MEMÒRIA DEL PROJECTE FI DE CARRERA

AOP i Enginyeria inversa amb Objectes Distribuïts

Segon semestre, curs 2003/04

Autor:	Andrés Torralbo Sepúlveda
Professor responsable de l'assignatura :	Jordi Àlvarez Canal
Consultor:	Antoni Oller Arcas

Índex:

1 Introducció

1.1 Resum del projecte

El plantejament inicial d'aquest projecte, ha sigut fer ús de la nova programació orientada a Aspectes (AOP) per a fer tasques de reenginyeria.
De manera que amb l'ajut d'aquesta tecnologia, poguéssim extreure informació de la execució d'una aplicació, i així a partir d'aquesta informació es pogués obtenir del diagrama de cas d'ús.
Amb aquest diagrama ens podrem fer una idea del disseny i l'arquitectura de la aplicació inicial.

El semestre passar l'alumne Josep Lluís Lérida Monsò va aconseguir realitzar un sistema que permetia generar el diagrama de seqüència a partir de l'execució d'una aplicació.

En aquesta línia el nostre treball es basa en l'extensió d'aquest sistema per a que suporti l'instrumentalització d'aplicacions distribuïdes.

Al final, s'ha aconseguit construir un prototipus que ens permet obtenir diagrames de seqüència per a aplicacions distribuïdes, enllaçant les crides entre cadascun dels components remots.
D'aquests diagrames de seqüències al final tindrem un per a cada procés, independentment d'on s'executi cada part d'aquest.
Es a dir, dels diferents fils d'execució.
Ja que al ser una aplicació distribuïda pot ser que aquests fils d'execució es puguin donar a diferents maquines virtuals de Java (JVM).
En aquest moment es on radicava la dificultat principal del projecte, doncs s'ha d'estudiat i després implementat un mecanisme que ens uneix el codi que s'ha executat a cada JVM es un únic fil del diagrama de seqüència.

A continuació, per tal de entendre millor el projecte, descriurem breument cadascuna de les tecnologies aplicades.

1.2 Objectes Distribuïts

Els sistemes distribuïts sorgeixen per a donar solució a les següents necessitats:

- Repartir el volum d'informació
- Compartir recursos, tant de hardware com de software

La construcció de sistemes distribuïts presenta una solució que augmenta les nostres capacitats.
Així no ens veurem limitats per les restriccions d'una màquina, ja que podem emprar els recursos de tota una xarxa.

Els sistemes distribuïts es poden implementar emprant dos tipus de model:

- Client – Servidor
- Orientació a objectes

En el nostre cas ens centrem en l'orientació a objectes.
En aquest model hi ha una sèrie d'objectes que sol·liciten serveis (clients) i un conjunt d'objectes que els proporcionen (Servidors), tot això emprant una interfície d'encapsulació definida.

La programació distribuïda no és trivial:
- Hi ha múltiples models
- S'han de considerar molts aspectes de seguretat.
- Es mesclen diferents tecnologies
- Hi ha una gran dificultat de provar i depurar.

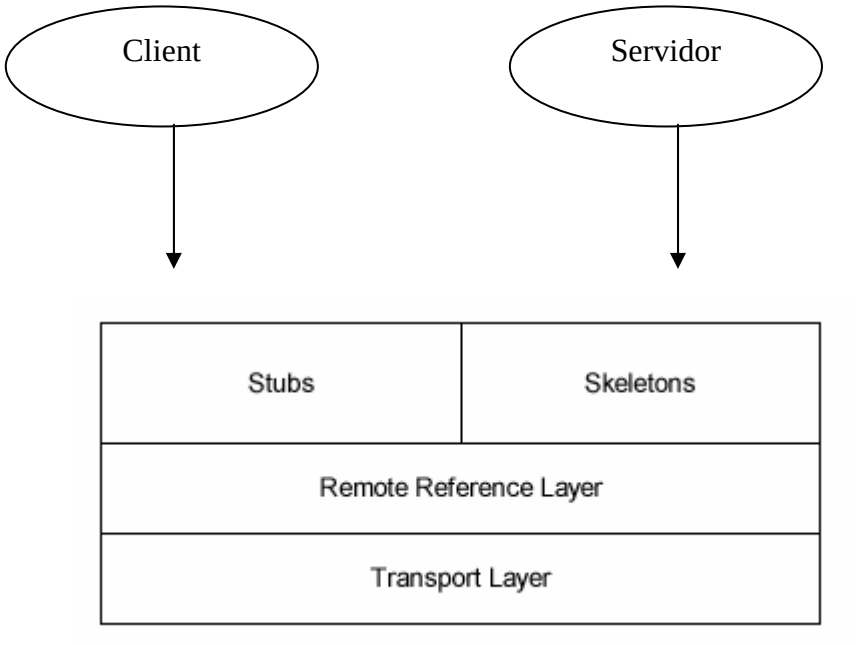
Però la principal preocupació és, com es comuniquen client i servidor?, amb quin entorn, llenguatge o eina?

En el nostre cas em escollit Java.

Java és una eina ideal per a la programació distribuïda, per la seva portabilitat, seguretat i el ampli ventall de components.
Emprant Java, ens podem oblidar del tipus de màquina en el que el fem servir, i el sistema operatiu, gràcies a la seva Java virtual Machine (JVM) que crea un entorn igual per a totes elles.

Una forma de crear un entorn distribuït en Java és utilitzar RMI.
RMI permet que objectes que s'estan executant en una màquina puguin emprar mètodes d'objectes que es trobin en maquines diferents. Sense adonar-se que són remots.

Per aconseguir això, RMI disposa de la següent arquitectura:



Arquitectura de RMI:

- stub/skeleton: és responsable de transferir dades a Remote Reference Layer, permet als objectes java ser transmesos a diferents espais de direccions.

- Remote Reference Layer és responsable de crear independència de los protocols.

- Transport Layer es responsable de establir les connexions als espais remots de direccions.

En RMI tota la informació sobre els serveis del servidor es donen en una definició de interfície remota.

1.3 Diagrames de Seqüència

Els diagrames de Seqüència són uns diagrames del estàndard UML on s'hi representen els Objectes que participen en una interacció respecte al temps.
D'aquesta manera es mostren les creacions, destruccions o crides als seus mètodes.

Formen part, juntament amb els diagrames de col·laboració d'un grup anomenat diagrames d'interacció.

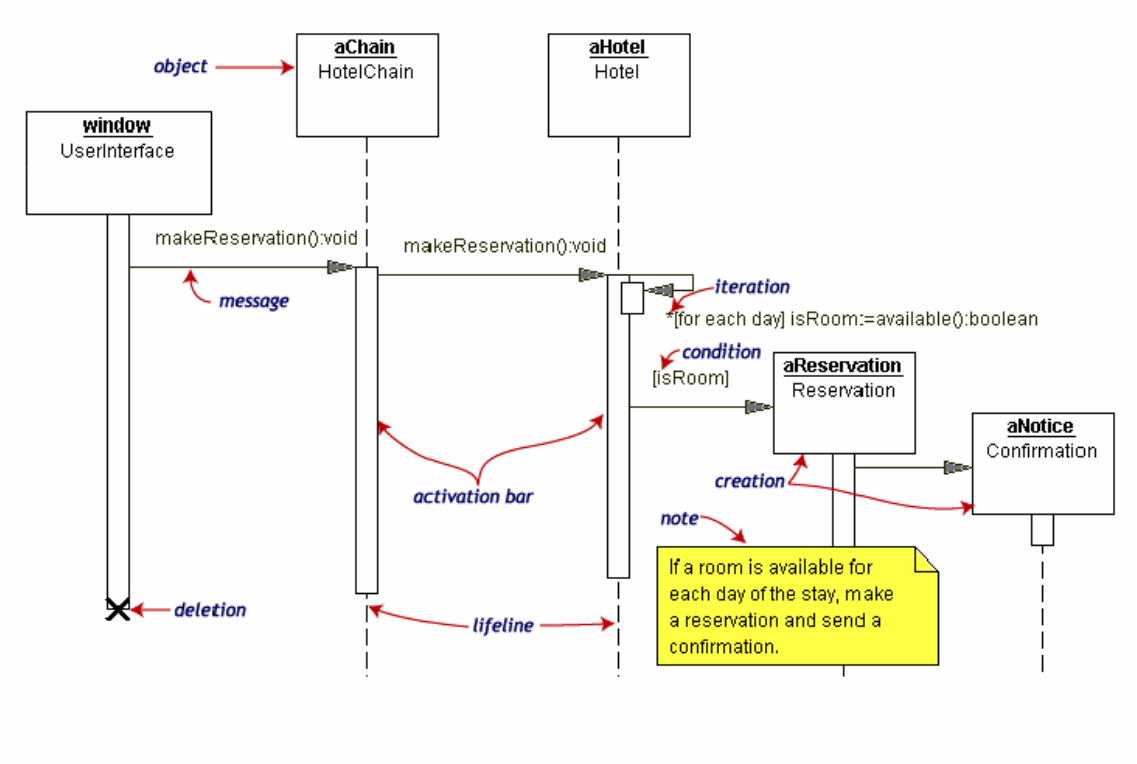
Els diagrames d'interacció (de seqüència i de col·laboració) són models dinàmics que descriuen com col·laboren els objectes que figuren un cas d'ús o un escenari.

En particular, un diagrama de seqüència és un diagrama d'interacció que detalla com s'executen les operacions en funció del temps: quins missatges són enviats, per quin objecte, a quin objecte i quan.

Per a representar els diferents elements emprem un disseny Standard:

- Un objecte es representa amb un rectangle
- El nom de la classe de l'objecte va precedit de (:)
- Es pot donar explícitament o no el nom de l'objecte
- El nom de l'objecte: classe es subratlla.
- Els objectes involucrats en el diagrama es llisten d'esquerra a dreta d'acord amb el moment que intervenen en la seqüència de missatges.
- Una línia de vida és el temps durant el qual un objecte existeix i es representa per una línia vertical puntejada.
- La barra d'activació representa el temps d'execució del missatge i es representa per un rectangle sobre la línia de vida.
- Un missatge es representa amb una fletxa. Una fletxa va des de l'emissari fins a la part superior de la *barra d'activació* del missatge situada a la línia de crida del receptor. La presència d'un missatge a un diagrama d'interacció implica l'existència d'una operació, amb la mateixa signatura que el missatge, a la classe de l'objecte receptor del missatge.
- Per a indicar que una nova operació d'un objecte ha estat invocada com a part de l'execució d'una operació prèvia, les barres d'activació s'aniuen.
- Els valors de retorn s'indiquen (opcionalment) fent servir una fletxa puntejada:

Per a veure millor tot això, a continuació donem l'exemple d'un diagrama de seqüències per a fer reserves d'hotel:



El que ens explica aquest diagrama de seqüències és:

Des de window s'envia un missatge makeReservation() a l'objecte HotelChain.

HotelChain envia un missatge makeReservation() a Hotel,

Hotel crida un mètode propi isRoom per veure si li queden habitacions

Si el mètode isRoom retorna true, vol dir que hi ha habitacions i Hotel crea un objecte Reservation.

I l'objecte Reservation crea un Objecte Confirmation.

L'asterisc de l'autocrida significa iteració (per assegurar que hi ha habitació lliure tots els dies d'estada l'hotel). L'expressió entre [ ] és una condició.

Per a obtenir els diagrames de seqüència s'ha fet servir una eina anomenada sequence a la que afegim el fitxer que ens crea la nostra aplicació, que té un format text determinat per a que el sequence l'interpreti, i obtenim el diagrama de seqüències.

1.4 Programació Orientada a aspectes (AOP) i AspectJ

1.4.1 Programació Orientada a Aspectes.

La programació orientada a Aspectes neix l'any 1996 en Xerox Parc.
El seu naixement bé donat per una sèrie de problemes que venia arrossegant la Programació orientada a Objectes i que havien de solucionar-se d'una manera eficient. Aquests problemes ja van ser detectats a 1970, però encara no tenien una solució clara.

El concepte principal de la programació orientada a Aspectes són els concerns.
D'aquesta paraula trobem infinites traduccions que cadascú li atorga com ara:

- Preocupacions
- Problemes
- Interessos
- Incumbències

En aquest projecte per a no embolicar més el tema l'anomenarem igual que amb anglès: concerns.

Els concerns són els diferents temes o assumptes dels que ens hem d'ocupar per a solucionar un problema.
De fet els concerns no es un concepte únic de l'AOP, sinó que es pot aplicar a tot. Per exemple, per a fer la declaració de la renda, quantes coses s'ha de fer fins arribar a la solució final. Haurem de fer un munt de concerns, com ara trucar per telèfon, demanar cita, preparar tots els papers, etc.

Llavors, qualsevol aplicació informàtica és en realitat un gran conjunt de concerns.

Veiem la descomposició en concerns d'una aplicació a mode d'exemple:

1- Verificació de credencials
2- Verificació de precondicions
3- Començament d'un transacció
4- Execució de la nostre funció de negoci.
5- Acabament de la transacció
6- Verificació de postcondicions.

Aquesta és un estructura que ens resulta molt familiar.
Però realment, només el punt 4 és el que volem que faci l'aplicació, tot el demés són una sèrie de concerns que probablement es repeteixen al llarg del nostre sistema.

Hi ha un tipus de concerns, que es donen a diferents parts del sistema que no tenen perquè estar relacionades, i que realitzen la mateixa acció, o lleugerament modificada. A aquests concerns els anomenarem "crosscuting concerns" però que la seva traducció és pot fer com incumbències transversals.

Exemples d'aquest crosscuting concerns són la connexió a una base de dades. Potser ens connectarem dins de molts objectes diferents i que no tenen res a veure, però la connexió serà sempre gairebé la mateixa, com a molt canviaran certs paràmetres com ara la identificació de la Base de dades a la que volem connectar.

Doncs bé, l' AOP com la nova extensió de la programació orientada a Objectes ens lliurarà d'aquesta sobrecarrega de codi repetit, que ens portaria de cap a l'hora de fer modificacions.

