ruta.
- Informació relacionada amb una IP (estructura InfoIP, veure diagrama 2.2.2)
- Mapa mundi amb els punts de les localitzacions geogràfiques de les IP's de la ruta.

Així doncs tenim dos grans elements: el mapa i la llista de IP's trobades. Aquests ocuparan la major part de la superficie de la pantalla.

Per altra banda cal tenir present que el motor de cerca retorna relativament poca informació, que sempre es podria ampliar en un futur. Aixi doncs haurem de preveure alguna manera de mostrar la informació que no ens hipotequi una possible futura ampliació d'informació.

Com que el mapa mundi serà bastant gran, la pantalla només mostrarà el mapa i la llista de IP's de la ruta, de forma que la llista només mostri la informació més rellevant de cada IP. Posteriorment, fent doble click a una IP de la llista apareixerà una nova finestra amb tota la informació relacionada amb la IP.

El diagrama de la pantalla queda així:

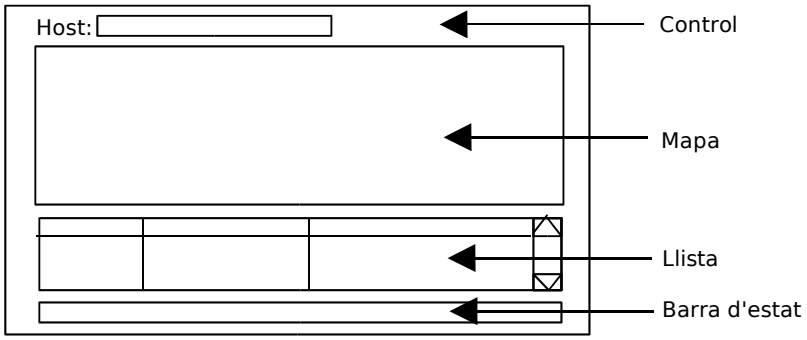


diagrama 2.3.2.1

El diagrama UML resultant d'aquesta pantalla és:

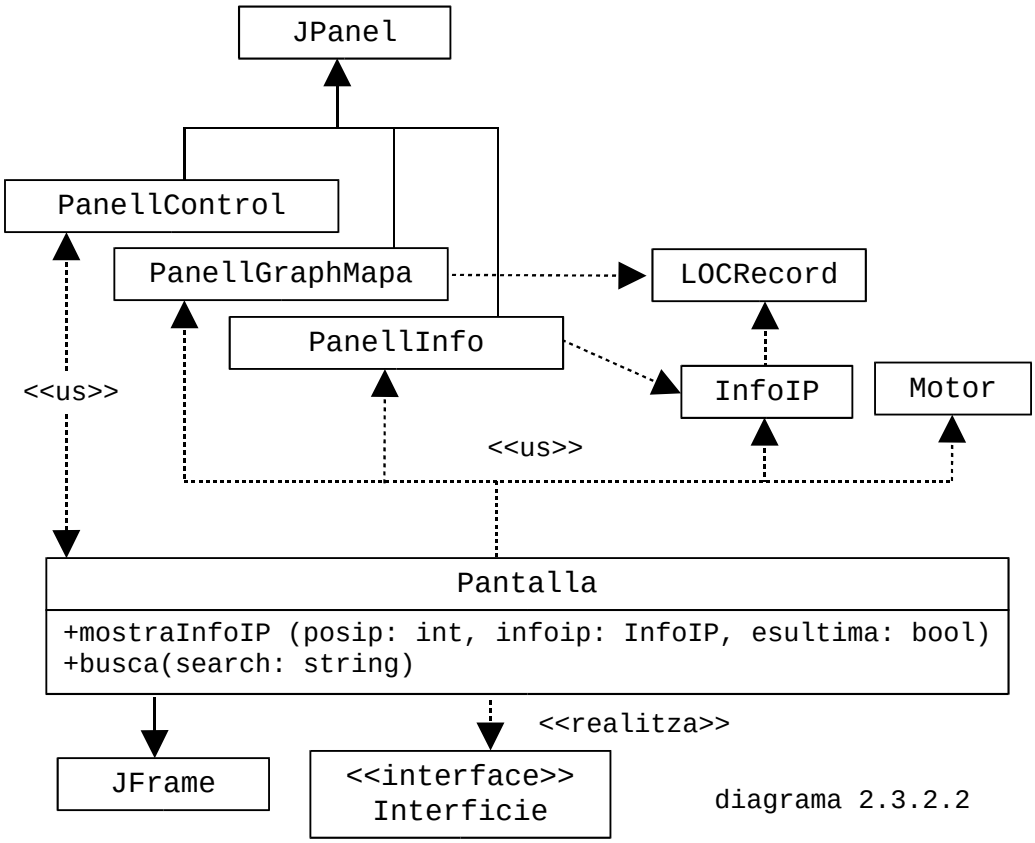


diagrama 2.3.2.2

On PanellInfo conté la llista, PanellGraphMapa conté el mapa i PanellControl el camp de domini a buscar:

