UNIVERSISTAT OBERTA DE CATALUNYA

ESTUDIOS DE INGENIERIA INFORMÁTICA

PROYECTO FINAL DE CARRERA

APLICACIONES WEB PARA TRABAJO COLABORATIVO

'DISEÑO Y PROTOTIPO DE UN SOFTWARE PARA LA REPLICACIÓN DE FICHEROS EN SISTEMAS DE TRABAJO EN GRUPO P2P'

Memoria

Alumno: **Ángel Navarro Estepa**

Consultor: **Fatos Xhafa**

ÍNDICE

1 INTRODUCCIÓN 5

1.1 UNA NUEVA FORMA DE ENTENDER LAS COMUNICACIONES 5
1.2 OBJETIVOS 5
1.3 MOTIVACIÓN 6

2 DESCRIPCIÓN DEL PROYECTO 7

2.1 INTRODUCCIÓN 7
2.2 EL PROYECTO 7
2.3 ALCANCE 8
2.4 ANÁLISIS DE REQUISITOS 8
2.4.1 bases del análisis de requisitos 8
2.4.2 requisitos nodo 9
2.4.3 requisitos alerta 10
2.5 TECNOLOGÍAS A UTILIZAR 11

3 ORGANIZACIÓN Y PLANIFICACIÓN 12

3.1 INTRODUCCIÓN 12
3.2 RELACIÓN DE TAREAS 12
3.3 PLANIFICACIÓN 15
3.3.1 bases de la planificación 15
3.3.2 hitos a resaltar 17
3.3.3 planificación 17

4 ESTUDIO FUNCIONAL 18

4.1 INTRODUCCIÓN 18
4.2 ÁMBITO DE LA SOLUCIÓN: NODO PEER& SUPERPEER 18
4.3 ACTORES 19
4.4 CASOS DE USO 19
4.4.1 casos de uso nivel I 19
4.4.2 casos de uso nivel II 20
4.5 FUNCIONALIDADES 22
4.5.1 conectar 22
4.5.2 desconectar 24
4.5.3 crear documentos xml 25
4.5.4 listar documentos xml 26
4.5.5 modificar documentos xml 27
4.5.6 borrar documento xml 28
4.5.7 configurar 29
4.5.8 salir 30
4.6 DIAGRAMAS DE ACTIVIDAD 30
4.6.1 actividad: nodos peer & superpeer 30
4.6.2 actividad conectar 31
4.6.3 actividad desconectar 35
4.6.4 actividad crear documento xml 36
4.6.5 actividad listar documentos xml 38
4.6.6 actividad modificar documento xml 39
4.6.7 actividad borrar documento xml 41
4.6.8 actividad configurar 43

4.6.9 actividad salir ... 43
4.7 INTERFICIES DE USUARIO ... 44
4.7.1 tipo de interficie ... 44
4.7.2 marco principal ... 44
4.7.3 opción conexión – gestionar ... 46
4.7.4 opción ficheros – listar ... 47
4.7.5 opción ficheros – nuevo ... 49
4.7.6 opción sistema – configurar ... 50
4.7.7 opción sistema – salir ... 51
4.7.8 avisos ... 51
4.8 DISEÑO DE LA BASE DE DATOS ... 52
4.8.1 diseño lógico ... 54
4.9 DISEÑO DEL PLAN DE PRUEBAS ... 59

5 ESTUDIO TÉCNICO ... 65

5.1 INTRODUCCIÓN ... 65
5.2 NECESIDADES DE ARQUITECTURA ... 65
5.3 TECNOLOGÍAS UTILIZADAS ... 66
5.3.1 java como elemento común ... 67
5.3.2 jxta y la red peer to peer ... 67
5.3.3 swing como frontal ... 71
5.3.4 base de datos derby ... 73
5.3.5 tratamiento documentos xml con jdom ... 74
5.3.6 sistema de trazas con log4j ... 75
5.4 DISEÑO DE ARQUITECTURA ... 76
5.5 OBJETOS PRINCIPALES. ESTRUCTURACIÓN ... 77
5.5.1 constantes ... 78
5.5.2 manejo de ficheros ... 78
5.5.3 marco principal ... 79
5.5.4 gestión de la base de datos ... 79
5.5.5 crear identificador únicos ... 79
5.5.6 mensaje ... 79
5.5.7 peer ... 80
5.5.8 comunicación unicast ... 80
5.5.9 comunicación multicast ... 80
5.5.10 superpeer ... 81
5.5.11 documentos xml ... 81
5.6 OPERACIONES DE NEGOCIO ... 81
5.6.1 iniciar la aplicación – nodo superpeer ... 81
5.6.2 iniciar la aplicación – nodo peer ... 85
5.6.3 conectar nodo peer ... 87
5.6.4 desconectar – nodo peer ... 95
5.6.5 crear documento xml – nodo peer ... 96
5.6.6 listar documento – nodo peer ... 99
5.6.7 baja documento xml – nodo peer ... 100
5.6.8 modificar documento xml – nodo peer ... 103
5.6.9 configuración – nodo peer ... 106
5.6.10 salir – nodo peer ... 109
5.7 LÓGICA DE NEGOCIO ... 109
5.7.1 introducción ... 109
5.7.2 recepción alta documento – nodo superpeer ... 109
5.7.3 recepción baja documento – nodo superpeer ... 110
5.7.4 recepción modificar documento – nodo superpeer ... 111
5.7.5 recepción alta documento – nodo peer ... 112
5.7.6 recepción baja documento – nodo peer ... 113
5.7.7 recepción modificar documento – nodo peer ... 114

5.7.8 petición actualizaciones – nodo superpeer 115
5.7.9 recepción actualizaciones – nodo superpeer 117
5.7.10 recepción actualizaciones – nodo peer 119
5.7.11 petición dame nuevo – nodo peer 120
5.7.12 petición dame nuevo – nodo superpeer (data center) 121

6 ESTUDIO DEL RENDIMIENTO 123

6.1 INTRODUCCIÓN 123
6.2 ESCENARIOS DE RENDIMIENTO 123
6.2.1 conexión al grupo – inicial 123
6.2.2 conexión al grupo – no inicial 125
6.2.3 alta de documento 126
6.2.4 actualizarse tras inactivo – escenario 1 126
6.2.5 actualizarse tras inactivo – escenario 2 128
6.2.6 actualizarse tras inactivo – escenario 3 129
6.2.7 activarse tras inactivo – escenario 4 130
6.3 CONCLUSIONES 130