La idea de l'AOP és agrupar aquests crosscuting concerns en unes noves estructures anomenades Aspectes.

El fet de tenir fora dels Objectes aquests crosscuting concerns ens proporciona:

- Un disseny modular més pur.
- Una més fàcil evolució. Al no tenir que fer la modificació a més de un lloc.
- Reciclatge de classes.

A continuació veurem un eina d'implementació de AOP anomenada AspectJ i el seu funcionament.

1.4.2 AspectJ

AspectJ és una eina d'implementació de l'AOP basada en Java.
Va ser creada al 1998 en Xerox PARC.

El concepte del que AspectJ realitza és afegir al java les noves estructures anomenades Aspectes que defineixen crosscuting concerns.

Amb aquesta eina podrem afegir a punts concrets (joinpoints) dels objectes, l'execució dels crosscuting concerns que haguem definit.

Internament, el funcionament de l'eina és que en temps de compilació afegim codi (advices) a certs punts que nosaltres definim (pointcuts).
D'aquesta manera en lloc d'haver de repetir codi a cada objecte, li diem a que objectes i a quin lloc d'aquests volem afegir aquest codi que només tenim escrit una vegada al Aspecte.

Una vegada s'executa és un programa normal, que no té cap característica especial.

Anem a veure amb detall els conceptes del AspectJ.

1.4.2.1 Joinpoints

El joinpoint és el concepte base del AspectJ.
No és tracta de res que haguem d'implementar, és només el concepte d'aquells punts que podrem capturar emprant AspectJ.
Quan parlem de captura, volem dir que en aquest punt es podrà realitzar una acció que nosaltres especifiquem, i que en aquesta acció disposarem de certa informació sobre el context d'aquest joinpoint.

Exemples de joinpoints són:

- La crida i l'execució d'un mètode.
- La crida o execució d'un constructor.
- La lectura o escriptura sobre un camp.
- La inicialització d'un Objecte.
- Etc.

1.4.2.2 Pointcut

Els Pointcuts són elements que podem implementar amb la intenció de poder especificar joinpoints o conjunts d'aquests.
D'aquesta manera tindrem identificats els conjunts de joinpoints que volem capturar al nostre aspecte.

Un exemple de Pointcut del nostre projecte és:

```
pointcut  pcExecutaMain() :
      execution(* *.main(..))
      &&!execution(* java..*.*(..))
      &&!execution(* eines..*.*(..))     ;
```

En aquests pointcut estem definint un conjunt de joinpoints.
El que fa aquest codi es definir un pointcut anomenat pcExecutaMain() que és composa del conjunt format per tots els joinpoints que siguin la execució del mètode main de qualsevol classe però exceptuant aquelles execucions de mètodes que es facin des de llibreries que el seu nom comenci pel literal "java" o "eines".
Com es pot apreciar, emprem els operadors lògics && (and), || (or), ! (negació), per a fer operacions de conjunts sobre els joinpoints.

A mode d'exemple veiem algunes formes d'especificacions de joinpoints per a crear els pointcuts.

Primer veiem especificacions de call amb diferents usos de caràcters de control:

Especificació	**Descripció**
Call(public void C.M (String))	Call al mètode M de la classe C que té un argument String i retorna void i és públic.
Call(public void C.M (..))	Call al mètode M de la classe C que té qualsevol argument i retorna void i és públic.
Call(* C.M (..))	Call al mètode M de la classe C que té qualsevol argument i retorna qualsevol tipus i que no té perquè ser públic.
Call(* C+.M (..))	Call al mètode M de la classe C i les seves subclasses que té qualsevol argument i retorna qualsevol tipus .
Call(* *.M (..))	Call al mètode M de qualsevol classe que té qualsevol argument i retorna qualsevol tipus.
Call(* *.*(..))	Qualsevol call que es faci.
Call(public * eines..*.*(..))	Call a qualsevol mètode públic de totes les classes d'un paquet anomenat eines.

Les combinacions que em vist a la figura anterior es poden fer amb altres tipus de joinpoints, a continuació veurem alguns exemples:

Especificació	**Descripció**
Execution(void C.M (String))	Execució del mètode M de la classe C que té un argument String i retorna void.
get(int C.x)	Lectura del camp x de la classe C de tipus enter.
set(int C.x)	Escriptura del camp x de la classe C de tipus enter.
Handler(RemoteException+)	Execució del bloc catch iniciat per una RemoteException
Within(C)	Qualsevol pointcut dins de la classe C.
Cflow(call(* C.M(..)))	Tots els joinpoints que estan en el control flow de qualsevol crida al mètode M de la classe C incloent la crida a si mateix.
Cflowbelow(call(* C.M(..)))	Tots els joinpoints que estan en el control flow de qualsevol crida al mètode M de la classe C excloent la crida a si mateix.
This(JComponent+)	Tots els joinpoints on this és una instància de JComponent
Target(C)	Tots els joinpoints on l'objecte en el qual el mètode és cridat és del tipus C.
Args(String,..,int)	Tots els joinpoints on el primer argument és de tipus String i el últim de tipus int.

1.4.2.3 Advices

Els advices especifiquen el codi que s'executa quan es captura el pas del fil d'execució per un determinat Pointcut.

D'aquesta manera, quan un execució arriba a un determinat Pointcut, podem especificar que el codi del advice s'executi en tres moments: before, after o around de l'execució del Joinpoint.

Un exemple que tenim al projecte és l'advice que es llança amb el Pointcut pcExecutaMain:

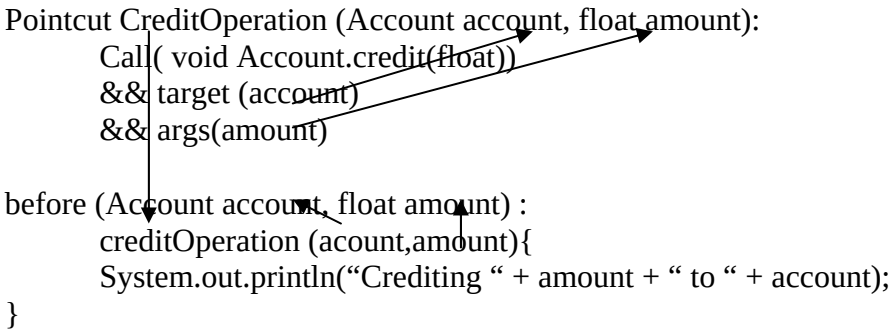
```
before():pcExecutaMain(){
   codi a executar
}
```

Aquest codi s'executaria just abans que ho fes el main.

1.4.2.4 Passant el context d'un Pointcut a un advice

Moltes vegades en un advice volem tenir certa informació sobre el context en que es trobava el Pointcut.
Ens és molt útil per exemple que si em capturat la crida a una funció, saber per exemple els paràmetres que li enviaven, o informació sobre l'objecte en que ens trobaven.

Veiem una implementació que agafa dades sobre l'entorn del Pointcut:



Amb aquesta implementació passem al advice l'objecte que s'executa i els arguments que li passem al mètode. El objecte és capturat per target(account) i els paràmetres per Args(account).

1.4.2.5 Exemple AspectJ

Anem a veure un petit exemple que ens fa molt més entenedor el funcionament del AspectJ.
Es tracta d'un exemple molt senzill que proporciones bones maneres a un mètode.

L'aplicació que anem a instrumentalitzar, en aquest cas una classe que anomenem HelloWorld amb 2 mètodes: say(String), sayToPerson(String,String)
Aquest mètodes l'únic que fan es treure per pantalla un missatge depenent dels Strings que passen com paràmetres.

```
public class HelloWorld {
    public static void say(String message) {
        System.out.println(message);
    }

    public static void sayToPerson(String message,String
name) {
        System.out.println(name + ", " + message);
    }
}
```

Ara el que farem és implementar un Aspecte que cada vegada que és crida a un dels mètodes de la classe a instrumentalitzar, abans d'executar-lo diu "Hola" i quan acaba diu "Adéu". Així li afegim bones maneres.

```
public aspect MannersAspect {
    pointcut callSayMessage() :
     call(public static void HelloWorld.say*(..));

     before() : callSayMessage() {
        System.out.println("Hola!");
    }

    after() : callSayMessage() {
        System.out.println("Adéu!");
    }
}
```

El que s'ha fet és un Aspecte amb un Pointcut que captura els Calls a tots els mètodes de la classe Helloworld que comencen pel literal "say".
Després hem creat un advice associat al before d'aquest PointCut , i que per tant s'executarà abans que el mètode, i que diu Hola!.
I un altre advice associat al after del PointCut , i que s'executarà després que el mètode, i que diu Adéu!.

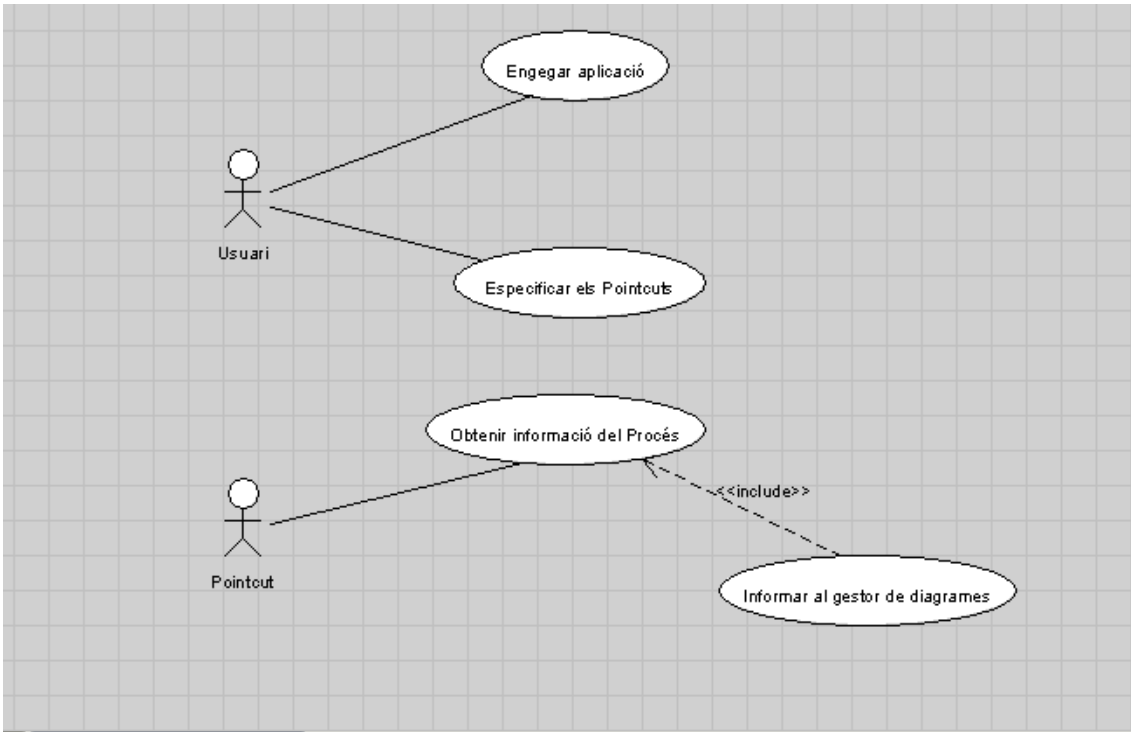
Per exemple si fem una crida a say("Text del Say") obtindríem:

```
Hola!
Text del Say
Adéu!
```

El seu funcionament és trivial. Però el concepte que amaga ens ofereix un gran ventall de possibilitats. Des de la més directa, com pot ser un tractament de log, com ara control d'accessos quan un mètode vol accedir a una base de dades, etc.
Tot un món de possibilitats sense haver de tocar el codi original i que podem variar quan vulguem sense afectar a l'aplicació.
I sobretot és una molt bona solució al codi que és repeteix una i altre vegada a Objectes diferents, però que sempre és el mateix.

2. Anàlisis

2.1 Diagrama de casos d'ús



2.2 Documentació Textual

2.2.1 Cas d'ús número 1: "Engegar aplicació"

Resum de la funcionalitat: Posada en marxa de l'execució de l'aplicació de la qual l'usuari vol crear el diagrama de seqüències.

Paper dins el treball de l'usuari: És el cas d'ús inicial i necessari per tal d'obtenir el Diagrama de seqüències de la aplicació escollida.

Actors: Usuari.

Casos d'ús relacionats: Cap

Precondició: S'ha de tenir l'aplicació compilada amb els Aspectes que ens proporcionaran les dades per a obtenir el diagrama de seqüències. També s'han de tenir registrades tots aquells objectes remots dels que l'aplicació en farà ús.

Postcondició: L'aplicació s'estarà executant amb les funcionalitats extres que li donarà l'aspecte encarregat de la obtenció d'informació necessària per a crear el diagrama de seqüències.

Descripció detallada: Encara que no és una funcionalitat pròpia del nostre projecte, es comenta la seva existència, per tal de aclarir que l'usuari ha d'executar la aplicació de la qual vol obtenir el diagrama de seqüències. A partir de l'execució d'aquesta s'aniran disparant els Pointcuts del Aspecte que n'obtindran la informació necessària, per tal de poder cridar al gestor de diagrames de seqüències, que a la seva vegada anirà conformant el diagrama de seqüències resultant.

2.2.2 Cas d'ús número 2: "Especificar els Pointcuts"

Resum de la funcionalitat: Especificació del JoinPoints que l'usuari vol capturar per tal de que surtin reflectits al diagrama de seqüències.

Paper dins el treball de l'usuari: L'usuari ha de definir els Pointcuts que vol que quedin reflectits al diagrama de seqüències.

Actors: Usuari.

Casos d'ús relacionats: Cap

Precondició: Cap

Postcondició: Els Joinpoints estan Especificats.