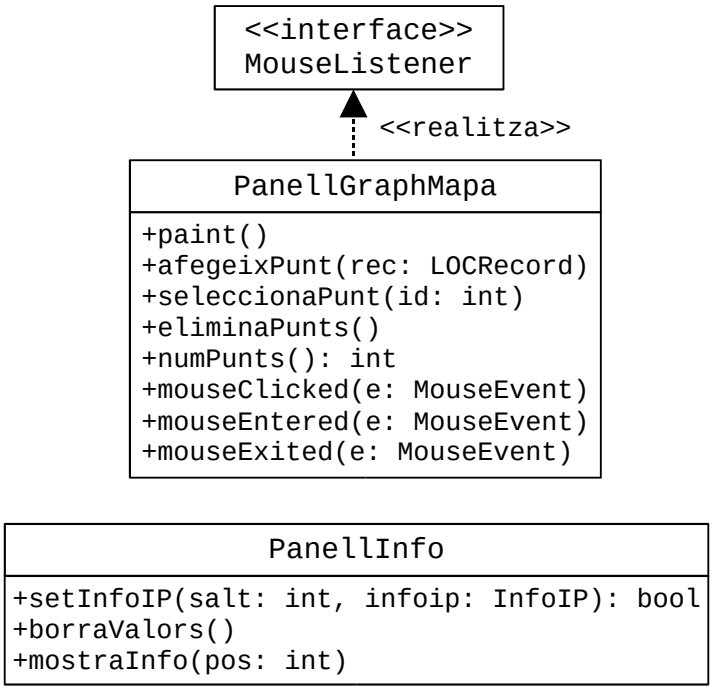


diagrama 2.3.2.3

Com es pot comprobar, s'ha repartit el contingut de la pantalla en panells, i cadascun realitzarà la seva feina independentment dels altres.

Cara a la implementació caldrà especificar bé la finalitat de cada classe:

- *Pantalla*: Gestiona els panells i el motor de cerca.
- *PanellControl*: Reb la entrada de l'usuari, i a través de *Pantalla*, llença la execució del motor de cerca.
- *PanellGraphMapa*: S'encarrega de la gestió del mapa. Això significa que a part de poder situar un punt en el mapa haurà de poder oferir algun sistema per facilitar la visualització dels punts, és a dir, haurà d'oferir la capacitat de fer zoom amb el ratolí. També haurà d'oferir la possibilitat d'afegir punts i sel.leccionar-los (per si hi ha més d'un i l'usuari vol saber a quina de les màquines pertany).
- *PanellInfo*: Mostra la informació. Inicialment només mostrarà les característiques més importants de la llista de IP's de la ruta. També oferirà la forma de poder accedir a la resta d'informació recopilada per cada IP. Finalment s'encarregarà de gestionar els punts del *PanellGraphMapa*, ja que *Pantalla* només enviarà la informació rebuda al panell d'informació.

2.3.3. Implementació.

Com en la iteració anterior, el primer que hem de fer és canviar la capçalera de totes les classes per que ara corresponguin al package iteracio3 en comptes de la 2. Aquests canvis, també com en la iteració anterior, repercuteixen en el nom de la capçalera de la llibreria nativa (veure fitxer *src/iteracio3/Motor.h*), i per tant haurem de modificar el nom de la funció que la implementa (fitxer *src/iteracio3/trace.c*) abans de compilar res.

Veient el disseny del diagrama 2.3.2.2, la part principal a implementar és la classe *src/iteracio3/Pantalla.java*. Aquesta classe realitzarà les funcions de la pantalla principal, i per tant gestionarà els diferents panells que visualitza (que també estan definits en el diagrama 2.3.2.2) i el motor de cerca. Com que en Java

utilitzarem les classes del paquet swing (paquet per a realitzar entorns visuals de finestres), aquesta classe heretarà directament de la classe *javax.swing.JFrame*, que implementa totes les funcionalitats que necessitem.

Les funcions bàsiques de *Pantalla* són: a partir de la funció *iniciar()* crear l'entorn gràfic (on definim les característiques dels diferents panells a mostrar i els carreguem); a partir de la funció *busca(abuscar: string)* que cridarà el panell de control, preparar i engegar el motor de cerca, i netejar el *PanellGraphMapa* de punts a través de la funció *PanellGraphMapa.eliminaPunts()*; i a partir de la funció *mostraInfoIP(posip: int, infoip: InfoIP, esultima: bool)*, desde on es rebran les finalitzacions de totes les preguntes que es facin, s'informarà al panell PanellInfo de les dades que ha de mostrar.

També cal dir que gestionarà la barra d'estat, de forma que mostrarà si la aplicació està realitzant la cerca o si aquesta està finalitzada.