7 CONCLUSIONES 132

8 TRABAJO FUTURO 134

9 ANEXOS 135

9.1 ESTUDIO DE LA BASE DE DATOS 135
9.1.1 introducción 135
9.1.2 sgbd postgresql 135
9.1.3 sgbd javadb 137
9.1.4 sgbd oracle 139
9.1.5 conclusiones 140
9.2 INTERFICIES JAVA 142
9.2.1 panel principal 142
9.2.2 Gestión de la BBDD 148
9.2.3 peer 174
9.2.4 superpeer 187

10 BIBLIOGRAFIA 196

10.1 ESTUDIO DE LA BASE DE DATOS 196
10.2 ESTUDIO TÉCNICO 196
10.3 OTROS 196

1 INTRODUCCIÓN

1.1 UNA NUEVA FORMA DE ENTENDER LAS COMUNICACIONES

Es por todos conocidos aplicaciones como *Emule, Kazaa, Torrent…*y la idea global de lo que permiten realizar. Cuando hablamos de alguna herramienta de este tipo lo que nos viene a la cabeza es la posibilidad de descargarme películas (si son las primicias de cine mucho mejor), poder disponer de la discografía de cantantes que nunca había oído hablar (pero resulta que hay una canción de moda y por ello me bajo toda la discográfica del grupo) y tener cualquier software de forma gratuita.

Aunque no podemos mirar para otro lado y sabemos que eso forma parte de la realidad, todo esto tiene otra lectura: Existe una nueva forma de comunicación.

Todos estos sistemas tiene una cosa en común, su comunicación no es la clásica *cliente – servidor* sino que podemos hablar de una comunicación *peer to peer*.

Por ello, aunque la idea más generalizada de estos sistemas es la parte más *comercial* de lo que pueden hacer, existe una maravillosa noticia y es que tenemos ante nosotros una nueva forma de comunicación en internet que permite abrir otra vía de diseño de las tecnologías de la información.

Tenemos ante nosotros la posibilidad de redefinir gran parte de las comunicaciones tradicionales y diseñar nuevos entornos colaborativos mucho más potentes permitiendo poder revolucionar el mundo de las TI. Todo esto sin olvidar otra realidad, el trabajo a distancia es un hecho y esto no tiene que significar no trabajar en grupo. Los vectores movilidad y grupo tienen que ir de la mano para permitir trabajos con máxima calidad.

1.2 OBJETIVOS

Como objetivo principal se establece el crear una aplicación mediante la cual poder crear un entorno colaborativo de trabajo donde ofrecer una herramienta de trabajo para poder compartir documentos y habilitar los mecanismos necesarios para que esta plataforma ofrezca los servicios para que cada miembro del equipo disponga de los documentos de otros compañeros actualizados en todo momento.

Directamente profundizar en la forma de comunicación *peer to peer* y poder realizar un pequeño estudio donde poder evaluar el

rendimiento del sistema dependiendo de una serie de parámetros dentro de la propia solución:

- Número de documento
- Número de miembros del grupo
- Número de cambios
- Tiempo de inactividad de los nodos

Y en base a estos parámetros poder evaluar rendimiento, fiabilidad y tiempo de respuesta del sistema creado evaluando indirectamente las comunicaciones *peer to peer*.

1.3 MOTIVACIÓN

En los tiempos que vivimos, y sobre todo, hacia donde vamos, el trabajo en grupo es uno de los puntos más importantes a la hora de llevar un proyecto a cabo.

Cada vez se está incentivando no sólo el estudio a distancia, que ya es una realidad, sino el trabajo con diferentes equipos situados en lugares geográficamente distintos. La globalización ha llevado a que una misma compañía ofrece servicios de soporte *full time* utilizando el área geográfica como punto vital en esta estrategia.

El trabajo en casa es un hecho y eso no tiene que impedir que no pertenezcas a un equipo de trabajo.

Por eso, es muy importante desarrollar herramientas y sistemas que tengan en cuenta esa premisa para el trabajo en equipo sin que la barrera geográfica sea un impedimento.

Todo esto provoca que sea de gran interés el área conocida como *Aplicaciones web para trabajo colaborativo*.

Pienso que ya es la base del trabajo de hoy, pero sobre todo, y viendo el rumbo que está cogiendo todo, será la base de trabajo del mañana.

Con este proyecto, se establecerán las bases del trabajo en equipo donde poder compartir los distintos documentos creados por cada miembro del grupo. La actualización de esta información y la posibilidad de poder disponer *online* de la última versión de cada uno de ellos, es de vital importancia para el objetivo final de un proyecto: entregar un producto de máxima calidad.

Mediante el desarrollo de este proyecto, se abre la posibilidad de conocer y entrar en profundidad del tipo de comunicaciones *peer to peer* y tener otra visión distinta al clásico *cliente -.servidor*.

2 DESCRIPCIÓN DEL PROYECTO

2.1 INTRODUCCIÓN

En descripción del proyecto empezaremos a ver qué se tendrá que hacer; en base a qué requisitos y definiendo el alcance del mismo. A parte, se empezará a indicar a grandes rasgos la tecnología a utilizar.

2.2 EL PROYECTO

El diseño y prototipo de un software para la replicación de ficheros en sistemas de trabajo en grupos P2P, englobado dentro del área de Aplicaciones web para trabajo colaborativo, consistirá en un sistema de alertas sobre un entorno descentralizado y distribuido que permitirá a un equipo de trabajo (grupo de usuarios) compartir documentos que podrían formar parte de un proyecto común.

Cada miembro del equipo podrá crear y editar documentos compartiéndolos con los demás miembros del grupo y a su vez, tendrá actualizada toda la documentación que otros compañeros hayan podido crear/editar.

Con este fin, se diseñará un entorno de trabajo distribuido y descentralizado que implantará ECA-rules (Evento-Condición-Acción):

- Evento: Señal que desencadena la invocación
- Condición: Parte lógica, que si se evalúa a cierto, hace que la acción se lleve a cabo
- Acción: Cambios a realizar

Para crear esta red entre los miembros del equipo, se crearán conexiones entre nodos (miembros del grupo) *peer to peer* con la existencia de un *SuperPeer* que será el encargado de mantener la consistencia entre los diferentes nodos y tener la información actualizada en todo momento.

De esta forma nos alejamos de un sistema cliente/servidor donde el depender de un pote común (servidor) puede provocar el colapso del sistema y la posible pérdida de información en caso de errores en el servidor.

2.3 ALCANCE

El prototipo a construir será un sistema *SuperPeer* con un número reducido de nodos tipo *peer* y con un solo nodo tipo *SuperPeer*.

Para reducir la complejidad del mismo, únicamente se manejarán documentos en formato XML con los que cada nodo, podrá crear, modificar o eliminar. Para ello, se habilitará una interficie para que el usuario pueda realizar esas acciones desde la aplicación.