Descripció detallada: En aquest cas tampoc parlem d'un cas d'ús de l'aplicació en el més pur sentit. Ja que no es tracta d'una funcionalitat de l'aplicació, sinó de la possibilitat que aquesta proporciona de ser parametrizada, per tal d'adaptar-se a qualsevol aplicació de la qual es vulgui obtenir el diagrama de seqüències.
Per a aconseguir aquesta fita l'AspectJ ens proporciona la definició d'aspectes abstractes. L'aplicació té definits uns aspectes abstractes que l'usuari ha de concretar. Llavors no es pròpiament una funcionalitat del sistema, ja que l'usuari a de fer una petita implementació dels Pointcuts que vol reflectir al diagrama de seqüències.

2.2.3 Cas d'ús número 3: "Obtenir informació del Procés"

Resum de la funcionalitat: L'aspecte ha d'obtenir informació del procés que l'activat.

Paper dins el treball de l'usuari: Cap, es una funcionalitat activada pel Pointcut de l'aspecte.

Actors: Pointcut.

Casos d'ús relacionats: Informar al gestor de diagrames.

Precondició: Cap

Postcondició: S'ha obtingut tota la informació necessària per a informar al Gestor de Diagrames.

Descripció detallada: Durant el funcionament de l'aplicació a "analitzar", el fil d'execució ha arribat a un dels PointCuts especificats pel usuari. D'aquesta manera l'aspecte ha cridat al codi corresponent (advice) associat a aquest Pointcut, que obtindrà la informació necessària sobre el procés per tal de enviar-la al gestor de diagrames de seqüències. Aquest procés d'obtenció d'informació es el punt clau del projecte, sobretot el fet d'obtenir informació del procés inicial quan estem en un Pointcut situat a un objecte remot.

2.2.4 Cas d'ús número 4: "Informar al gestor de diagrames."

Resum de la funcionalitat: L'aspecte envia al gestor de diagrames la informació necessària sobre el context d'execució del Pointcut, per tal d'anar construint el diagrama de seqüències.

Paper dins el treball de l'usuari: L'aspecte el fa servir indirectament..

Actors: Cap, no es demana explícitament.

Casos d'ús relacionats: Obtenir informació del Procés.

Precondició: Cap

Postcondició: S'ha enviat al gestor de diagrames la informació necessària per a reflectir el context d'execució del Pointcut definit per l'usuari, en el diagrama de seqüències.

Descripció detallada: Una vegada obtinguda la informació necessària sobre el context d'execució del Pointcut definit per l'usuari, l'aspecte proporciona al gestor de diagrames tota aquesta informació per tal de que pugui crear el diagrama de seqüències.

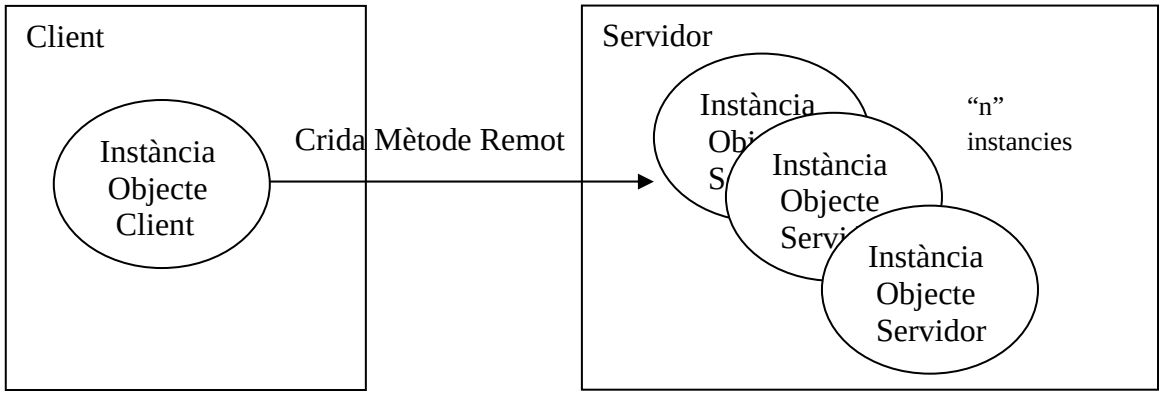
3 Disseny

3.1 Conceptes generals

3.1.1 Introducció

Com ja hem dit amb anterioritat, el nucli del Projecte és aconseguir enllaçar un fil d'execució iniciat en un client, i anar-ho seguint per tot el seu recorregut pels diferents servidors que pugui anar. Tot això sense perdre mai la seva identitat.

Cal comprendre que a un Objecte distribuït tindrem en un moment donat "n" fils d'execució. On cadascun haurà sigut creat per una crida a un mètode seu per part de un client.



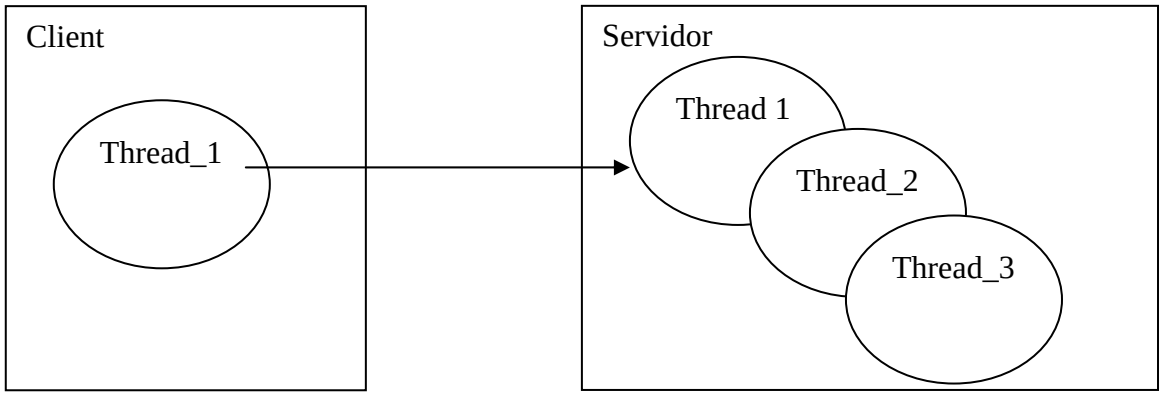
Si volem saber quina és la instància de l'objecte Servidor que pertany a la crida feta per la instància de l'objecte client, hem de crear alguna manera de poder emparellar-les.

La solució aplicada per a aconseguir aquesta fita ha estat identificar cada fil d'execució, des del seu començament, amb un identificador de sessió.
Després s'ha d'anar propagant aquest identificador pels diferents fils d'execució que es van creant en els objectes remots, per tal de tenir-ho sempre identificat.

D'aquesta manera al Gestor de Diagrames, que és una eina que aprofitarem d'un anterior PFC i que crea diagrames de seqüència, només cal que li diguem per a cada element que volem afegir, quin és el seu identificador de sessió. I afegirà aquest element a un diagrama de seqüència o a un altre, depenent d'aquest identificador.

Per a desenvolupar la feina de la variable que conté l'esmentat identificador de sessió s'ha emprat la propietat Name de l'objecte Thread. Que per les seves característiques es manté activa durant tota l'execució de cada fil.

Amb aquest nou identificador tindrem un escenari on tots els fils d'execució creats en un Objecte distribuït tindran sempre el mateix nom que el fil d'execució de l'objecte Client que l'ha creat:



En aquest cas veiem que el Thread_1 es troba tant al client com al servidor.
Els altres Threads (Thread_2 i Thread_3) hauran estat creats per altres clients, que a la seva vegada tindran el mateix nom al seu Thread.

3.1.2 Propagació del Nom del Thread

Per tal de propagar el nom del Thread necessitem quatre elements bàsics:

- El moment precís en que es crea el Thread de l'objecte Remot.

- La instància del Thread de l'objecte Client

- La instància del Thread de l'objecte Remot

- Un mecanisme per passar el nom del Thread del Client al Servidor.

Aquests quatre elements bàsics venen donats per unes necessitats:

- Hem de donar un nom al thread de l'objecte remot quan es crea, per tal de que qualsevol crida al gestor de diagrames ja vingui informada amb el nou nom.

- Necessitem la instància del Thread de l'objecte client per obtenir el nom i la de l'objecte Remot per a modificar-ho.

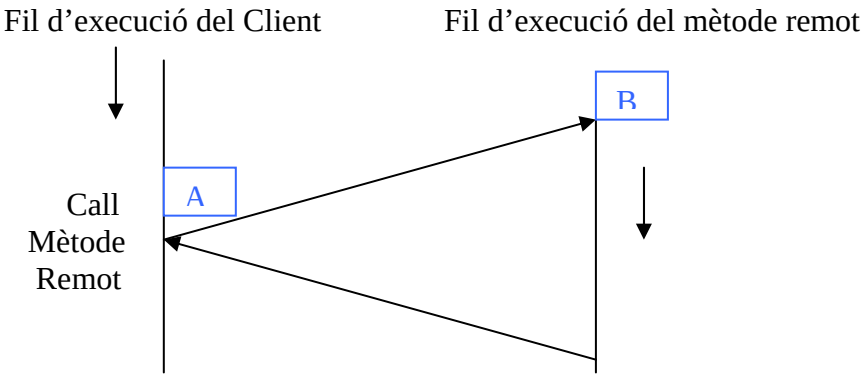
- I ens cal el mecanisme per a fer aquesta propagació del nom del Thread.

Per tal d'obtenir l'instant precís de creació del Thread de l'objecte Servidor creem un aspecte anomenat aspecteServidor.

Aquest aspecte te definit el pointcut `pcExecutaCallRemot` que representa les execucions de mètodes remots i un advice que s'executa en el before d'aquest pointcut. D'aquesta manera capturem l'instant just abans de executar-se la crida al mètode remot. En aquest advice tindrem la instància al Thread de l'objecte client i es el moment just en el que hem de canviar aquest nom del Thread.

Per tal d'obtenir la instància del Thread del client creem un altre aspecte anomenat aspecteClient .

Aquest aspecte te definit el pointcut `pcCridaCallRemot` que representa la crida a mètodes d'objectes remots i un advice que s'executa en el before d'aquest pointcut i que captura l'últim moment del fil d'execució del client abans de passar al objecte servidor. El concepte clau de la eina de propagació de nom del Thread rau en el fet que quan es fa el call de un mètode remot, immediatament després sabrem que comença l'execució.



En aquest dibuix podem veure exemplificats com el punt A i B els dos moments capturats per l'aspecte: el before de la crida remota i el before de l'execució. Aquests dos moments s'executaran sempre seguits , en el mateix ordre i amb un interval de temps entre ells molt reduït.

D'aquesta manera emprarem una eina en la que al instant A se li informa el nom del Thread del Client i a l'instant B recollim aquest nom.

Les característiques que ha de complir aquesta eina son:

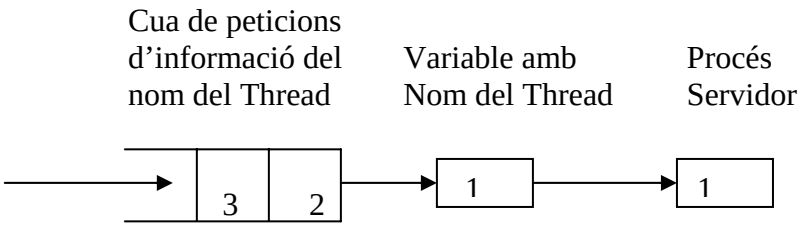
- Ha de ser distribuïda, per a que des de diferents JVM es pugui cridar.

- Ha de aconseguir que a cada punt B només s'agafi informació del punt A que li correspon.

Per aconseguir aquesta propagació sense que cap altre procés la contamini canviant el valor de la variable abans que el Servidor la consulti, s'ha creat una classe amb un sistema de semàfors.

Quan un client informa aquest valor, si arriben altres clients intentant propagar el seu nom del Thread, queden encuats fins que el Servidor el llegeix.
Com aquests clients resten esperant el seu torn, no es produirà cap inici d'execució d'un procés servidor, més que el produït per la crida que ha aconseguit informar el seu nom del Thread.

En aquest esquema representem el moment en que un procés de l'objecte remot llegeix el nom del Thread (1) que l'aspecte client ha deixat informat.
Mentre altres processos estan encuats esperant el seu torn (2 i 3)



Quan el procés servidor li canvia el nom al seu Thread amb aquest valor, envia un missatge a la cua informant que pot deixar passar al següent.

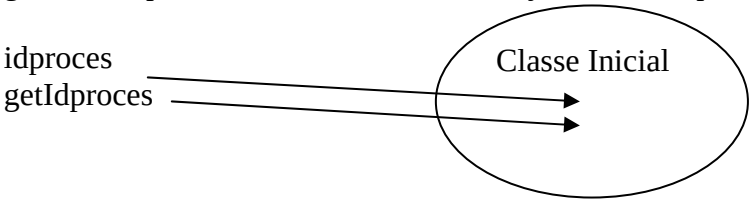
3.2 Arquitectures estudiades anteriorment

3.2.1 Aprofitament dels Objectes proporcionats per AspectJ.

La primera línia d'investigació va ser aprofitar els objectes proporcionats per l'AspectJ. AspectJ proporciona accés a informació tan estàtica com dinàmica associada als joinpoints.
Emprant aquestes eines podem obtenir referències als objectes on es troben els pointcuts, els arguments de les crides, etc.
Es va aprofundir molt en el tema i l'únic que es va aconseguir és arribar a saber l'adreça IP dels clients. Aquesta informació pot ser molt útil per a aplicacions web, però en el nostre projecte no ens podíem conformar arribar fins a aquest nivell de coneixement. Ja que pot donar-se el cas que diferents processos s'executin al mateix ordinador i no els diferenciarien.