Així doncs tenim que la classe *Pantalla* gestiona els recursos bàsics de la aplicació sense tenir que implementar els aspectes gràfics i la funcionalitat concreta de cada panell.

La classe *src/iteracio3/PanellControl.java* s'encarrega de la entrada de l'usuari. Com em vist en el diagrama 2.3.2.1, aquest panell es limitarà a subministrar una caixa de text per que el usuari pugui introduir una cadena de text per a cercar. Un cop l'usuari teclegi una cadena i premi la tecla *intro*, la classe *PanellControl* utilitzarà la referència a *Pantalla* passada com a paràmetre en el constructor per a cridar a la funció *Pantalla.busca(abuscar: string)*.

Com tots els panells, la classe *PanellControl* heretarà de la classe *javax.swing.JPanel*, que defineix les característiques bàsiques dels panells.

La classe *src/iteracio3/PanellInfo.java* s'encarrega de mostrar la informació trobada i enmagatzemada als contenedors InfoIP. Com ja s'ha comentat abans, la forma de fer-ho és a través d'una llista de característiques principals, i com que utilitzem swing, aquesta llista la implementa la classe *javax.swing.JTable*.

Tal com defineix el motor de cerca, un cop un thread de pregunta acaba, avisa a la interficie a través del mètode (veure diagrama 2.3.2.2):

```
Pantalla.mostraInfoIP(id: int, infoip: InfoIP, esultima: bool)
```

El identificador de salt ens anirà molt bé ja que així la llista sabrà a quina fila ha de situar els resultats rebuts, i per tant, *PanellInfo* suministrarà un mètode que permetrà aprofitar aquesta informació rebuda per *Pantalla* (veure diagrama 2.3.2.3):

```
PanellInfo.setInfoIP(salt: int, infoip: InfoIP): bool
```

En aquesta llista només es mostraran el número de salt, la IP, el nom, i la direcció de correu del administrador. Per accedir a tota la resta d'informació s'haurà de fer doble click a un element de la llista, acció que crearà una nova i petita finestra amb tota la informació recopilada, inclosa la informació mostrada en la llista.

Com que aquest panell és qui gestiona els resultats de la cerca d'una ruta, és aquest mateix panell qui s'encarrega, directament, de cridar als mètodes de *PanellGraphMapa* que permeten a aquest gestionar els punts localitzats de forma gràfica. Com ja s'ha comentat, les funcions que utilitzarà el *PanellInfo* seran:

```
PanellGraphMapa.afegeixPunt (id: int, reg: LOCRecord)
PanellGraphMapa.seleccionaPunt (id: int)
```

Per a poder cridar les funcions de *PanellGraphMapa* el constructor del *PanellInfo* necessita com a paràmetre la referència al panell del mapa.

Els punts s'afegiran automàticament al *PanellGraphMapa* quan es rebi un contenedor *InfoIP* amb el registre LOC informat. Acte seguit es sel.leccionarà l'element afegit al mapa.

Finalment, si l'usuari fa click damunt un element de la llista que té un punt associat, aquest es sel.leccionarà altra vegada, acció que portarà al *PanellGraphMapa* a canviar el color del punt sel.leccionat (veure explicació detallada de la classe *PanellGraphMapa*).

La última classe implementada en aquesta iteració és la classe *src/iteracio3/PanellGraphMapa.java*.

Aquesta classe, com tots els panells, heretarà de la classe *javax.swing.JPanel*, que pertany al paquet gràfic swing de Java. De totes formes, la funcionalitat la implementa a través de la llibreria gràfica awt de Java, ja que és la llibreria que permet la manipulació de les primitives gràfiques. Això és degut a que el mapa és una imatge que canvia de tamany, i a que els punts no són imatges.

Així doncs, la part principal de la classe és el mètode *paint (Graphics g)*. Aquest és l'encarregat de pintar la pantalla, i és cridat per la Java Virtual Machine reiteradament. Per tant, la filosofia d'implementació es basa en tenir un estat del panell, i en funció d'aquest, pintar una cosa o altra. En el nostre cas l'estat defineix el tamany del mapa, les coordenades orígen del mapa, i la llista de punts que s'han de pintar. Així el panell podrà pintar el mapa (fitxer *imgs/world10.jpg*) amb el tamany que calgui, i els punts que hi hagin en l'estat.

Gràcies a que la funció *paint (Graphics g)* s'executa reiteradament, qualsevol canvi d'estat provocat per la interacció de l'usuari es veurà al següent refresc de la pantalla, és a dir, automàticament.

S'ha pensat definir el tamany i la posició orígen del mapa a causa del zoom que el panell implementa. Aquest s'utilitza amb el ratolí: el butó esquerra apropa la vista del mapa, i el butó dret la allunya. Com que volem que el zoom es pugui aplicar a qualsevol part del mapa, en el moment de canviar el tamany del mapa fent un click, la aplicació guarda el punt on s'ha fet click per a calcular el nou orígen del mapa a partir d'aquest centre. És a dir, el punt del mapa on es faci click serà el nou centre del mapa en la següent vista.