Las acciones a realizar sobre los documento sólo las podrá hacer de aquéllos propiedad del nodo; de tal forma que no podrá modificar ni eliminar un documento no creado por el propio nodo.

La implementación del prototipo será genérica en cuanto a número de nodos, aunque en la fase de *deployment* se limitará el estudio a 4 ó 5 nodos.

Cada nodo dispondrá de una base de datos donde almacenará la situación de cada uno de los documentos con los que el grupo trabaja. Esta situación será la propia del nodo.

El nodo *SuperPeer* dispondrá de otra base de datos donde tendrá la situación real de cada uno de los documentos que maneja el grupo. A parte, este nodo hará las funciones de *Data Center*; almacenando los documento originales para poder habilitarlos aquellos nodos que estuvieran inactivos en el momento de creación del documento y que, cuando se conectan de nuevo a la red, el nodo propietario del documento está off-line. De esta forma se mantiene la consistencia del sistema y se permite que estos nodos puedan disponer de los nuevos documentos sin tener que esperar que el propietario se conecta a la red.

2.4 ANÁLISIS DE REQUISITOS

2.4.1 BASES DEL ANÁLISIS DE REQUISITOS

El siguiente apartado describe cada uno de los requisitos detectados para diseñar la aplicación tal cual las especificaciones aportadas por el consultor.

Se realizarán dos agrupaciones: Por un lado requisitos orientados a las funcionalidades propias del nodo. Funcionalidades que podrá realizar y que estarán vinculadas a la acción del usuario.

Por otro lado a funcionalidades no vinculadas directamente (pero si indirectamente) a la acción del usuario y si a alertas que se dispararán con determinados sucesos.

Cada requisito estará numerado de forma secuencial siguiendo el siguiente patrón:

Para requisitos tipo nodo → REQ-nx

Donde *n* indica que es un requisito tipo nodo.

Donde *x* es un número secuencial

Para requisitos tipo alerta →REQ-ax

Donde a indica que es un requisito tipo nodo.

Donde *x* es un número secuencial

Se utilizará esta codificación a lo largo del proyecto para poder identificar el requisito en cuestión.

2.4.2 REQUISITOS NODO

A continuación se describen todos los requisitos identificados para cumplir las funcionalidades necesarias de un Nodo.

REQ_n1: Crear documento XML

La solución tiene que permitir al usuario poder crear documentos XML (únicamente se podrá trabajar con documentos XML).

Estos documentos se almacenarán en una carpeta definida por el sistema.

REQ_n2: Listar documentos XML

Se podrá listar todos los documentos XML existentes. A partir de ese listado, se podrá acceder al documento.

REQ_n3: Modificar documento XML

Se podrá acceder a un documento XML y modificar su contenido. Una vez modificado, quedarán guardados dichos cambios.

Sólo se podrán modificar aquellos documentos creados por el propio nodo.

REQ_n4: Borrar documento XML

Se podrá borrar cualquier documento XML creado por el propio nodo. La acción de borrar requiere confirmación por parte del usuario.

REQ_n5: Conectar/desconectar

Al iniciar la solución ésta trabajará en modo *Sin Conexión*. Se permitirá poder conectar/desconectar a la red (grupo) cuando el usuario indique.

2.4.3 REQUISITOS ALERTA

A continuación se describen todos los requisitos identificados como *Alerta* que se activarán con la consecución de determinados sucesos.

REQ_a1: Informar cambios en los documentos

Cualquier cambio que se produzca en un documento de un nodo (alta, baja o modificación) tiene que ser notificado al grupo.

REQ_a2: Petición de actualización

Cuando un nodo se conecte, tiene que ser capaz de sincronizar su documentación con los posibles cambios sufridos en otros nodos.

Al hablar de conexión se entiende que es cada vez que se pone un nodo en marcha dentro del grupo; no cuando se arranque la solución y se trabaje sin conexión.

REQ_a3: Informar a Nodo dónde buscar actualización

Indicar a un nodo donde poder ir a buscar la información (documento/s) que se tiene que actualizar.

REQ_a4: Informar al resto de nodos de cambios

Poder informar al resto de nodos del grupo de cambios realizados en un nodo.

REQ_a5: Comunicación con otros grupos

Conexión con otros grupos mediante nodos *SuperPeer*.

2.5 TECNOLOGÍAS A UTILIZAR

Se trata de desarrollar una aplicación con un carácter de comunicaciones muy marcado y que pueda operar en cualquier sistema existente. Tener en cuenta que los nodos pueden estar repartidos geográficamente y en plataforma dispares.

Otro punto importante a la hora de elegir las tecnologías a utilizar es la necesidad de disponer de un soporte de base de datos pero con el requisito de no *obligar* al usuario final tener que comprar, instalar y configurar un sistema gestor de base de datos. Esto tiene que ser transparente para el usuario final.

Por todo esto, se utilizará **J2EE** como plataforma y lenguaje de programación con soporte **DERBY** para el sistema gestor de base de datos.

Como se ha mencionado, a nivel de sistema operativo no hay vinculación con ninguno de ellos.

NOTA: En la parte de definición técnica se detallan todos estos elementos.

3 ORGANIZACIÓN Y PLANIFICACIÓN

3.1 INTRODUCCIÓN

Como organización y planificación se indicarán todos los pasos necesarios y para lograr los objetivos establecidos dentro del marco temporal indicado.

3.2 RELACIÓN DE TAREAS

La organización global del proyecto se he estructurado en una serie de tareas para garantizar que el resultado final sea un proyecto que cumpla con todos los objetivos establecidos y asegure la entrega de todos los puntos necesarios indicados en la docencia de la propia universidad.

Se ha seguido una codificación de tareas incremental siguiendo un orden cronológico de realización.

Tarea_1: Formalizar propuesta de proyecto

Una vez leído plan docente, se seleccionará una propuesta de proyecto para indicar al consultor el área de donde se le asignará uno o en caso de disponer de una idea clara a realizar, proponerla para ser estudiada.

Con todo esto, se formalizará el proyecto a llevar a cabo.

Tarea_2: Estudio material entregado por el consultor

Estudio del material entregado por el consultor sobre el proyecto a realizar. Se puede tratar de documentación y material de un proyecto de partida; o de documentación del proyecto a realizar.

Tarea_3: Planificación y Análisis de requisitos del proyecto

Elaborar los siguientes puntos del proyecto:

Tarea_3.1: Descripción del proyecto

Describir e identificar objetivos y alcance del proyecto.
Identificar tareas a realizar

Tarea_3.2: Tareas

Detallar las tareas a llevar a cabo que nos permitirán llevar a buen puerto el proyecto.