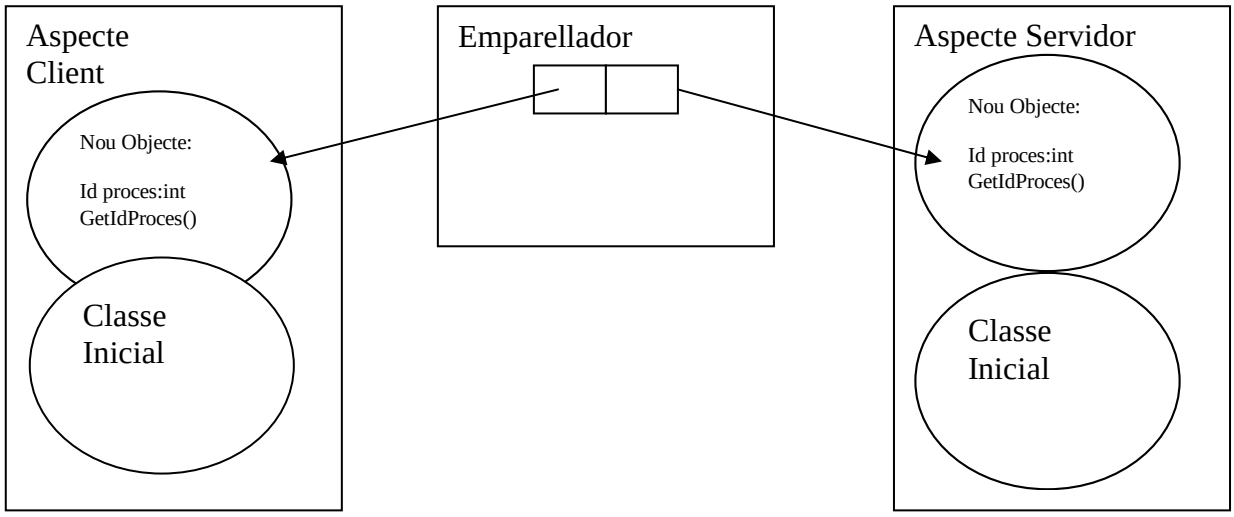
3.2.2 Afegir idproces a les classes

L'AspectJ ens permet afegir mètodes i membres a les classes de java.
La idea era introduir a cada classe una variable idproces amb un valor identificatiu d'aquest procés. Que aniríem propagant pels objectes remots emprant la propietat del AspectJ que ens permet tenir referències dels objectes.
No obstant ens vam trobar amb una limitació. Per introduir nous atributs i mètodes als objectes ens calia conèixer el seu nom a priori, i si volem fer una aplicació de caire general no podem saber els noms dels objectes en temps de programació.



3.2.3 Ús de Callbacks.

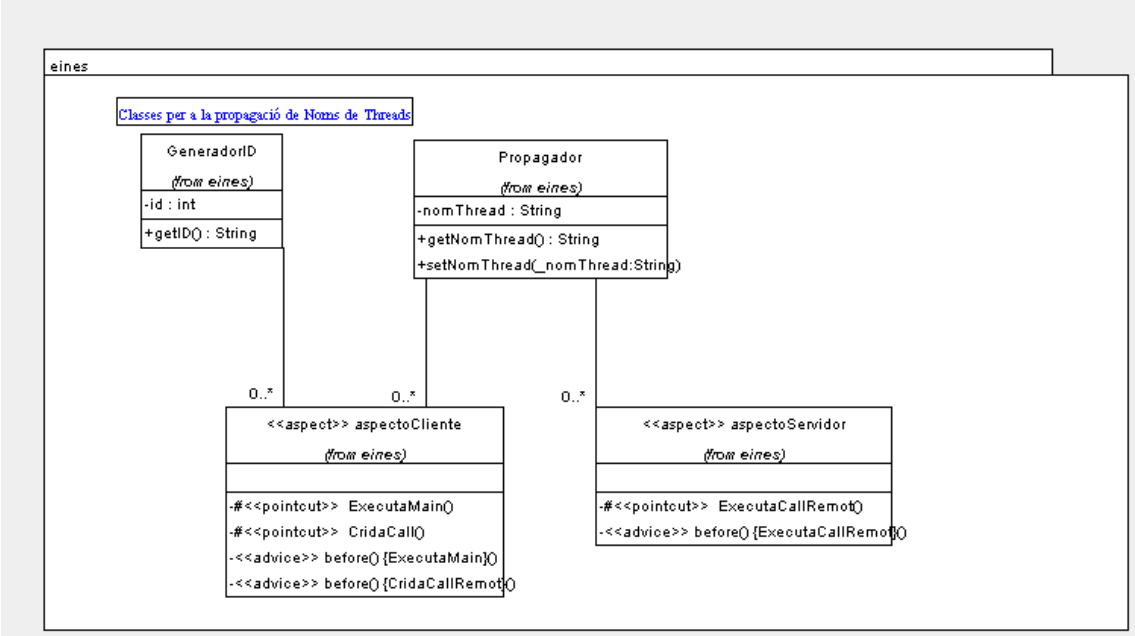
Si no podíem introduir atributs a un objecte del qual no coneixem el nom, almenys podríem crear els nostres propis objectes que associaríem amb els de l'aplicació.
D'aquesta manera, un aspecte associat al objecte crearia una instància d'un objecte nostre que només tindria l'atribut idproces i els seus mètodes de lectura i escriptura.
Així cada fil d'execució tindria el seu Objecte identificador associat.



El objecte emparellador s'encarrega de tenir instàncies de cadascú dels objectes associats, així podem seguir el fil d'execució del client al servidor.

Amb aquesta arquitectura possiblement s'hagués pogut arribar a fer l'eina, però a cada pas que donaven es complicava una mica més.
Primer de tot introduïa massa operacions dins l'execució, el fet de fer CALLBACKS als objectes per obtenir els identificadors era molt costos.
Altre problema que tenien era a per obtenir després el fil d'execució. S'havia de crear un sistema que anés emparellant les parelles del emparelladors. Encara que sembla un joc de paraules, nosaltres només tenim parelles d'objectes, i potser que al llarg de l'execució tinguem "n" parelles que cal unir.
L'aparició de la possibilitat d'emprar el nom de Thread i la seva simplicitat van ser detonants de la decisió d'abandonar aquesta línia de treball.

3.3 Arquitectura de la part de propagació del nom del Thread

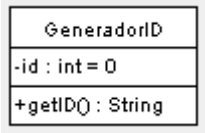


4. Implementació

4.1 Classe GeneradorID

Aquesta classe és la encarregada de proporcionar noms diferents que identifiquen a cada Thread.

Aquesta classe s'ha fet remota per tal que qualsevol procés, independentment de la màquina en la que s'estigui executant, pugui obtenir un ID que el permeti identificar respecte els altres processos que s'estan executant al llarg de tota la nostra aplicació distribuïda.



Com podem veure consta d'un únic atribut i un mètode.

L'atribut id és un enter privat que al construir la classe té per defecte el valor 0.

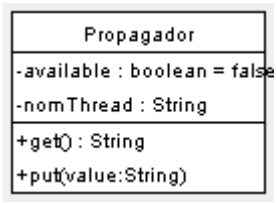
El mètode getID() ens retorna un String que emprarem com a nom del nostre Thread, a mode de identificador. Aquest String estarà format per la concatenació del literal "`Thread_`" més el nombre que tinguem en aquell moment al ID.

Cada vegada que demanem un getID() s'anirà incrementant la variable id en una unitat.

El mètode getID() està definit com synchronized per evitar que hi hagi més d'un procés alhora demanant aquest ID i que puguin sorgir problemes com ara Ids repetits.

4.2 Classe Propagador

La classe propagador és la encarregada de permetre la propagació del nom del Thread entre Client i Servidor.



Aquesta classe consta de 2 atributs privats i 2 mètodes:

- Atribut nomThread: Aquest atribut és on la classe guarda el nom del Thread que va a propagat.

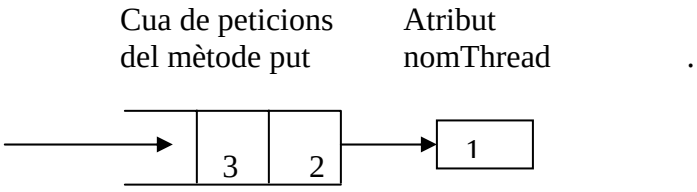
- Atribut available: Aquest atribut private és un boolean que ens marca si la propera acció que espera realitzar la classe és un get o un put.
  Si available = false vol dir que està esperant que l'atribut nomThread sigui informat.
  Si available = true vol dir que l'atribut nomThread està informat i llest per a ser consultat.

- Mètode put(value:String): Aquest mètode s'utilitza per a informar el valor del nomThread amb el paràmetre introduït. Una peculiaritat d'aquest mètode només deixa informar al primer que el crida. Els altres resten bloquejats fins que una crida al mètode get() desbloqueja el següent a mena de cua.

- Mètode get(): Aquest mètode retorna el valor de l'atribut nomThread i desbloqueja als processos encuats al put.

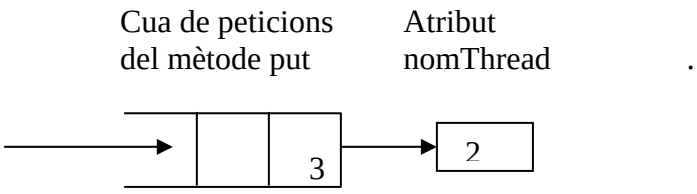
Anem a veure una exemplificació del seu funcionament:

Els processos amb nomThread 1, 2 i 3 han fet una crida al mètode put.
No obstant només ha aconseguit informar el nomThread el 1. Els 2 i el 3 resten encuats



Quan un altre procés fa un get, llegeix el valor del nomThread = 1 i desbloqueja els altres processos.

Passarà el procés amb nom del thread = 2 i el 3 restarà encuat, fins al pròxim get.



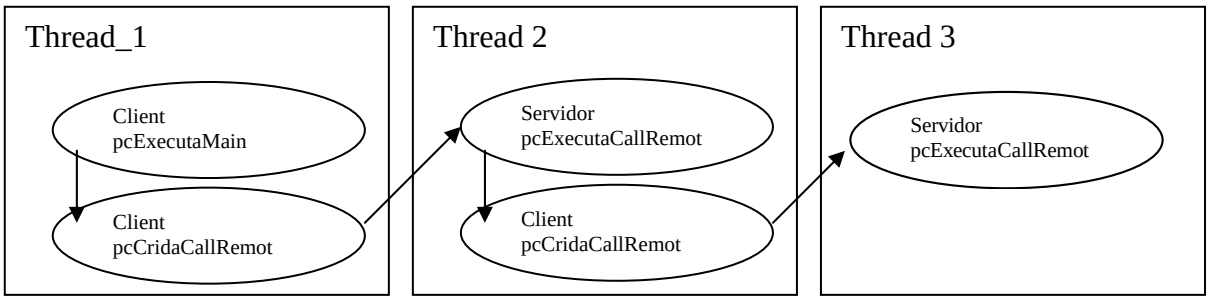
4.3 Aspectes

Per a obtenir una eina de propagació de nom del Thread, s'han hagut de crear 2 aspectes diferents.
Aquests aspectes son: el aspecteClient i el aspecteServidor.

Hem fet aquesta separació, perquè a l'aspecteClient tenim definits uns pointcuts que s'activaran en el Thread del Client, i l'aspecteServidor en té els que s'activaran al Servidor.

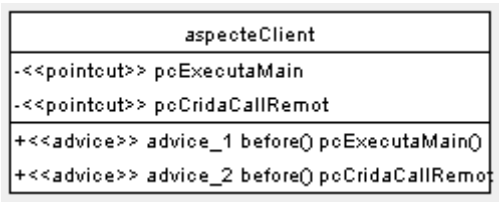
No obstant, cal mencionar, que els conceptes de Client i Servidor són relatius.
Un Thread pot haver començat en un Objecte remot com Servidor, però si després crida a un mètode intern ho fa com a client. Fins i tot pot cridar a un altre mètode remot i també ho farà com client.

Anem a veure una petita exemplificació dels pointcuts que s'activaran, i que explicarem en detall en següents apartats.



4.3.1 aspecteClient

L'aspecteClient té el següent format:



Es defineixen dos pointcuts:

-ExecutaMain: Aquest pointcut captura l'instant inicial de creació d'un procés. En el nostre cas, adaptat a RMI , captura la execució de la funció main, que és com comencen els processos. Per a altres entorns caldrà definir el punt inicial.

```
execution(* *.main(..))
```

-pcCridaCallRemot: Aquest pointcut captura l'instant en que es fa una crida a un objecte remot. Aprofitem el fet que els objectes remots han de controlar la RemoteException per a identificar-los.

```
call(public * *.*(..)throws RemoteException)
```

A més de controlar aquestes joinpoints, els nostres pointcuts, de tots els aspectes, controlen que no siguin capturades crides a llibreries de java o pròpies de la nostre eina. No obstant no tenen molta importància de cara al nostre estudi, per aquesta raó, introduir aquests controls a la memòria no afegeix més que una saturació d'informació.

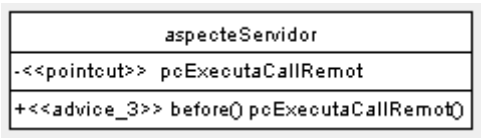
Es defineixen dos advices que es dispararan en el before de cada pointcut:

-before()ExecutaMain: Així ens assegurem que el codi d'aquest advice s'executa abans que res del procés. Ja que s'activa en el before de la seva execució.
En aquest advice executem un codi que demana un nom per a identificar el Thread al objecte remot generadorID i actualitzem el nom del Thread amb aquest identificador.

-before()CridaCallRemot: Aquest advice s'executa abans de fer la crida a un objecte Remot, d'aquesta manera emprem el Objecte Remot Propagador i li informen el nostre nom del Thread per a que després el pugui agafar el Servidor.