El zoom es calcula a partir d'un factor de proporcionalitat respecte el tamany original del mapa (1897 píxels d'amplada per 1332 píxels d'altura), de forma que un factor de 200 implica que el tamany final que es veurà a la pantalla és exactament el doble del tamany original. Com que serà més gran que la pantalla de l'usuari, el mapa quedarà retallat pels limits de la finestra.

Per facilitar una navegació senzilla amb el mapa, quan es fa click amb el ratolí (veure funció *mouseClicked (MouseEvent e)*) s'augmenta o redueix 50 punts el factor fins als limits (35, 200). Això fa que només hi hagi tres vistes diferents del mapa. De totes formes la aplicació està preparada per treballar amb qualsevol factor.

El mapa (fitxer *imgs/world10.jpg*) està extret d'una reducció i d'una neteja de l'original escanejat. La reducció ha servit per reduir el pes del fitxer i s'ha acompanyat d'una reducció de qualitat JPG al 75%. Aquesta reducció de qualitat fa que el tamany del fitxer sigui més petit, però alhora també perdi qualitat d'imatge. En el nostre cas interessava que la aplicació, que al arrancar ha de carregar el fitxer del mapa, no trigués gaire, per tant s'ha buscat un punt intermig entre qualitat i pes. El resultat és que al fer el màxim de zoom els textos costen molt de llegir, per lo que és de deduir que en aquesta fase podríem haver optat per sacrificar el temps d'arrencada en benefici de la qualitat de la imatge final. En qualsevol cas es conserven els originals, i en cas de ser necessari per a l'usuari una millor qualitat d'imatge, es pot substituir el fitxer del mapa.

La visualització del mapa mundi és la estructura principal del panell, però l'objectiu final és la visualització dels punts en aquest mapa. Per a fer-ho, el panell ofereix les funcionalitats abans comentades (funcions *afegeixPunt(id: int, reg: LOCRecord)*, *seleccionaPunt(id: int)* i *eliminaPunts()*).

Aquestes funcions permeten situar un punt en el mapa, sel.leccionar un punt perque canvii de color i es distingeixi dels altres, i esborrar els punts per començar una nova cerca.

Tots els punts que s'afegeixen es guarden en una llista i queden vinculats al identificador associat (veure primer paràmetre de la funció *afegeixPunt*). D'aquesta forma, quan es vulgui sel.leccionar un punt només caldrà especificar el identificador del punt. En el nostre cas, l'identificador dels punts equival al número de salt ja que, com a molt, la aplicació mostrarà tants punts com el màxim número de salts.

La part més important del component és la localització d'un punt en el mapa. Aquest càlcul es realitza en el moment que s'afegeix un punt, tal com es pot haver deduït pel segón paràmetre de la funció *afegeixPunt*, que és un registre LOC d'un servidor DNS.

Localització d'un punt en el mapa de la classe PanellGraphMapa

Per calcular la posició del punt partim de la imatge que visualitza el mapa mundi i del registre LOC enmagatzemat a un servidor DNS.

La imatge del mapa és una projecció cilíndrica de Miller, i per tant fa que totes les longituds quedin paral.leles i, en teoria, equidistants; i que totes les latituds quedin paral.leles, però no equidistants. Aquestes últimes segueixen una distribució semblant a la del següent diagrama:

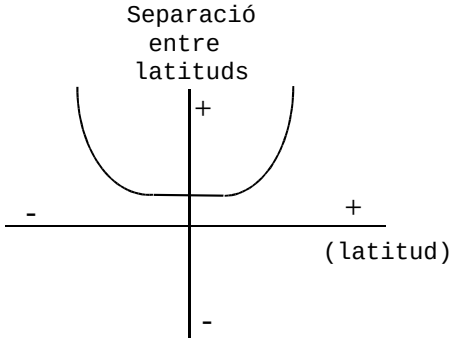


diagrama 2.3.3.1

Com que ens cal fer una transformació a apartir d'una latitud i una longitud donades (que estaran informades al registre LOC) a una coordenada del mapa, necessitem saber quina és la fòrmula que realitza la transformació de la latitud. Extret de la url mathworld.wolfram.com/MillerCylindricalProjection.html, la fòrmula de la transformació de la coordenada latitud en y és la que segueix:

$$y = 5/4 \ln[\tan(\pi/4 + 2/5 \phi)]$$

On λ correspon a la longitud del punt en el globus, λ_0 la longitud central utilitzada per la projecció, i ϕ la latitud del punt.