Tarea_3.3: Planificación del proyecto

Establecer plan de trabajo donde se indicará fechas de entrega de cada uno de los hitos establecidos.

Tarea_3.4: Análisis de requisitos

Determinar funcionalidades propias del software (funcionalidades de cada uno de los nodos y reglas a desarrollar en el sistema de alertas).

Tarea_3.5: Estudio de la base de datos

Estudio de cada una de las bases de datos identificadas y selección de la más adecuada para nuestros propósitos.

Tarea_4: Estudio funcional

Detalle de las funcionalidades del software (identificadas en el análisis de requisitos) y definición de los casos de uso.

Diagramas de casos de uso, diagramas de actividad y definición de la base de datos.

Tarea_5: Estudio técnico

Diseño de la arquitectura del software. Descripción de los diagramas de secuencia. Detalle de las tareas a realizar en el desarrollo.

Tarea_6: Diseño del plan de pruebas

Una vez realizado estudio funcional y técnico, elaboración del plan de pruebas (tanto funcional como técnico) a llevar a cabo una vez el software esté desarrollado y una vez que se haya implantado.

Tarea_7: Implementación

Llevar a cabo el desarrollo del software siguiendo los pasos funcionales y técnicos indicados.

Tarea_8: Ejecución del plan de pruebas

Llevar a cabo el plan de pruebas diseño en la fase de análisis. Solucionar cualquier problemática encontrada.

Tarea_9: Estudio del rendimiento

Someter al software creado a las pruebas de rendimiento y estrés establecidas para poder obtener datos estadísticos donde poder extraer conclusiones.

Tarea_10: Despliegue del software (prototipo)

Entrega de una primera versión del software con toda la documentación generada. Se adjunta plan de pruebas para poder ser pasado.

Tarea_11: Revisión del prototipo

Revisar todos los puntos detectados en la primera entrega del prototipo.

Tarea_12: Manuales

Generación de los manuales de usuario y administración del software creado.

Tarea_13: Memoria final

Generación de la memoria final del proyecto.

Tarea_14: Presentación

Generación de la presentación del proyecto.

Tarea_15: Entrega final

Entrega de la versión definitiva del software y recopilación de toda la documentación generada.

Entrega de la memoria final y presentación del proyecto.

Tarea_16: Defensa virtual

Defensa virtual del proyecto presentado por el alumno.

3.3 PLANIFICACIÓN

3.3.1 BASES DE LA PLANIFICACIÓN

La planificación del proyecto está condicionada por tres elementos:

- Estimación de tiempo de cada una de las tareas identificadas.

- Persona que lleva a cabo la tarea
- Calendario de trabajo

Estimación de tiempo de cada tarea

Se podrá observar la estimación de carga de cada una de las tareas en el Diagrama de Gantt mostrado en el apartado 4.3.

Persona que lleva a cabo la tarea

Como en todo proyecto, hay que definir un equipo de trabajo.

En este caso el equipo de trabajo será el propio alumno que lleva a cabo todo el ámbito del proyecto; desde la parte de definición, como la parte de desarrollo, documentación y explicación.

Dentro del Diagrama de Gantt se indicará en cada tarea la persona que la lleva a cabo (en este caso *Angel*). Como tal, esta persona tendrá definido un calendario de trabajo donde se indicará cuál es su horario de trabajo (ver siguiente punto).

Calendarios de trabajo

Se han definido el siguiente calendario de trabajo:

Calendario de trabajo 'Angel'

Lunes a Jueves	20:00h	23:00h
Viernes	17:00h	23:00h
Sábado y Domingo	10:00h	14:00h
	16:00h	22:00h

Con las siguientes excepciones:

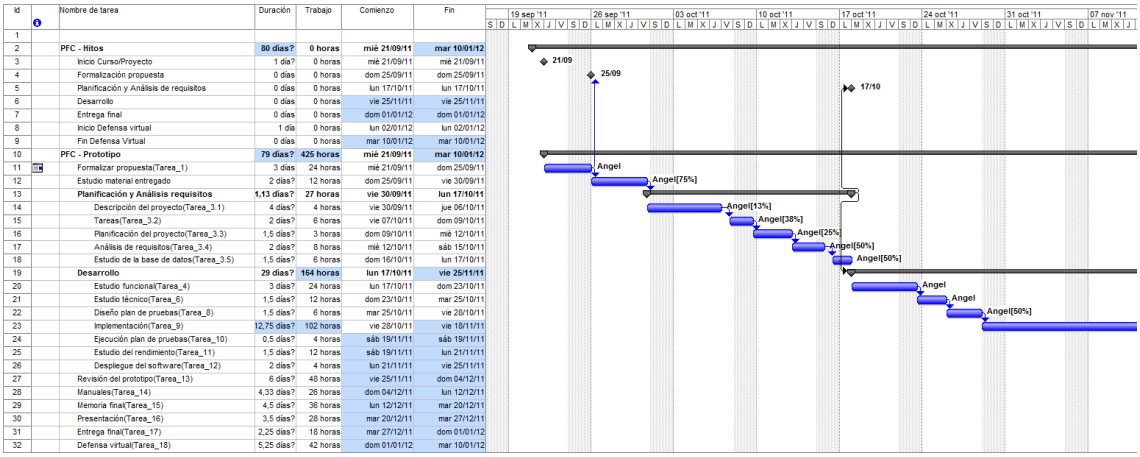
01/11/2011	10:00h	14:00h
	16:00h	22:00h
25/12/2011	10:00h	14:00h
31/12/2011	10:00h	14:00h
01/01/2012	16:00h	22:00h

3.3.2 HITOS A RESALTAR

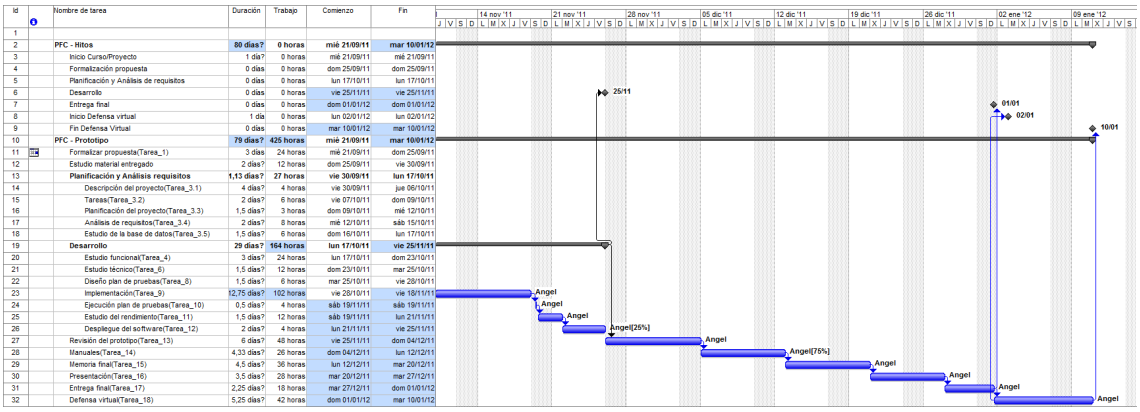
La siguiente tabla muestra los hitos más importantes del proyecto:

Hito	Fecha prevista
Inicio proyecto	21/09/2011
Formalización propuesta	25/09/2011
Planificación y Análisis de requisitos – Entrega PAC1	17/10/2011
Desarrollo – Entrega PAC2	25/11/2011
Entrega final – PAC 3	01/01/2012
Inicio Defensa Virtual	02/01/2012
Fin Defensa Virtual	10/01/2012

3.3.3 PLANIFICACIÓN



Id	Nombre de tarea	Duración	Trabajo	Comienzo	Fin
1					
2	**PFC - Hitos**	**80 días?**	**0 horas**	**mié 21/09/11**	**mar 10/01/12**
3	Inicio Curso/Proyecto	1 día?	0 horas	mié 21/09/11	mié 21/09/11
4	Formalización propuesta	0 días	0 horas	dom 25/09/11	dom 25/09/11
5	Planificación y Análisis de requisitos	0 días	0 horas	lun 17/10/11	lun 17/10/11
6	Desarrollo	0 días	0 horas	vie 25/11/11	vie 25/11/11
7	Entrega final	0 días	0 horas	dom 01/01/12	dom 01/01/12
8	Inicio Defensa virtual	1 día	0 horas	lun 02/01/12	lun 02/01/12
9	Fin Defensa Virtual	0 días	0 horas	mar 10/01/12	mar 10/01/12
10	**PFC - Prototipo**	**79 días?**	**425 horas**	**mié 21/09/11**	**mar 10/01/12**
11	Formalizar propuesta(Tarea_1)	3 días	24 horas	mié 21/09/11	dom 25/09/11
12	Estudio material entregado	2 días?	12 horas	dom 25/09/11	vie 30/09/11
13	**Planificación y Análisis requisitos**	**1,13 días?**	**27 horas**	**vie 30/09/11**	**lun 17/10/11**
14	Descripción del proyecto(Tarea_3.1)	4 días?	4 horas	vie 30/09/11	jue 06/10/11
15	Tareas(Tarea_3.2)	2 días?	6 horas	vie 07/10/11	dom 09/10/11
16	Planificación del proyecto(Tarea_3.3)	1,5 días?	3 horas	dom 09/10/11	mié 12/10/11
17	Análisis de requisitos(Tarea_3.4)	2 días?	8 horas	mié 12/10/11	sáb 15/10/11
18	Estudio de la base de datos(Tarea_3.5)	1,5 días?	6 horas	dom 16/10/11	lun 17/10/11
19	**Desarrollo**	**29 días?**	**164 horas**	**lun 17/10/11**	**vie 25/11/11**
20	Estudio funcional(Tarea_4)	3 días?	24 horas	lun 17/10/11	dom 23/10/11
21	Estudio técnico(Tarea_6)	1,5 días?	12 horas	dom 23/10/11	mar 25/10/11
22	Diseño plan de pruebas(Tarea_8)	1,5 días?	6 horas	mar 25/10/11	vie 28/10/11
23	Implementación(Tarea_9)	12,75 días?	102 horas	vie 28/10/11	vie 18/11/11
24	Ejecución plan de pruebas(Tarea_10)	0,5 días?	4 horas	sáb 19/11/11	sáb 19/11/11
25	Estudio del rendimiento(Tarea_11)	1,5 días?	12 horas	sáb 19/11/11	lun 21/11/11
26	Despliegue del software(Tarea_12)	2 días?	4 horas	lun 21/11/11	vie 25/11/11
27	Revisión del prototipo(Tarea_13)	6 días?	48 horas	vie 25/11/11	dom 04/12/11
28	Manuales(Tarea_14)	4,33 días?	26 horas	dom 04/12/11	lun 12/12/11
29	Memoria final(Tarea_15)	4,5 días?	36 horas	lun 12/12/11	mar 20/12/11
30	Presentación(Tarea_16)	3,5 días?	28 horas	mar 20/12/11	mar 27/12/11
31	Entrega final(Tarea_17)	2,25 días?	18 horas	mar 27/12/11	dom 01/01/12
32	Defensa virtual(Tarea_18)	5,25 días?	42 horas	dom 01/01/12	mar 10/01/12



Id	Nombre de tarea	Duración	Trabajo	Comienzo	Fin
1					
2	**PFC - Hitos**	**80 días?**	**0 horas**	**mié 21/09/11**	**mar 10/01/12**
3	Inicio Curso/Proyecto	1 día?	0 horas	mié 21/09/11	mié 21/09/11
4	Formalización propuesta	0 días	0 horas	dom 25/09/11	dom 25/09/11
5	Planificación y Análisis de requisitos	0 días	0 horas	lun 17/10/11	lun 17/10/11
6	Desarrollo	0 días	0 horas	vie 25/11/11	vie 25/11/11
7	Entrega final	0 días	0 horas	dom 01/01/12	dom 01/01/12
8	Inicio Defensa virtual	1 día	0 horas	lun 02/01/12	lun 02/01/12
9	Fin Defensa Virtual	0 días	0 horas	mar 10/01/12	mar 10/01/12
10	**PFC - Prototipo**	**79 días?**	**425 horas**	**mié 21/09/11**	**mar 10/01/12**
11	Formalizar propuesta(Tarea_1)	3 días	24 horas	mié 21/09/11	dom 25/09/11
12	Estudio material entregado	2 días?	12 horas	dom 25/09/11	vie 30/09/11
13	**Planificación y Análisis requisitos**	**1,13 días?**	**27 horas**	**vie 30/09/11**	**lun 17/10/11**
14	Descripción del proyecto(Tarea_3.1)	4 días?	4 horas	vie 30/09/11	jue 06/10/11
15	Tareas(Tarea_3.2)	2 días?	6 horas	vie 07/10/11	dom 09/10/11
16	Planificación del proyecto(Tarea_3.3)	1,5 días?	3 horas	dom 09/10/11	mié 12/10/11
17	Análisis de requisitos(Tarea_3.4)	2 días?	8 horas	mié 12/10/11	sáb 15/10/11
18	Estudio de la base de datos(Tarea_3.5)	1,5 días?	6 horas	dom 16/10/11	lun 17/10/11
19	**Desarrollo**	**29 días?**	**164 horas**	**lun 17/10/11**	**vie 25/11/11**
20	Estudio funcional(Tarea_4)	3 días?	24 horas	lun 17/10/11	dom 23/10/11
21	Estudio técnico(Tarea_6)	1,5 días?	12 horas	dom 23/10/11	mar 25/10/11
22	Diseño plan de pruebas(Tarea_8)	1,5 días?	6 horas	mar 25/10/11	vie 28/10/11
23	Implementación(Tarea_9)	12,75 días?	102 horas	vie 28/10/11	vie 18/11/11
24	Ejecución plan de pruebas(Tarea_10)	0,5 días?	4 horas	sáb 19/11/11	sáb 19/11/11
25	Estudio del rendimiento(Tarea_11)	1,5 días?	12 horas	sáb 19/11/11	lun 21/11/11
26	Despliegue del software(Tarea_12)	2 días?	4 horas	lun 21/11/11	vie 25/11/11
27	Revisión del prototipo(Tarea_13)	6 días?	48 horas	vie 25/11/11	dom 04/12/11
28	Manuales(Tarea_14)	4,33 días?	26 horas	dom 04/12/11	lun 12/12/11
29	Memoria final(Tarea_15)	4,5 días?	36 horas	lun 12/12/11	mar 20/12/11
30	Presentación(Tarea_16)	3,5 días?	28 horas	mar 20/12/11	mar 27/12/11
31	Entrega final(Tarea_17)	2,25 días?	18 horas	mar 27/12/11	dom 01/01/12
32	Defensa virtual(Tarea_18)	5,25 días?	42 horas	dom 01/01/12	mar 10/01/12