4.3.2 Aspecte Servidor

L'aspecteServidor té el següent format:



Es defineix un pointcuts:

-pcExecutaCallRemot: Aquest pointcut captura l'instant en que s'executa un objecte remot. Aprofitem el fet que els objectes remots han de controlar la RemoteException per a identificar-los.

```
execute(public * *.*(..)throws RemoteException)
```

Es defineixen un advice que es dispararà en el before d'aquest pointcut:

-before()ExecutaCallRemot:
Cada vegada que un objecte remot es instànciat es crea un Thread diferent. Així que tindrem un per a cada crida al objecte remot.
L'instant inicial del Thread és quan s'executa un mètode definit com remot, i amb aquest advice capturem l'instant inicial de l'execució.
En aquest advice executem un codi que demana un nom per a identificar el Thread al objecte remot Propagador i actualitzem el nom del Thread amb aquest identificador.
Donada la implementació de l'objecte Propagador i el fet que tots els Threads menys un són encuats, sempre tindrem la certesa que el valor del nom del Thread que consultem a l'execució d'un objecte remot coincideix amb el informat pel client al fer el call del mateix mètode.
D'aquesta manera, el nou Thread creat al Objecte Servidor tindrà el mateix nom que el Thread que ha fet la crida. Així hem aconseguit propagar el nom del Thread.

4.4 Integració amb el projecte de Josep Lluís Lérida Monsò

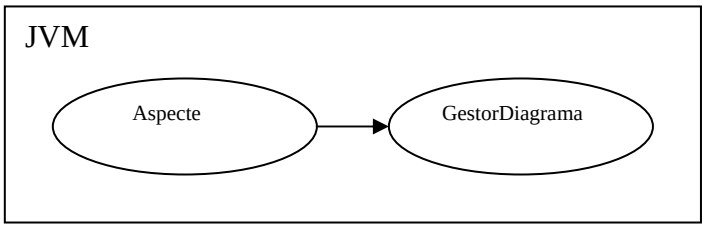
Un anterior alumne de Enginyeria Informàtica anomenat Josep Lluís Lérida Monsò, va fer un projecte Final de carrera sobre "L'Ús de la programació Orientada a Aspectes en tasques de Reenginyeria" en el qual va crear una aplicació que realitzava un diagrama de seqüències d'una aplicació monolítica. I va deixar oberta a la possibilitat de estendre aquesta funcionalitat a aplicacions distribuïdes.

Del seu projecte es van aprofitar un conjunt de classes per a realitzar diagrames de seqüència.
No obstant aquestes classes han hagut de ser modificades per a integrar-se en el nostre projecte.

A continuació veurem els canvis realitzats al PFC del Josep Lluís Lérida Monsò per a integrar-se al nostre projecte.

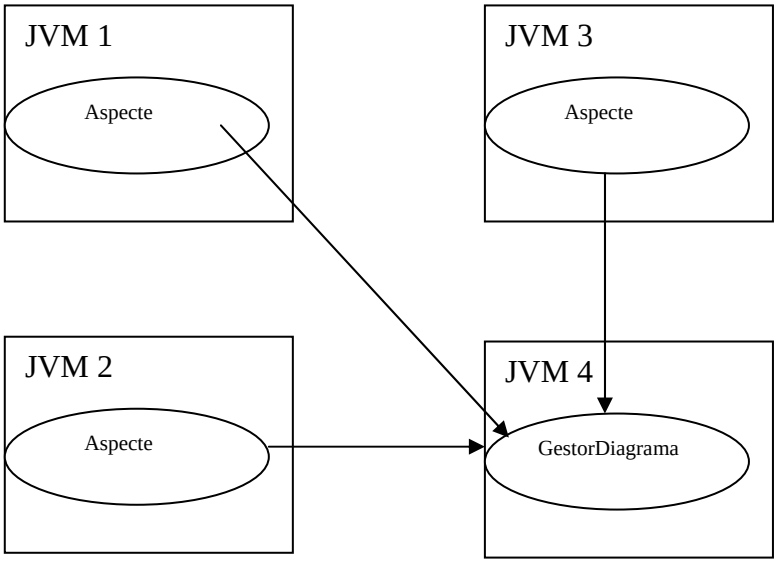
4.4.1 Modificació de la classe GestorDiagrama per a que sigui remota.

En el projecte anterior els aspectes que proporcionaven la informació a aquest Gestor de Diagrames estaven a la mateixa Java Virtual Machine (JVM) que la classe GestorDiagrama.



Ara aquests aspectes proporcionen informació des de JVMs diferents i també des de ordinadors diferents.

Per a que sigui visible des de qualsevol lloc, la classe GestorDiagrama ha de ser remota.



4.4.2 Parametrització de les funcions del GestorDiagrama:

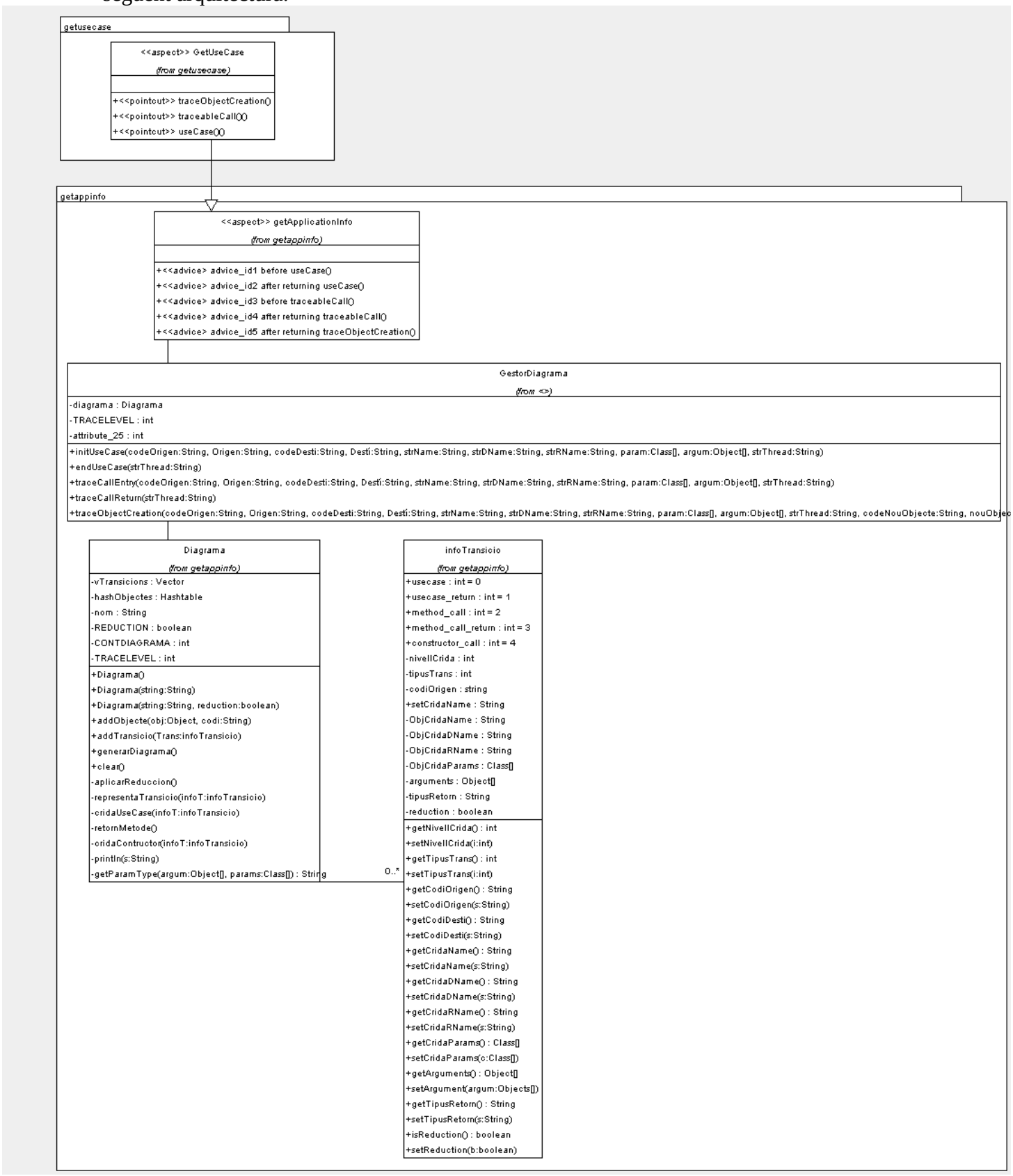
Hi havia a la classe GestorDiagrama paràmetres als seus mètodes que no complien la condició de ser serializables.
Per exemple, es passaven com a paràmetres Objectes sencers que mai sabíem de quin tipus eren, al ser cada vegada un objecte diferent (el que el pointcut agafes).
Moltes vegades resultava que aquest objectes no eren serializables.
Per aquesta raó no es podien fer crides remotes emprant aquesta Parametrització.
Però estudiant en detall el funcionament de les classes es va veure que d'aquests paràmetres complexos, només es treballava amb un conjunt d'atributs simples i bastant reduït. D'aquesta manera en lloc de parametritzar tot l'objecte es va optar per descompondre'l en el conjunt d'atributs senzills que realment s'utilitzaven i que ja eren serializables.
Aquests paràmetres solien ser noms d'objectes i mètodes i atributs per l'estil.

4.4.3 Modificació dels Aspectes

L'Aspecte abstracte GetUseCase que defineix l'usuari, ha quedat igual a com estava.
No obstant a l'Aspecte getApplicationInfo, hem hagut de canviar els advices.
En aquests advices es cridava a mètodes de GestorDiagrama amb els antics paràmetres no serializables. El que s'ha fet és descompondre a l'advice aquests paràmetres en Objectes més senzills que si són serializables i que són els que ara passem als mètodes.

4.4.4 Arquitectura final

Una vegada modificades les classes per a crear diagrames de seqüència, ens queda la següent arquitectura:



5 Manual D'ús

5.1 En línia Comandes Ms-Dos

Per a instrumentalitzar una aplicació amb la nostra eina, es pot emprar infinitat de productes.
Per tal de que tothom el pugui fer servir, emprarem la forma més directa, encara que no pas la millor.
Emprarem la línia de comandes de Ms-Dos.

Necessitarem disposar de la versió 1.4.2 del JDK o superior, que la podem trobar a:

-http://java.sun.com/j2se/1.4.2/downloadl.html

A més cal tenir les funcionalitats del AspectJ, i el seu compilador ajc. Que podem trobar a:

-http://www.eclipse.org/aspectj/index.html

I per passar els nostres fitxers en format text a diagrames de seqüència, necessitem disposar de la eina sequence, que podem trobar a:

-http://www.zanthan.com/itymbi/archives/cat_sequence.html

Una vegada tenim tot això, s'han de seguir els següents passos.

Si no tenim el aspectjjrt.jar o el tools.jar accessibles per al nostre compilador de java, posarem al classpath el nom absolut de cadascun dels 2 fitxers.
El tools.jar ens servirà per a les crides rmi que fa servir l'eina i el aspectjrt.jar per a que suporti AspectJ.

Crearem un directori de treball (per exemple c:\DirTreball), i copiarem l'aplicació a instrumentalitzar.

Dins d'aquest directori descomprimirem el fitxer eines.zip, que conté la nostra eina per a instrumentalitzar aplicacions.

Si volem modificar el GetUseCase per Apropar-lo a les nostres necessitats, només hem de re-escriure el GetUseCase.java.

Per a compilar tot junt, i que els Aspectes es “teixin” amb l’aplicació, ho hem de fer amb el compilador de AspectJ.

C:\DirTreball>ajc eines/*.jar *.java

En aquesta línia li diem al compilador de AspectJ (ajc) que volem compilar el que hi ha dins del directori eines acabat en java (*.java) amb tot el que hi ha al DirTreball que te la terminació java (*.java).

En cas de tenir subdirectoris (Subdirectori1,Subdirectori2) poden fer:

C:\DirTreball>ajc -aspectpath eines.jar .java Subdirectori1/.*java Subdirectori2/*.java

Amb aquesta instrucció ja hem compilat l’aplicació. No cal fer javac ni emprar cap altre compilador de Java.

Com normalment treballarem amb aplicacions distribuïdes a més de compilar les classes, haurem de compilar les classes remotes amb el compilador del rmi.