El problema d'aquesta fòrmula, però, és que no té en compte la escala del mapa i que no és intuitiva. Així doncs, després d'haver fet algunes proves per veure quins resultats donava la fòrmula amb diferents latituds, s'ha vist que no es podia utilitzar a causa que l'angle és un factor massa petit respecte la *tan(π/4)*, i amb totes les latituds pròpies del mapa (fins a 80º N i fins a 80º S) els resultats sempre donàven el mateix.

Per altra banda, el mapa que tenim no pinta les longituds ni les latituds de forma equidistant. Per tant, s'ha arribat a la conclusió que el mapa aproximarà el punt a la seva localització (les coordenades exactes es podran llegir a la finestra informativa que llença el *PanellInfo* quan es fa doble click a una màquina analitzada de la ruta).

Així doncs, com que aproximarem la localització trobada, el punt serà un cercle, de forma que assegurarem que dins el cercle estigui el punt i alhora evitarem el marge d'error (també informat al registre LOC).

Aquesta decisió també ens ajudarà a utilitzar sense modificacions la classe LOCRecord, que en comptes de retornar el contingut del registre (de forma que podríem calcular els graus, minuts i segons del punt) ens retorna només els graus en forma decimal.

La aproximació la farem a partir de les línies que pinta el mapa. Aquestes equivalen als múltiples de 10º (tant en longituds com en latituds), per tant, s'han hagut de localitzar visualment i guardar les coordenades resultants en dues llistes: la de longituds i la de latituds. Un cop tenim les coordenades de referència, els punts es calcularan proporcionalment a partir del reste de l'angle a la distància entre les dues coordenades de referència més pròximes.

Un exemple podria ser una longitud de 55º S, cas en que recuperarem les coordenades x de la llista de longituds per a les longituds 50º S i 60º S (que equivalen als punts del mapa 644.1 i 593.9 respectivament), i es calcularà proporcionalment el reste (5º) a la distància entre les longituds limit. En aquest cas s'ha agafat el nombre 55 perque es vegi més clarament que el resultat és justament el punt del mig, és a dir:

L = 55 mod 10 = 5, coordxmax = 644.1, coordxmin = 593.9

desplaçamentx / L = coordxmax - coordxmin / 10
desplaçamentx = 5 * (644.1 – 593.9) / 10 = 25.1
coordenada final = coordxmin + desplaçamentx = 619

que justament és el punt mig entre 644.1 i 593.9.

Evidentment, aquesta aproximació “proporcional” a les coordenades de referència més pròximes simula una distribució linial de les longituds, correcte ja que segons la projecció les longituds són equidistants, i per tant la seva distribució en el mapa és linial. Però en el cas de les latituds, el resultat tindrà més marge d'error degut a que simulem una distribució exponencial a partir de funcions linials. El següent diagrama mostra com s'ha transformat la distribució teòrica de les latituds a la següent distribució real (comparar amb diagrama 2.3.3.1):

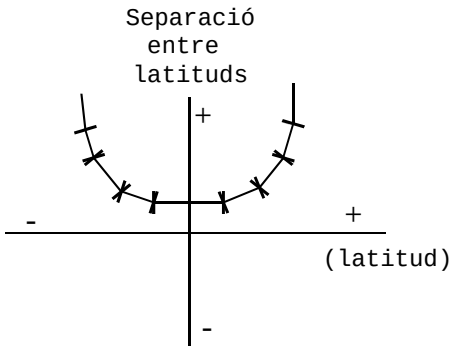


diagrama 2.3.3.2

2.3.4. Test.

En les primeres execucions s'ha observat que la llibreria dnsjava no retornava correctament les coordenades, i després de parlar via mail amb el seu autor (Brian Wellington), vem estar d'acord que hi havia una errada al codi font de la llibreria, exactament en la classe *src/org/xbill/dns/LOCRecord.java* en la linia 252:

```
/*** CODI ERRONI ***/
public double
getLatitude() {
   return (double)(latitude - (1<<31)) / (3600 * 1000);
}

/*** CODI OK ***/
public double
getLatitude() {
   return (double)((int)latitude - 0x80000000) / (3600000.0);
}
```