4 ESTUDIO FUNCIONAL

4.1 INTRODUCCIÓN

El estudio funcional se organizará de la siguiente forma:

- Definición/diagrama de casos de uso: Definir los casos de uso y sus diagramas teniendo en cuenta los actores.
- Definición de las funcionalidades: Partiendo de los requisitos, detallar la funcionalidad de cada uno de ellos.
- Diagrama de actividad: Definición de los diagramas de actividad de la solución.
- Definición de las interficies: Indicar cómo será la interficie gráfica de la aplicación con la que los usuarios podrán interactuar.
- Diseño de la base de datos: Definición del modelo de base de datos necesario para el correcto funcionamiento de la solución.
- Definición del juego de pruebas: Diseño y redacción del plan de pruebas que una vez desarrollado el prototipo, se ejecutará para confirmar el correcto funcionamiento de la solución.

4.2 ÁMBITO DE LA SOLUCIÓN: NODO PEER& SUPERPEER

Por regla general estamos acostumbrados hablar de soluciones que pueden tener usuarios con distinto perfil. En función de ese perfil, la aplicación dejará hacer unas u otras funcionalidades.

En el caso que nos ocupa no es así. Nuestro prototipo estará formado por dos aplicaciones con objetivos bien diferenciados; estas dos aplicaciones las conoceremos como *Nodo Peer* y *Nodo SuperPeer*.

Cada uno de estos dos nodos tiene una misión concreta.

El *Nodo Peer* es el encargado de ofrecer una interficie al usuario y habilitar los mecanismos necesarios para disponer de las funcionalidades oportunas para poder crear/editar documentos XML y compartirlos con los demás nodos del grupo.

El *Nodo SuperPeer* será el encargado de orquestar la actividad creada por los *Nodos Peers*. Como tal, este nodo no dispone de interficie y el usuario no puede interactuar con él. A parte, el *superpeer* hará las funciones de *data center*. Esto significa que se guardará el documento xml inicial (documento informado por cualquier nodo en el momento de su alta) en disco para poder informarlo aquel nodo que en el momento de la alta no estaba *on-line* y que al conectarse al grupo y

recibir las actualizaciones, no encuentra al nodo propietario *on-line* por lo que recurre al *data center* para obtener dicho documento.

Aunque son dos módulos distintos, tienen que trabajar conjuntamente para que el resultado u objetivo final se pueda cumplir.

4.3 ACTORES

La utilización de actores en esta solución es muy sencilla y reducida.

Como ya se ha explicado, se trata de una solución final pensada para ser utiliza en un ámbito de trabajo en grupo con el fin de poder compartir documentos (nodos *peer*).

Esta aplicación será de fácil uso y cada usuario que quiera utilizarla (poder compartir documentos dentro del grupo) tendrá que tenerla instalada.

Con esto queremos decir que la aplicación de usuario (nodo *peer*) como tal será única y no tendrá distintos perfiles; aquel que la tenga instalada y configurada podrá hacer lo mismo que otro miembro del grupo que también la tenga.

Otra cosa es conocer todos los mecanismos técnicos necesarios para poder llevar a cabo este fin (como los nodos s*uperpeer)*, pero estos puntos se tratarán más adelante en el estudio técnico.

Por todo ello sólo se define un tipo de Actor que será el usuario que utilice la aplicación (*nodo Peer*). Todos los usuarios de la aplicación podrán realizar las mismas acciones.

Aunque sería entrar en aspectos más técnicos, decir que el nodo *superpeer* no dispondrá de usuarios directos pero si de usuarios (procesos) indirectos, como serán cada uno de los *nodos peer* que se creen formando el grupo que tendrá que gestionar.

4.4 CASOS DE USO

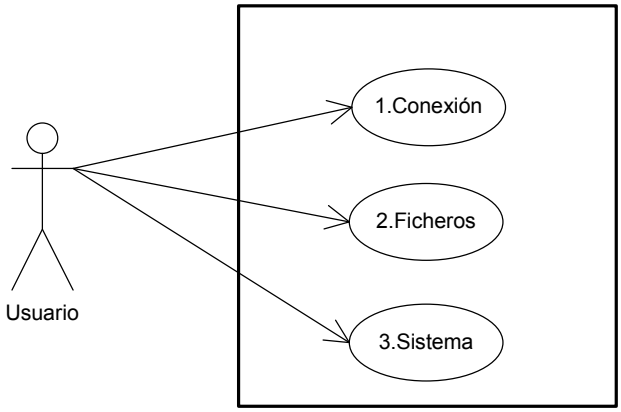
4.4.1 CASOS DE USO NIVEL I

Al tratarse de una solución con un enfoque técnico y no tanto de usuario, los casos de uso son bastante sencillos.

Recordamos que la vertiente usuario (la que dispone de interficie con la que el usuario podrá interactuar) de la aplicación es el nodo *peer*.