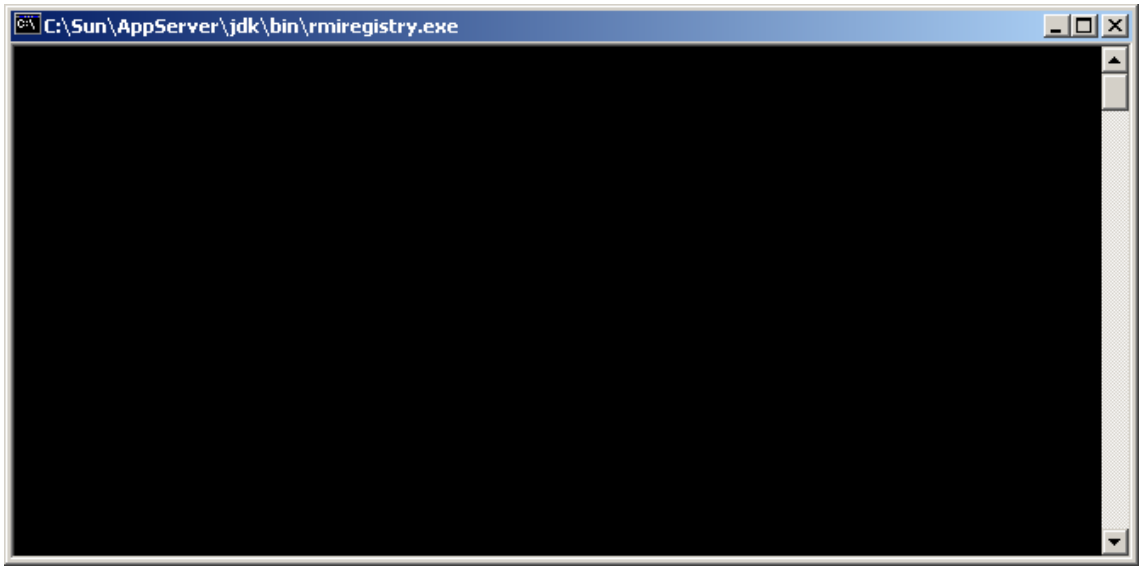
Per exemple si tenim dues classes remotes anomenades CreditManagerImpl i CreditCardImpl, farem:

rmic CreditManagerImpl

rmic CreditCardImpl

Després engegarem el registre rmi per a poder registrar les classes remotes:

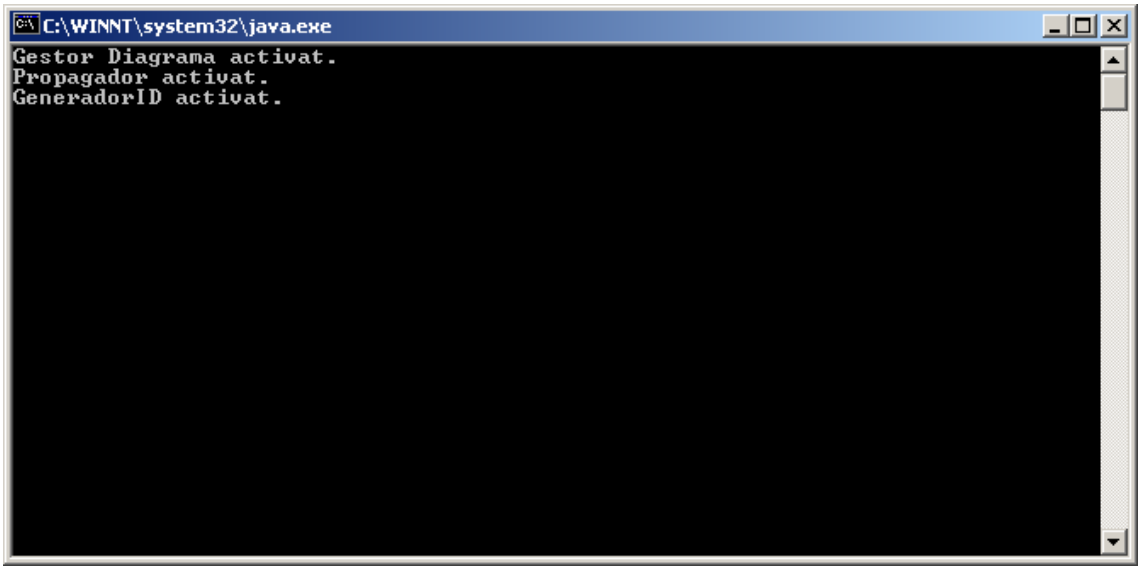
start rmiregistry



Sempre haurem de engegar el rmiregistry, encara que l’aplicació no sigui RMI, ja que la eina d’instrumentalització el fa servir.
Ara cal engegar les classes que publicaran els objectes remots:

Primer engeguem la de la nostra eina instrumentalitzadora:

start java eines/activaEines



Una vegada arribat a aquest punt engeguem l'aplicació a instrumentalitzar.

Engeguem el servidor. Si un cas sigues la classe CardBank, faríem:

start java CardBank

I finalment, engegaríem el client. Que si s'anomenés Shopper, faríem:

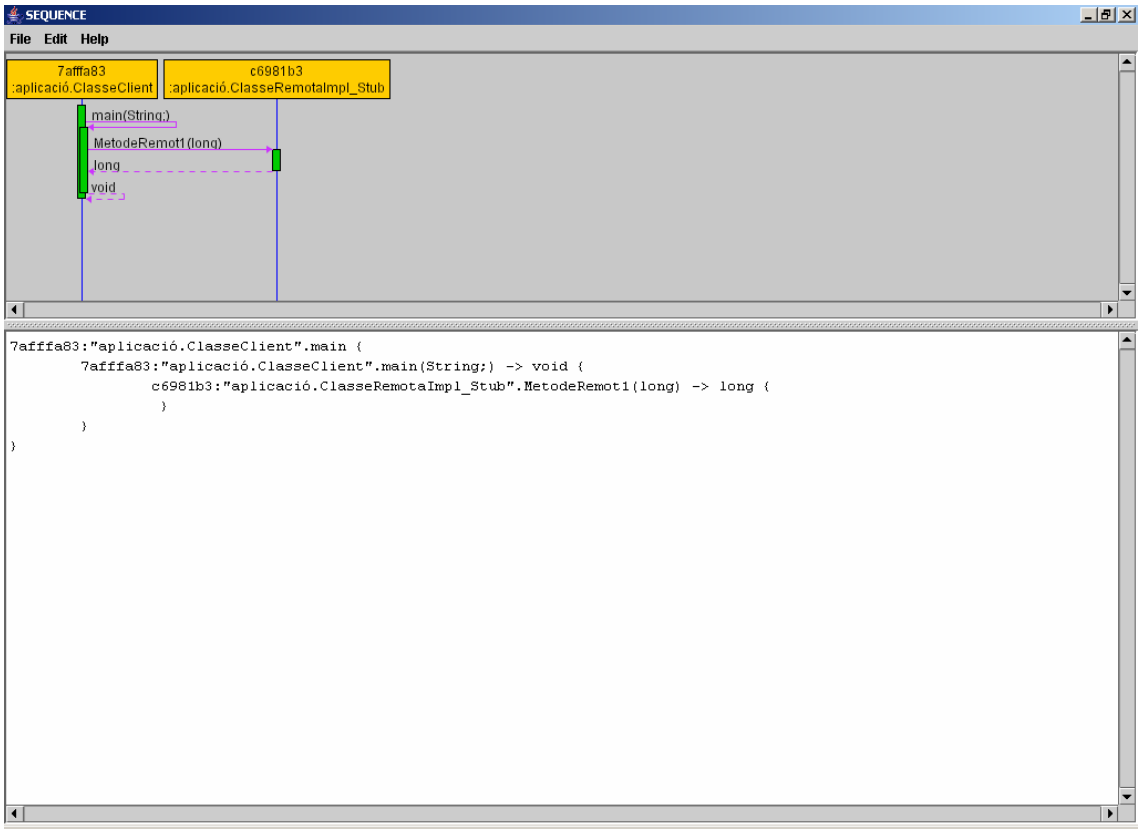
Java Shopper.

Al final de la execució obtindrem al nostre DirTreball un fitxers de format text amb un nom del tipus main_Thread_x_y.

Cadascú d'aquest fitxers, representa un cas d'ús.

Si engeguem el Sequence i introduïm un d'aquests fitxers obtindrem el diagrama de seqüències.

Per a executar el sequence l'hem de cridar amb la línia de comandes: java –jar sequence.jar-


SEQUENCE
File Edit Help
7afffa83
:aplicació.ClasseClient
c6981b3
:aplicació.ClasseRemotaImpl_Stub
main(String;)
MetodeRemot1(long)
long
void
7afffa83:"aplicació.ClasseClient".main {
7afffa83:"aplicació.ClasseClient".main(String;) -> void {
c6981b3:"aplicació.ClasseRemotaImpl_Stub".MetodeRemot1(long) -> long {
}
}
}

5.2 En eclipse

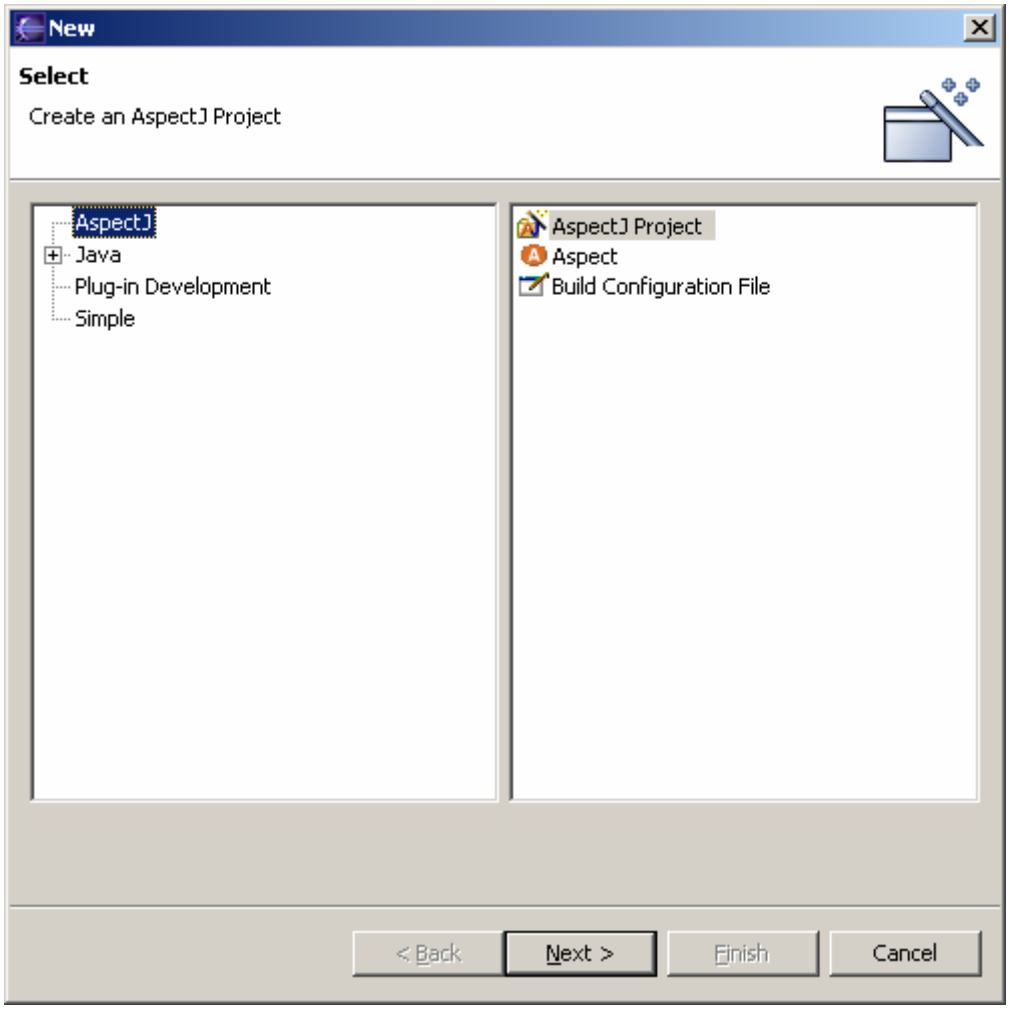
Si volem emprar la eina en eclipse haurem de preparar l'entorn per treballar amb RMI i AspectJ.

Per a preparar l'eclipse per a que suporti AspectJ , ens haurem de baixar e instal·lar, el puglin de AspectJ de :

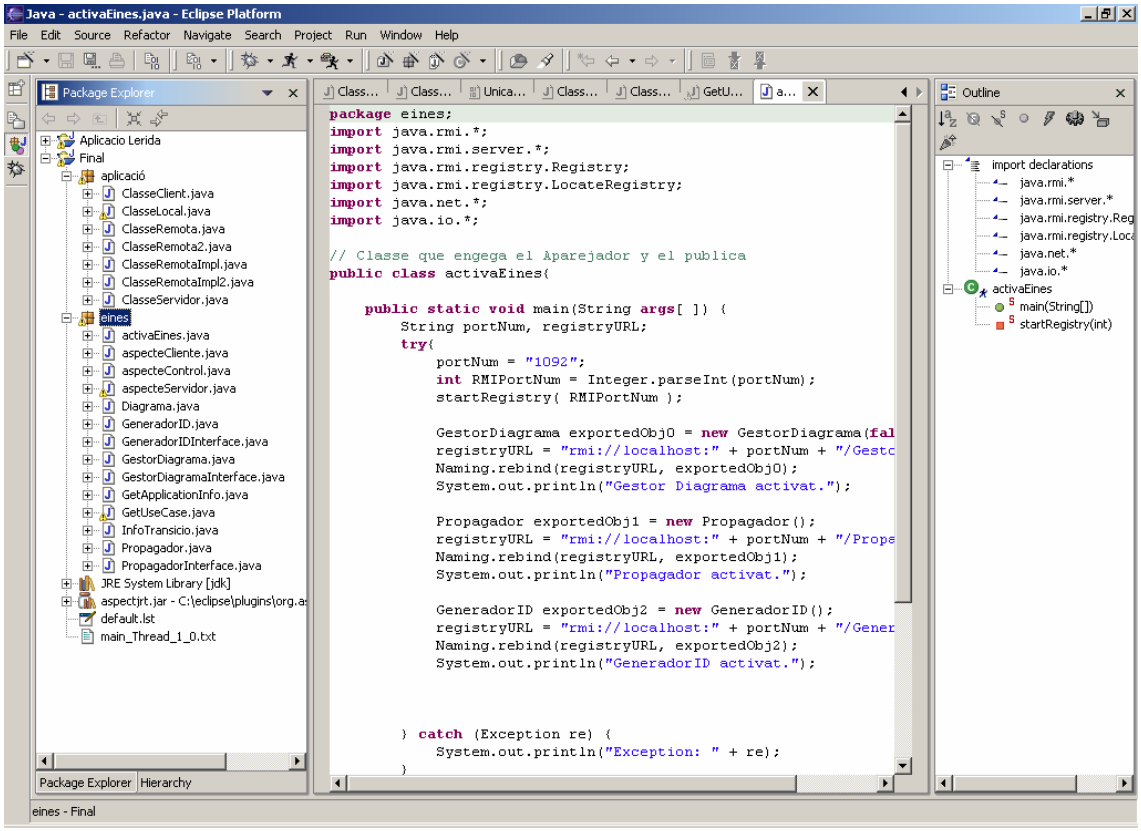
. http://www.eclipse.org/downloads/index.php

I el plugin de Rmi:

Una vegada tenim tots el plugins instal·lats s'ha de crear un nou projecte que suporti AspectJ:



Després hem de importar al nostre projecte el paquet eines.