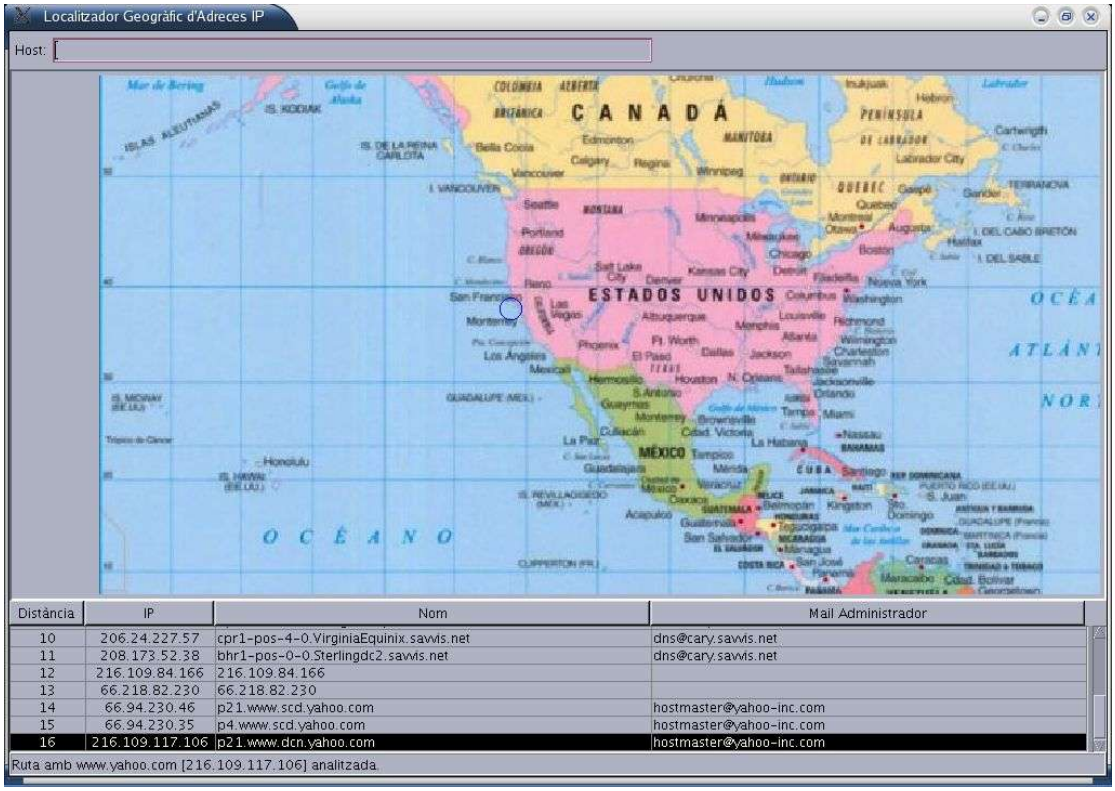
En la antiga linia 257 hi havia el mateix error però amb la funció *getAltitude ()*. Els dos errors en la llibreria s'espera que estiguin fixats en la següent actualització de la llibreria.

Tant per a la compilació com per a la execució s'ha utilitzat el script *bin/main* comentat en el punt 2.1.4, Nota 2; i detallat al punt *7.1 Annexe 1: Script bin/main*. S'ha fet amb la següent comanda:

```
bin/main JAVA_PATH -e 3
```

On JAVA_PATH en el meu cas és: */usr/java/j2sdk1.4.2_03*.

Un cop executada la comanda el resultat ha de ser una finestra java que s'obre. De fet la classe *Test*, que és la que llença el script *bin/main*, crea una instància de Pantalla, configura els paràmetres visuals, i inicia la seva execució.



La pantalla anterior correspon al programa en execució i després d'haver analitzat la ruta amb www.yahoo.com.

En la part superior es pot veure el camp de text per on s'introdueix la màquina a analitzar (només cal introduir el text i prèmer la tecla intro per a començar la cerca). Just a sota s'observa el mapa un cop em fet dos zoom-in's (amb el botó esquerra del ratolí). Després ve la llista de màquines de la ruta analitzades. I finalment a sota de tot, hi ha la barra d'estat que ens informa de quina ruta està analitzant o si ja ha acabat la cerca.

Si fem doble click a una màquina analitzada ens apareixerà una finestra amb tota la informació trobada per a la màquina com la següent:

Tenint en compte que l'objectiu de la iteració actual era la implementació de la part gràfica de la aplicació, els detalls a comentar són els següents:

- La llista d'elements no acaba de actualitzar-se correctament, és a dir, mentre es van afegint nous elements no sempre es pot visualitzar automàticament les dades de la última màquina analitzada.
- El rendiment de la aplicació depen del motor de cerca i de l'entorn gràfic de Java. El motor de cerca s'ha analitzat a la iteració anterior, però el rendiment gràfic de Java es pot considerar relativament lent (aquest és un fet conegut entre els desenvolupadors ja que tot component ha de passar pel coll de botella que és la Java Virtual Machine). Això repercuteix no tant en la carga de la imatge (que triga per el seu tamany en bytes) com en el temps de resposta al introduir una cadena de cerca. De totes formes el rendiment és acceptable.
- En el cas de la eina zoom el rendiment és bo degut a que el panell només tracta amb primitives gràfiques.

2.4. Iteració 4: Compatibilitat amb IPv6.

En aquesta iteració volem adaptar el motor de cerca a la versió 6 del protocol IP.

2.4.1. Recollida de requisits.

Com que en aquesta iteració partim del motor de cerca inicial, el primer que cal fer és identificar les parts del codi que treballen exclusivament amb dades pròpies del format IPv4.