Por otro lado y tal y como hemos comentado, sólo existe un actor que es el propio usuario de la aplicación.

A continuación se detalla el caso de uso de nivel I:

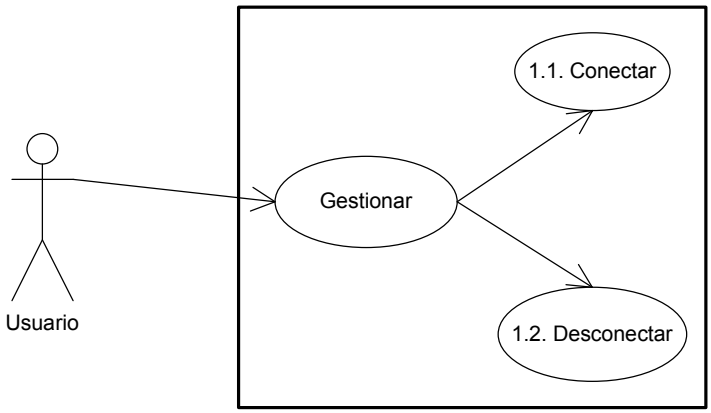


4.4.2 CASOS DE USO NIVEL II

Un nodo del tipo *peer* es la visión gráfica del proyecto y básicamente se encarga de gestionar dos áreas: la parte que trata con los ficheros xml que se compartirán con los otros nodos el grupo; y la propia gestión de la conexión o desconexión al grupo de nodos. A parte, una pequeña parte de sistema para controlar algún aspecto de la propia configuración del sistema y la opción de salir.

1. <u>Conexión</u>

Se trata de gestionar todo lo referente a la conexión con el grupo de nodos del sistema P2P.

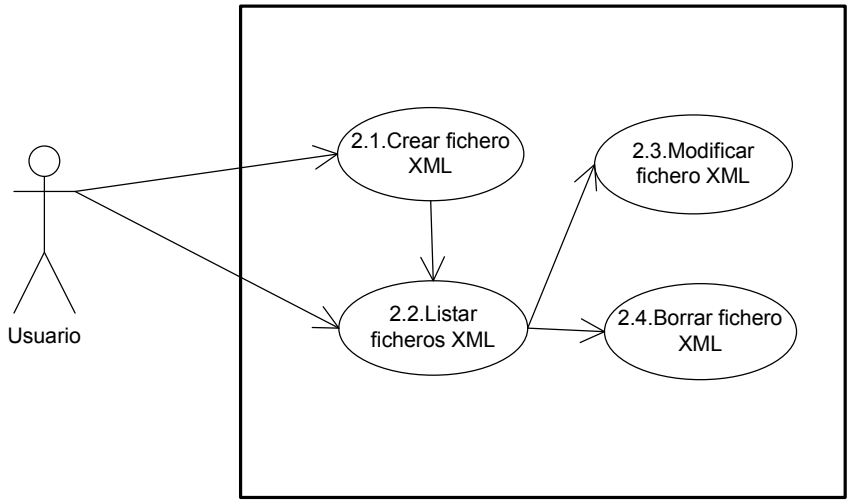


Una vez el usuario ha iniciado la aplicación y accede a la opción de menú *Gestionar*, se le mostrará una misma pantalla y en ella tendrá disponible las opciones:

- **1.1.** Conectar: La aplicación se conectará a la red P2P del grupo al que pertenece. Una vez conectado, se iniciará el sistema de sincronización de los ficheros (en caso de ser necesario) y de obtención de los cambios que los otros componentes del grupo hayan podido realizar sobre sus documentos. A partir de ese momento, el usuario estará *online* por lo que cualquier cambio o alta realizada, será notificada de forma automática al grupo.
- **1.2.** Desconectar: La aplicación se desconectará del grupo de nodos y trabajará en modo desconexión (cualquier cambio no se notificará al grupo y el nodo no será notificado de ningún cambio).

2. Ficheros

Se trata de gestionar todo lo que un usuario puede hacer con los ficheros XML que la aplicación puede tratar.



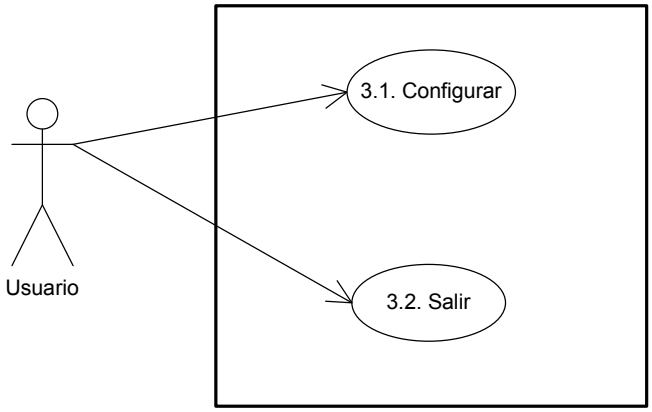
Para empezar a trabajar con ficheros, el usuario tendrá que iniciar la aplicación y una vez dentro, pulsar la opción de menú *Ficheros* y a partir de ahí podrá realizar los siguientes pasos:

- **2.1.** Crear fichero XML: Podrá crear un nuevo documento XML.

- **2.2.** Listar ficheros XML: El usuario podrá visualizar todos los ficheros XML disponibles en el grupo; no sólo los creados por el propio nodo sino por otros nodos.
- **2.3.** Modificar fichero XML: Una vez seleccionado un fichero, el usuario podrá modificarlo siempre y cuando sea un fichero creado por el propio nodo.
- **2.4.** Borrar fichero XML: Una vez seleccionado un fichero, el usuario podrá borrarlo siempre y cuando sea un fichero creado por el propio nodo.

3. Sistema

Se trata de gestionar algún aspecto de la propia configuración y una opción para poder salir de la aplicación.



Una vez el usuario ha iniciado la aplicación y pulsa la opción *Sistema*, podrá realizar:

- **3.1.** Configurar: El usuario podrá cambiar parámetros de configuración de la aplicación.
- **3.2.** Salir: Se sale de la aplicación.

4.5 FUNCIONALIDADES

4.5.1 CONECTAR

Requisito:REQ_n5

Caso de uso: 1.1

Tipo de aplicación: Nodo Peer

Como se ha ido comentando en las otras funcionalidades de un *nodo peer*, la aplicación puede trabajar conectada o desconectada al resto de nodos del grupo.

Cuando se abre la aplicación, por defecto, no se está conectado al resto de nodos del grupo.

Para poder conectar, se tendrá que ir a la opción de menú *Conexión → Gestión*.