Ens hem d'assegurar que el nostre Projecte suporta RMI, seleccionant el projecte, desplegant les propietats d'aquest amb el botó dret del ratolí i seleccionant:
RMI -> Add RMI support

Una vegada tenim això cal compilar el projecte.

Després executarem activaEines i ja estem preparats per a executar el projecte amb normalitat.

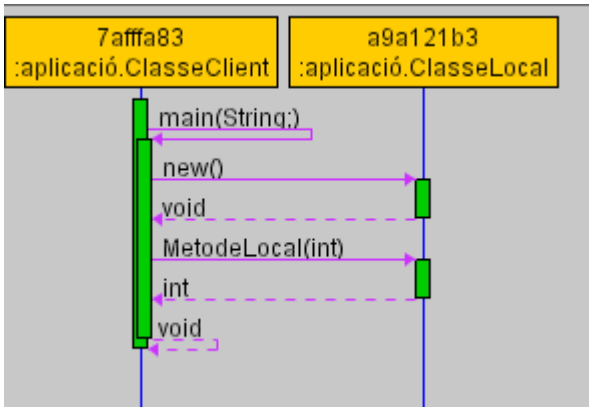
Al directori d'aplicacions del eclipse ens apareixeran els fitxers creats amb els diagrames d'ús.

6 Proves

Anem a veure un seguit de proves d'instrumentalització amb la nostra eina.
Hem fet servir una aplicació amb uns noms de classe i mètodes força entenedors, per a veure el diagrama de seqüències més clarament.
D'aquesta manera una classe Local l'anomenarem ClasseLocal, una de remota ClasseRemota. I de forma anàloga els mètodes rebran noms com ara MetodeLocal.

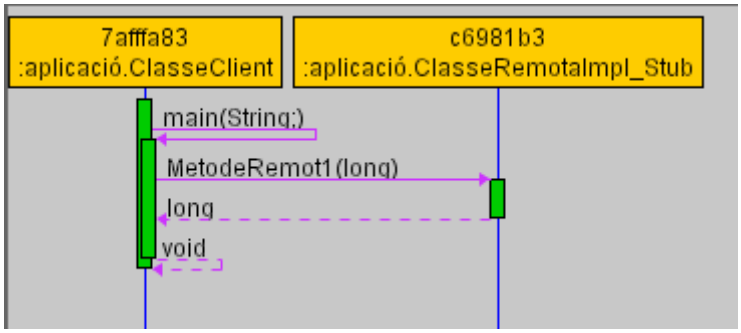
6.1 Creació Classe Local i crida a mètode d'aquesta:

En aquest cas disposem d'una classe Client que crea una Classe Local i en crida un dels seus mètodes anomenat metodeLocal.



6.2 Crida a mètode de classe remota 1:

En aquest cas la Classe client crida a un mètode (metodeRemot1) que retorna long, d'una classe remota (ClasseRemota).

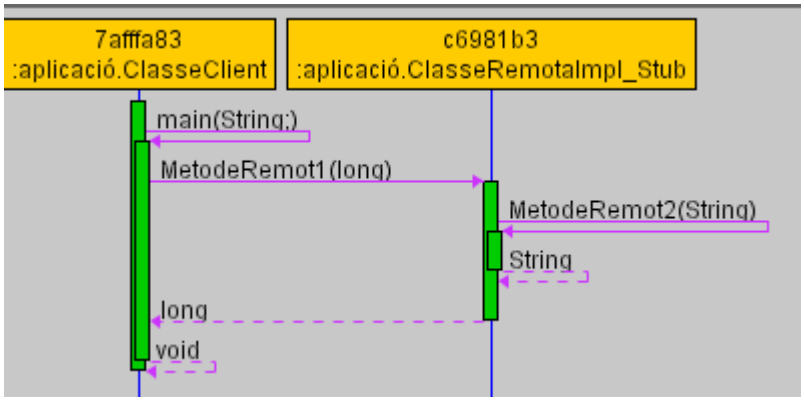


Hi ha dos canvis evidents respecte al exemple anterior:

a) La Classe Remota no es crea. Ja ha sigut creada per una altre classe anomenada ClasseServidor, però que no forma part d'aquests cas d'ús (més endavant veurem el cas d'us d'aquesta classe).
b) No cridem directament a la classeRemota, sinó a la ClasseRemotaImpl_Stub , això es degut a que el RMI crea aquesta classe intermitja per a la comunicació.

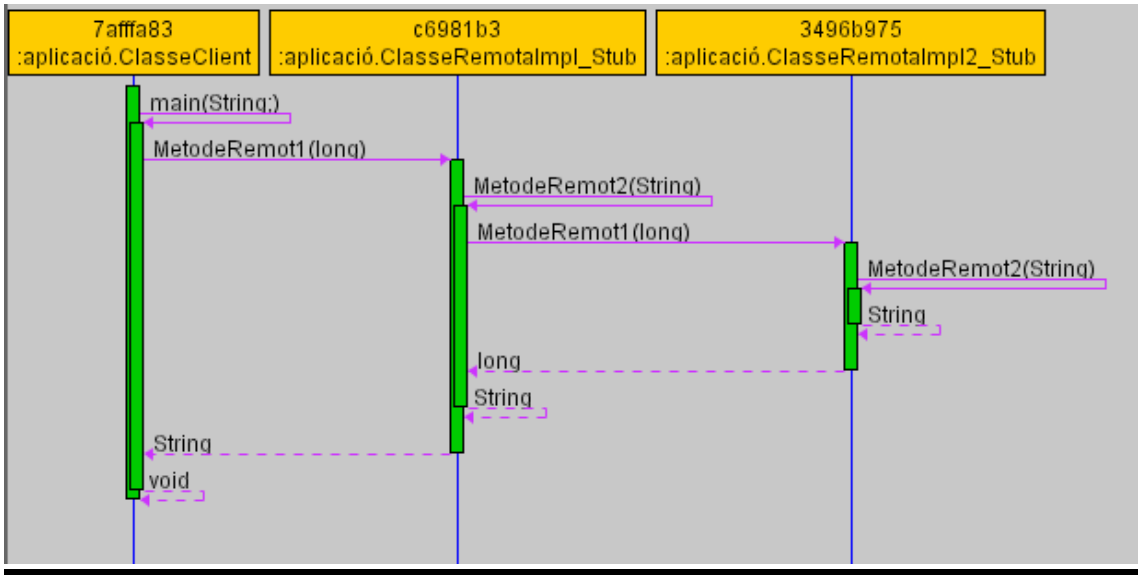
6.3 Crida a mètode de classe remota 2

Per complicar-ho una mica més, anem a veure que diagrama d'ús es crea en el cas que aquest metodeRemot1 cridi a un altre mètode de la classeRemota anomenat metodeRemot2 que retorna un String.
Encara que aquest mètode es remot, al cridar-se des de la classe mateixa es comporta com local.



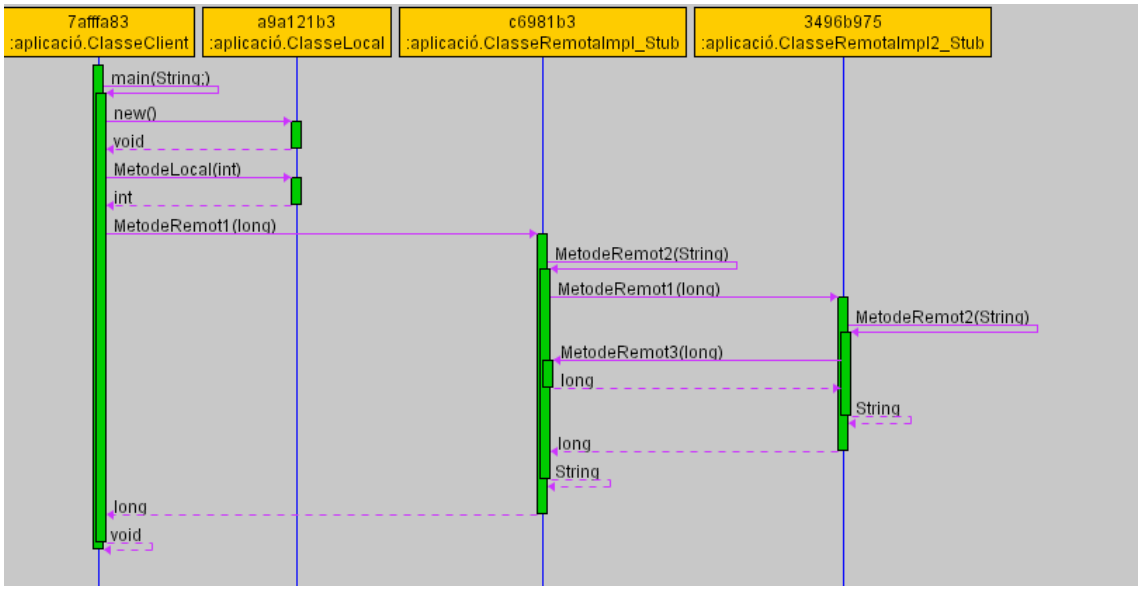
6.4 Crida a mètode de classe remota 3

Complicant, encara una mica més la situació, anem a veure que passa si aquest MetodeRemot2 crida a un altre mètode remot anomenat MetodeRemot1 d'una altre classe remota anomenada ClasseRemota2, que a la seva vegada crida a un mètode propi anomenat MetodeRemot2.



6.5 Crida a mètode de classe remota 4

Per acabar amb aquesta sèrie d'exemples farem que el MetodeRemot2 de la ClasseRemota2 cridi a un nou MetodeRemot3 de la ClasseRemota.

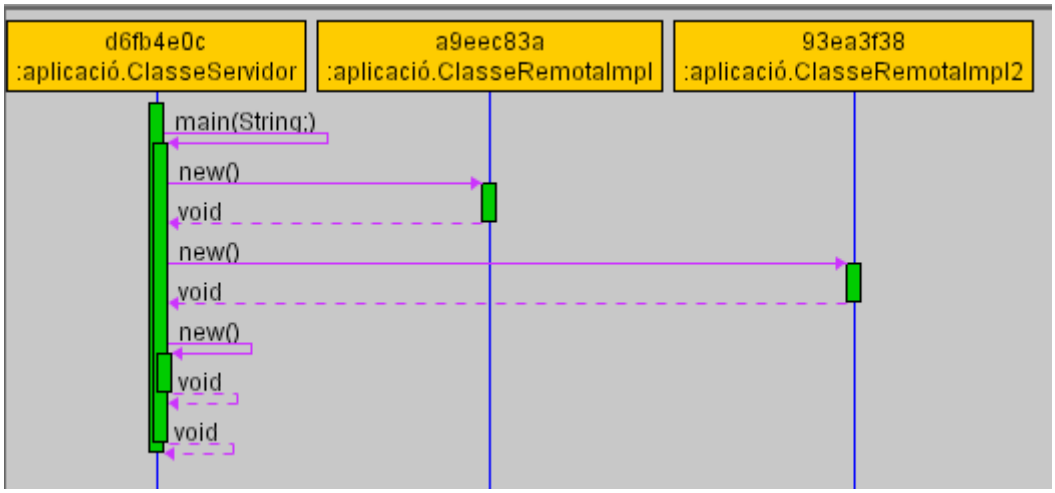


6.6 Exemple Complet

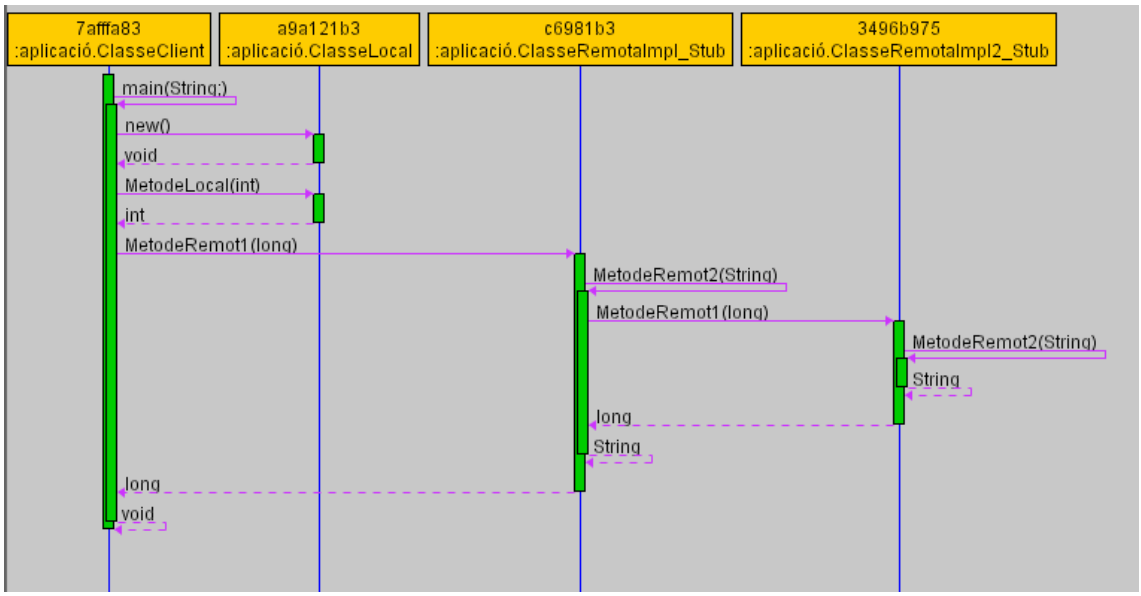
A continuació veurem un exemple complert que ens dona una vista més general de les possibilitats de la nostra eina.
En aquest exemple veurem dos casos d'ús:

a) El cas d'ús de la ClasseServidor, que és la classe que crea i publica les classes remotes.
b) El cas d'ús de la ClasseClient, que és una classe que realitzarà tant crides locals com remotes.

Anem a veure el cas d'ús de la ClasseServidor:



I ara el de la ClasseClient:



6.7 Exemple Real

Per tal de veure que no només funciona amb exemples creats expressament per a el PFC, anirem a instrumentalitzar una aplicació distribuïda que hem estret de les aplicacions de prova d'un altre PFC.