Gràcies a la utilització de les adreces IP com a cadenes de text, i gràcies a la utilització de la classe *java.net.InetAddress* de la API de Java (que és compatible amb les dues versions del protocol IP ja que també treballa amb la seva representació textual), les úniques parts afectades per aquesta iteració són:

- Classe *src/iteracio4/Pregunta.java*: L'algoritme *rastrejaSubxarxa (byte[] ip)* suposa que el array de bytes representa una IPv4.
- Llibreria nativa *src/iteracio4/trace.c*: La llibreria utlitza estructures exclusives d'IPv4. A part omple byte a byte una petició d'echo en un paquet IPv4. Així doncs quedarà afectat gairebé en la seva totalitat.

També cal dir que la part de cerca DNS implementada per la llibreria lliure dnsjava es compatible amb les adreces IPv6, tot i que aquest no és el mateix problema que ens ocupa en la iteració (ja que una cosa és la comunicació i l'altra és la informació de les bases de dades DNS).

El problema el tenim amb l'entorn de proves. Actualment hi ha diferents xarxes que estant treballant amb les dues versions d'IP (la 4 i la 6), com la 6bone, de forma que ofereixen la possibilitat de testejar els programes i el rendiment de la xarxa per a la IPv6. El problema està en que la comunicació s'ha de “tunelitzar”, és a dir, s'ha de transportar un paquet IPv6 dins un paquet IPv4 fins a una xarxa compatible amb IPv6. Per a fer-ho cal tenir el sistema operatiu preparat per treballar amb les dues piles IP (tenir-lo preparat normalment requereix recompilar el nucli), donar-se d'alta d'algun portal que ofereixi el servei “Tunnel broker” (com ara http://tb.tid.euro6ix.org:8080/tb/ de Telefònica), activar el servei i posteriorment fer les proves.

Com que aquestes proves no es podran validar per part dels correctors del projecte, i tenint en compte que l'àmbit del projecte pot quedar sobrepassat per aquesta nova característica, s'ha arribat a la conclusió que no es farà la adaptació a la versió 6 del protocol IP. Així doncs la iteració queda abortada i no s'adjuntarà cap carpeta amb cap mena de fitxers que indiquin que hi ha hagut alguna activitat.

3. Valoració econòmica

El cost econòmic del projecte queda desglossat en: material i hores treballades.

El material queda limitat al mapa mundi a mostrar per el programa. Aquest s'ha hagut de comprar a una llibreria i escanejar a una copisteria, i el total ha sigut de 20€.

Les hores treballades són la part més important del cost econòmic. Això és degut al seu nombre: aproximadament unes 260 hores. I comptant un salari d'enginyer tècnic informàtic en pràctiques, on la hora treballada rondaria els 6 €, el cost final s'aproxima als **1580 €**.

4. Conclusions

Primer de tot cal analitzar la realització de cada objectiu definit al punt *1.2. Objectius del TFC*:

Respecte el punt de visualitzar tots els punts d'una ruta en el mapa cal dir que l'objectiu no està acomplert: només es visualitzen menys del 13% de les màquines analitzades (veure punt *2.2.4 Test*). Això és degut a que partim de la informació trobada als servidors DNS únicament. En cap cas intentem extreure més informació d'altres servidors per a poder creuar-la amb la dels DNS i així poder calcular la localització d'una màquina (un exemple d'aquesta tècnica seria implementar el protocol WHOIS, on per cada registrant del domini queda informat el ZIPCODE, una especie de codi postal que està associat a una regió concreta, de forma que si un domini no tingués el LOC informat es pogués determinar la localització del registrant). Una altra tècnica que es podria haver incorporat seria la el.laboració d'una base de dades on estiguèssin informades els dominis de les principals xarxes dels països, de forma que cada cop que no es trobés la localització d'una màquina, poguèssim calcular una aproximació en base a la localització de la última màquina analitzada que pertanyés a una d'aquestes xarxes conegudes (cal dir que aquesta tècnica podria donar problemes a causa de la alta varibilitat dels noms de les màquines i de les rutes internacionals, i de la estaticitat dels continguts d'aquesta base de dades).

Respecte el segon punt, recollir el màxim d'informació útil sobre cada IP tractada, només es recull com a informació útil les direccions de correu dels responsables del domini, i la font d'informació de cada camp extret dels servidors DNS. Es podia haver recuperat la distància en temps respecte cada màquina de la ruta, la informació associada a un domini enmagatzemada a un servidor WHOIS (comentat en el paràgraf anterior), etc. Cal dir que aquesta nova informació no s'ha considerat útil per a una localització certera d'una màquina de la ruta, i per tant s'ha considerat complementaria a l'objectiu principal del projecte.

Respecte el tercer punt, el rendiment, es pot considerar satisfet tenint en compte les limitacions imposades per el temps de reposta d'internet.

Finalment, respecte el quart punt, la compatibilitat amb IPv6, no s'ha pogut satisfer pel cost que introduïa al projecte sumat al desviament de l'objectiu principal del projecte.