Es tracta d'una simulació de compra amb targeta de crèdit on cridem als mètodes remots del banc, que en aquest cas són CreditManager i CreditCard.

7 Planificació

7.1 Itinerari previst

L'itinerari previst per al PFC ha sigut:

- Estudi i familiarització amb AOP:
  - Estudi AOP
  - Instal·lació de programari
  - Prova d'exemples senzills
  - Familiarització amb codi de J.L.Lerida
  - Implementació exemple de instrumentalització
- Estudi de RMI:
  - Estudi informació disponible en crida RMI
  - Estudi de com aparellar la crida a un RMI-stub (client) amb el RMI-Remot
- AOP i RMI:
  - Realització exemple d'instrumentalització en aplicació distribuïda.
  - Estudi de la millor estratègia per a la gestió de logs.
- Capturar emparellaments amb AOP:
  - Estudi informació necessària per aparellament.
  - Generació d'aspectes que envien la informació al mòdul central de logs.
  - Creació del mòdul central d'emparellaments.
- Generació de diagrames de Seqüència:
  - Incorporació dels Aspectes que capturen informació sobre les crides.
  - Incorporació del Gestor de Diagrama del PFC-Lérida.
  - Unir diferents fils d'execució equivalents.
- Estudi en altres entorns distribuïts.
- Redacció de la memòria del projecte
- Creació de la presentació virtual

7.2 Fites

Les fites que teníem assignades eren:

- **Inici del Projecte:**
  Moment inicial del projecte.

- **Programari instal·lat:**
  Tenim el programari necessari instal·lat.

- **Exemple senzill d'instrumentalització:**
  Una vegada fet l'estudi i familiarització amb l'AOP hem aplicat els coneixements adquirits implementant un exemple senzill d'instrumentalització que genera un log de la aplicació.

- **Exemple senzill d'instrumentalització en aplicació distribuïda:**
  Una vegada acabats l'estudi de RMI apliquem els coneixements adquirits creant un exemple senzill d'instrumentalització que genera un log de la aplicació distribuïda.

- **Escollida la millor estratègia per la gestió de logs:**
  En aquest punt hem escollida la millor estratègia per a la gestió de logs (centralitzada o distribuïda).

- **Aplicació que escriu un log d'emparellaments a un fitxer**
  Al finalitzar l'estudi d'emparellaments obtindrem una aplicació que escriu un log d'emparellaments a un fitxer de diferents fils d'execució amb crides remotes.

- **Aplicació que crea diagrames d'ús per aplicacions distribuïdes:**
  Després d'incorporar al projecte els aspectes i classes del PFC-Lérida per tal de crear diagrames de seqüència i aconseguir emparellar els diferents fils d'execució, implementarem aquesta aplicació que crea diagrames d'ús per aplicacions distribuïdes

- **Final de projecte:**
  En aquest moment hem acabat de redactar la memòria i creat la presentació virtual.
  El projecte ja es pot lliurar.

7.3 Calendari de treball inicial

En un principi es va crear aquest calendari de treball, que com veurem posteriorment va variar substancialment a partir de la tasca d'estudi dels emparellaments.

Id	Nom Tasca	Duració	Data Inici	Data Final	Predecessores
1	Inici del Projecte	0 dies	lu 15/03/04	lu 15/03/04	
2	**Estudi i familiarització amb AOP**	14 dies	lu 15/03/04	do 28/03/04	1
3	Estudi AOP	14 dies	lu 15/03/04	do 28/03/04	
4	Instal·lació de programari	3 dies	lu 15/03/04	mi 17/03/04	
5	Programari instal·lat	0 dies	mi 17/03/04	mi 17/03/04	4
6	Prova d'exemples senzills	4 dies	ju 18/03/04	do 21/03/04	5
7	Familiarització amb codi de J.L.Lerida	11 dies	ju 18/03/04	do 28/03/04	4
8	Implementació exemple de instrumentalització	7 dies	lu 22/03/04	do 28/03/04	6
9	Exemple senzill d'instrumentalització	0 dies	do 28/03/04	do 28/03/04	8
10	**Estudi de RMI**	21 dies	lu 15/03/04	do 04/04/04	1
11	Estudi informació disponible en crida RMI	7 dies	lu 15/03/04	do 21/03/04	
12	Estudi de com aparellar la crida a un RMI-stub (client) amb el RMI-Remot	14 dies	lu 22/03/04	do 04/04/04	11
13	**AOP i RMI**	14 dies	lu 29/03/04	do 11/04/04	2
14	Realització exemple d'instrumentalització en aplicació distribuïda	7 dies	lu 29/03/04	do 04/04/04	9
15	Exemple senzill d'instrumentalització en aplicació distribuïda	0 dies	do 04/04/04	do 04/04/04	14
16	Estudi de la millor estratègia per a la gestió de logs	7 dies	lu 05/04/04	do 11/04/04	14;15
17	Escollida la millor estratègia per la gestió de logs	0 dies	do 11/04/04	do 11/04/04	16
18	**Capturar emparellaments amb AOP**	21 dies	lu 12/04/04	do 02/05/04	13
19	Estudi informació necessària per aparellament	7 dies	lu 12/04/04	do 18/04/04	17
20	Generació d'aspectes que	7 dies	lu 19/04/04	do 25/04/04	19

	envien al mòdul la informació				
21	Creació del mòdul central	7 dies	lu 26/04/04	do 02/05/04	20
22	Aplicació que escriu un log d'emparellaments a un fitxer	0 dies	do 02/05/04	do 02/05/04	21
23	**Generació de diagrames de Seqüència**	21 dies	lu 03/05/04	do 23/05/04	18
24	Incorporació dels Aspectes que capturen informació sobre les crides	7 dies	lu 03/05/04	do 09/05/04	22
25	Incorporació del Gestor de Diagrama del PFC-Lérida	7 dies	lu 10/05/04	do 16/05/04	24
26	Unir diferents fils d'execució equivalents	7 dies	lu 17/05/04	do 23/05/04	25
27	Aplicació que crea diagrames d'ús per aplicacions distribuïdes	0 dies	do 23/05/04	do 23/05/04	26
28	Estudi en altres entorns distribuïts	12 dies	lu 24/05/04	vi 04/06/04	23
29	Redacció de la memòria del projecte	14 dies	sá 05/06/04	vi 18/06/04	28
30	Creació de la presentació virtual	14 dies	sá 05/06/04	vi 18/06/04	28
31	Final del projecte	0 dies	vi 18/06/04	vi 18/06/04	30

7.4 Riscos associats

Com a riscos de Projecte s'havien localitzat els següents:

- **Coneixement de l'AOP**
  El meu desconeixement de la programació orientada a aspectes pot causar variacions en les estimacions de temps inicials. Hem fet unes estimacions basades en suposicions, ja que mai he fet cap projecte d'aquest tipus i alhora de fer un pla de treball sempre és un punt a favor tenir experiència prèvia en projectes similars.

- **Aprenentatge i estudi.**
  Moltes de les tasques a realitzar en el projecte són d'aprenentatge de noves tecnologies i estudis de solucions. Fer l'estimació de temps d'aquestes tasques es molt arriscat, és difícil de saber quant temps trigarem a aprendre una tecnologia o decidir-nos per una o altre solució.
  A més el fet de trobar una manera d'emparellar les crides entre components remots, que és una de les parts més importants del projecte, no es pas un tema fàcil de mesurar en quan a temps.

- **L'AOP és encara una tecnologia en estudi.**
  El fet de ser la programació orientada en aspectes una tecnologia encara en fase d'estudi, pot fer que hagi encara problemes inherents als que no s'hagin donat prou solucions.

I com era d'esperar el risc associat a trobar una manera d'emparellar el fil d'execució del client amb el de l'execució de l'objecte remot va durar molt més del esperat.

Fins a la fita de "**Aplicació que escriu un log d'emparellaments a un fitxer"** es va complir el nostre calendari.
Però en lloc de tenir aquesta fita enllestida el 2 de Maig, no es va poder realitzar fins al 22 de Maig. En aquest moment portaven un retràs de significatiu de 3 setmanes.
Aquest retràs va endarrerir tot el projecte i fins al 5 de Juny no es va tenir una aplicació final prou estable.

No obstant es va començar a fer la memòria dintre de les dates previstes i es van realitzar en paral·lel les tasques d'estudi d'altres tecnologies i enllestiment de l'aplicació.

7.3 Calendari de treball final

Com ja s'ha explicat, al final el calendari va variar considerablement.
Un dels riscos que havíem detectat va convertir-se en un greu problema.
No trobàvem la manera idònia d'enllaçar el Thread del Client amb el del Servidor.

No obstant es va avançar molt en les tasques següents per a poder enllestir el projecte a la data prevista.

El calendari que va quedar al final va ser:

Id	Nom Tasca	Duració	Data Inici	Data Final	Predecessores
1	Inici del Projecte	0 dies	lu 15/03/04	lu 15/03/04	
2	**Estudi i familiarització amb AOP**	14 dies	lu 15/03/04	do 28/03/04	1
3	Estudi AOP	14 dies	lu 15/03/04	do 28/03/04	
4	Instal·lació de programari	3 dies	lu 15/03/04	mi 17/03/04	
5	Programari instal·lat	0 dies	mi 17/03/04	mi 17/03/04	4
6	Prova d'exemples senzills	4 dies	ju 18/03/04	do 21/03/04	5
7	Familiarització amb codi de J.L.Lerida	11 dies	ju 18/03/04	do 28/03/04	4
8	Implementació exemple de instrumentalització	7 dies	lu 22/03/04	do 28/03/04	6
9	Exemple senzill d'instrumentalització	0 dies	do 28/03/04	do 28/03/04	8
10	**Estudi de RMI**	21 dies	lu 15/03/04	do 04/04/04	1
11	Estudi informació disponible en crida RMI	7 dies	lu 15/03/04	do 21/03/04	
12	Estudi de com aparellar la crida a un RMI-stub (client) amb el RMI-Remot	14 dies	lu 22/03/04	do 04/04/04	11
13	**AOP i RMI**	14 dies	lu 29/03/04	do 11/04/04	2
14	Realització exemple d'instrumentalització en aplicació distribuïda	7 dies	lu 29/03/04	do 04/04/04	9
15	Exemple senzill d'instrumentalització en aplicació distribuïda	0 dies	do 04/04/04	do 04/04/04	14
16	Estudi de la millor estratègia per a la gestió de logs	7 dies	lu 05/04/04	do 11/04/04	14;15
17	Escollida la millor estratègia per la gestió de logs	0 dies	do 11/04/04	do 11/04/04	16

18	**Capturar emparellaments amb AOP**	39 dies	lu 12/04/04	ju 20/05/04	13
19	Estudi informació necessària per aparellament	30 dies	lu 12/04/04	ma 11/05/04	17
20	Generació d'aspectes que envien al mòdul la informació	7 dies	mi 12/05/04	ma 18/05/04	19
21	Creació del mòdul central	9 dies	mi 12/05/04	ju 20/05/04	19
22	**Generació de diagrames de Seqüència**	15 dies	vi 21/05/04	vi 04/06/04	18
23	Incorporació dels Aspectes que capturen informació sobre les crides	5 dies	vi 21/05/04	ma 25/05/04	21
24	Incorporació del Gestor de Diagrama del PFC-Lérida	5 dies	mi 26/05/04	do 30/05/04	23
25	Unir diferents fils d'execució equivalents	5 dies	lu 31/05/04	vi 04/06/04	24
26	Aplicació que crea diagrames d'ús per aplicacions distribuïdes	0 dies	vi 04/06/04	vi 04/06/04	25
27	Estudi en altres entorns distribuïts	7 dies	sá 05/06/04	vi 11/06/04	22
28	Redacció de la memòria del projecte	7 dies	sá 12/06/04	vi 18/06/04	27
29	Creació de la presentació virtual	7 dies	sá 12/06/04	vi 18/06/04	27
30	Final del projecte	0 dies	vi 18/06/04	vi 18/06/04	29

8. Conclusions

Amb l'enginyeria orientada a Aspectes em aconseguit lliurar-nos de la verticalitat de les aplicacions orientades a Objectes.
El fet de crear logs o diagrames de seqüència a partir d'una aplicació són unes de les múltiples possibilitats que ens ofereix.
La complexitat de seguir el comportament d'un procés al llarg d'una aplicació distribuïda , ha estat reduïda considerablement gràcies a la nostra eina.
Aquesta eina ha estat provada en altres entorns a part del RMI, com ara amb Servlets. Només cal variar el getUseCase per a que capturi les crides al Servlet.
Igual que amb aquesta prova, l'eina funcionaria en tots els escenaris implementats en Java, oferint un gran ventall de possibilitats.

No obstant degut a les complicacions en el calendari de treball ha estat impossible aprofundit tot el que haguessin volgut en l'estudi d'altres tipus de tecnologies a part del RMI.
És per això que es proposa aquest estudi com a futura línia de treball.

9. Valoració personal

Aquest projecte ha enriquit molt la meva base en java i sobretot m'ha obert un món de possibilitats gràcies a l'AOP.
No obstant ha sigut força complicat el desenvolupament d'aquest.
El fet de tractar-se d'un projecte on predominen les etapes d'estudi a les d'implementació, ha fet que el meu calendari inicial de treball variés totalment al que es va fer al final.
És molt difícil esbrinar els temps de tasques basades en la creativitat i l'estudi de noves tecnologies. Hagués sigut molt més fàcil realitzar un projecte amb una tecnologia coneguda i unes fites més clares.
No obstant, una de les fites de tot PFC és una part de innovació, i en aquest ha hagut força.
Al final, després de ocupar durant un semestre totes les hores lliures al projecte, he de reconèixer que he après força i m'ha obert noves perspectives de treball.
He aconseguit aprendre el que jo anomeno TRIGGERS de programa, i que si tingués la possibilitat en un futur de treballar amb Java, aquesta nova tecnologia em solucionaria molt la meva feina.