Així doncs, podem concloure que el projecte no ha satisfet, en general, les expectatives inicials. Ha servit per descubrir que la informació publicada als servidors DNS dista molt de la informació útil que necessita una aplicació com aquesta, i també com a punt de partida per un nou i més exhaustiu anàlisis de quina informació es pot extreure d'internet que serveixi per a ubicar la situació geogràfica d'una màquina.

5. Glossari

- JNI: Java Native Interface. Arquitectura Java pensada per a comunicar processos Java amb processos d'altres tecnologies.
- Class: Fitxers executables (bytecode) de Java. Aquests s'executen a través del fitxer binari java.
- Java Virtual Machine: Màquina virtual de Java, intèrpret del bytecode dels fitxers class que executa.
- Llibreria nativa: Funció implementada per a una plataforma concreta.
- Constructor: Funció que s'encarrega de crear un objecte.
- Referència: Direcció de memòria d'un objecte.
- Zoom-in: Apropar la prespectiva.
- Zoom-out: Allunyar la prespectiva.

6. Bibliografia

- Apunts assignatura Xarxes de Computadors II (UOC)
- Apunts assignatura Enginyeria del Programari I (UOC)
- RFC 0792: Internet Control Message Protocol
- RFC 1034: Domain names - concepts and facilities
- RFC 1035: Domain names - implementation and specification
- RFC 1876: A Means for Expressing Location Information in the Domain Name System
- RFC 1886: DNS Extensions to support IP version 6
- RFC 2460: Internet Protocol, Version 6 (IPv6) Specification

URL's

- API de Java: http://java.sun.com/j2se/1.4.2/docs/api/
- API de dnsjava: http://www.xbill.org/dnsjava/doc/
- Projecció: http://mathworld.wolfram.com/MillerCylindricalProjection.html

7. Annexos

7.1. Annexe 1: Script bin/main

El script està pensat per automatitzar el procés de compilació i execució de cada iteració. Això és degut a que per cada iteració s'ha d'executar el procés següent:

1. Compilar els fonts Java.
2. Generar la capçalera de la llibreria nativa.
3. Generar la llibreria nativa.
4. Definir la variable LD_LIBRARY_PATH.
5. Arrancar la aplicació.

Aquest script està pensat per ser cridat desde el directori pare, on està instal.lat el projecte. Això és degut a que l'script no reconeix el directori on està ubicat i necessita accedir a les carpetes *bin* i *src*.

Procés per a compilar una iteració:

```
bin/main JAVA_PATH -c num_iteracio
```

Procés per a executar una iteració:

```
bin/main JAVA_PATH -e num_iteració [-t | argument]
```

El JAVA_PATH és el directori on està instal.lat el Java. Aquest és necessari per a buscar els binaris executables Java i les llibreries Java necessaries per a compilar la llibreria nativa. Tot i que es podia especificar que la variable estigués definida en el PATH del sistema s'ha optat per passar-la per paràmetre.

En totes les iteracions la execució la arranca la classe Test (*bin/iteracioX/Test.class*), però en les dues primeres aquesta classe espera un argument, que és la IP o domini destí. En el script el paràmetre és opcional a causa de que en les iteracions superiors a la segona no requereixen el paràmetre.

L'argument igual a -t indica al script que ha de llençar el joc de proves del motor de cerca. Aquest recupera una llista de dominis a cercar d'un fitxer que s'anomena *src/iteracio2/joc_de_proves,* i va cridant a la classe *Test* de la iteració per cada domini de la llista. De moment com que el motor de cerca l'implementa la segona iteració només és útil en la execució de la segona iteració.

Abans d'executar qualsevol opció l'script valida els paràmetres d'entrada i valida que el directori des d'on se'l crida sigui el pare (ho realitza comprobant si existeix una carpeta anomenada *bin*). En cas d'execució també valida que l'usuari sigui root. Això ho fa comprobant que la variable del sistema USERNAME sigui root. En qualsevol cas la validació de l'usuari es pot saltar cridant al script de la següent forma:

```
bin/main JAVA_PATH -e num_iteració -noval [argument]
```

El script no comproba el fitxer */etc/resolv.conf*, que és on hi ha definits els servidors DNS que utilitzarà el resolvedor DNS de la llibreria dnsjava. Això és molt important ja que s'ha detectat un error en la llibreria, que consisteix en que no carrega correctament els diferents servidors i només carrega el primer. El problema apareix quan tenim el localhost com a primer servidor DNS en el fitxer, situació que impedeix al resolvedor Java trobar cap informació. Per tant, és molt recomenable

comprobar el fitxer */etc/resolv.conf* manualment en cas de que la aplicació no trobi cap resultat.

Finalment només resta comentar que en el procés de compilació no hi ha tractament d'error, de forma que el script es limita a llençar els processos, indiferentment de si fallen o no.