Al pulsar sobre esta opción de menú se cargará una pantalla común para las opciones de *Conectar/Desconectar* donde se mostrará la siguiente información:

Campo	**Descripción**
Nombre Peer	Nombre del nodo
Identificador peer	Identificador único del peer
Nombre del Grupo	Nombre del grupo de trabajo donde se conectan
Nombre canal Grupo	Nombre del canal (pipe) de grupo que se crea para las conexiones al grupo (Multicast)
Identificador canal Grupo	Identificador único del canal de Grupo
Nombre canal individual	Nombre del canal que se crea para realizar las conexiones uno a uno (Unicast)
Identificador canal individual	Identificador único del canal uno a uno
SuperPeer	IP y Puerto del SuperPeer

Sólo se trata de información para el interés del usuario pero no se podrá editar ninguno de los campos.

En esta pantalla se mostrarán los dos botones de *Conectar* y *Desconectar*.

En la situación en la que estamos, que la aplicación NO está conectada al grupo, el botón *Conectar* estará habilitado y el botón *Desconectar* estará deshabilitado.

A parte, en la parte inferior de la aplicación, se puede ver una pequeña barra donde se indica el estado de la aplicación. En este estado inicial se mostrará que está *Desconectado*.

Una vez el usuario pulsa el botón *Conectar*, la aplicación iniciará los pasos necesarios para conectarse al grupo y empezará con el proceso de comunicación de cambios que se hayan podido realizar en los documentos propiedad y de obtener aquellas actualizaciones que otros integrantes del grupo hayan podido realizar en sus documentos.

Una vez conectado, el botón *Conectar* se deshabilitará y el botón *Desconectar* se habilitará. A parte, la barra de estado indicará que la aplicación está *Conectada*. Durante el proceso de conexión se indicará al usuario el avance del mismo mediante una barra de situación donde se contemplará el % de avance del mismo.

En caso de error o problema durante el proceso de conexión, la solución avisará del hecho sucedido.

A partir de ese momento se trabajará en modo *online* por lo que cualquier cambio que se haga, tanto interno como externo, se verá reflejado en el propio nodo y en el grupo.

4.5.2 DESCONECTAR

Requisito:REQ_n5

Caso de uso: 1.2

Tipo de aplicación: Nodo Peer

Como se ha ido comentando en las otras funcionalidades de un *nodo peer*, la aplicación puede trabajar en modo desconexión.

Para poder desconectar, se tendrá que ir a la opción de menú *Conexión → Gestión*.

Como ya se ha comentado, la pantalla resultante será común para la opción *Conectar* y *Desconectar* (pantalla descrita en el apartado anterior).

Para poder desconectarnos se tiene que estar en modo *online* (conectados al grupo). Por ello, al entrar en esta pantalla el botón *Conectar* tiene que estar desactivado y el botón *Desconectar* activado. A parte, la barra de estado tiene que indicar *Conectado*.

Al pulsar sobre el botón *Desconectar*, la aplicación se desconectará del grupo y pasará a trabajar en modo *off-line*; por lo que cualquier cambio que se pueda producir tanto interno como externo no se verá reflejado en el grupo ni en el propio nodo (en aquellos documento que no sean propiedad del nodo).

Una vez acabado el proceso de desconexión, la barra de estado tendrá que indicar *Desconectado*.

4.5.3 CREAR DOCUMENTOS XML

Requisito:REQ_n1

Caso de uso: 2.1

Tipo de aplicación: Nodo Peer

El fin de la aplicación es poder compartir documentos XML entre las personas de un mismo equipo de trabajo.

Mediante la opción de menú *Ficheros → Nuevo*, el usuario podrá acceder a la operativa de crear nuevo documento XML.

Esta funcionalidad permitirá introducir el texto necesario que formará el documento XML.

Cuando el usuario accede a la aplicación podrá visualizar un listado con todos los documentos con los que el grupo puede trabajar. Se tratan documentos propiedad del propio nodo y documentos propiedad de nodos del grupo.

También dispondrá de un área de trabajo (editor) donde introducir el contenido del documento y donde indicar el nombre del archivo a crear. A parte, dos opciones de trabajo: *Guardar* y *Eliminar*; aunque la opción *Eliminar* estará deshabilitada (ver opción *Listar* para ver el funcionamiento de esta opción).

Para que el usuario pueda crear un documento, en primer lugar introducirá el contenido necesario que formará el xml final, insertará el nombre del fichero xml que quiere crear y pulsará el botón *Guardar*.

Se validará que el usuario haya introducido contenido y que dicho contenido tenga el formato apropiado para crear un documento xml.

Se validará que el usuario haya introducido un nombre válido para el fichero xml. Se verificará que el fichero a crear no exista ya y que tanto el nombre como la extensión sean las apropiadas.

Con esta acción, se creará un nuevo documento con el nombre que haya indicado el usuario y en el directorio definido en el sistema.

A partir de aquí, el documento formará parte de la lista de archivos a compartir con el resto de compañeros (nodos) del grupo. Automáticamente se refrescará la lista de documentos y se podrá visualizar el nuevo documento dado de alta. De igual forma, se limpiará el área de edición y de introducción del nombre del documento para que el usuario pueda volver a crear uno nuevo.

Tener en cuenta que la acción de crear documento XML se podrá realizar tanto si la aplicación está conectada al grupo de nodos como si la aplicación está desconectada.

En caso de estar conectada en el momento de crear el documento, la solución se encargará de informar al resto de compañeros del grupo de la existencia del nuevo documento para activar las directrices necesarias de actualización de documentos al resto de nodos del grupo.

En caso de no estar conectada, una vez se conecte se encargará de informar de los cambios al resto de nodos del grupo.

4.5.4 LISTAR DOCUMENTOS XML

Requisito:REQ_n2

Caso de uso: 2.2

Tipo de aplicación: Nodo Peer

Con anterioridad se ha visto como el usuario puede crear nuevos documentos XML.

Pues si accedemos por la opción de menú *Ficheros → Listar*, el usuario podrá acceder al listado completo de documentos con los que puede trabajar.

Este listado mostrará todos los documentos XML creados por el propio nodo pero también, todos los documentos XML creados por otros nodos que forman parte del grupo.

La información a mostrar en el listado será:

Campo	Descripción
Documento	Nombre del documento
Fecha	Fecha última modificación
Propietario	S/N. Indica si el fichero es propio o de otro nodo del grupo

La lista de documentos sale ordenada por fecha de forma que el documento más reciente sale primero y el más antiguo el último.

Cada vez que el usuario pulse sobre un documento del listado, se cargará el contenido y el nombre del mismo para que pueda ser visualizado.

Esta acción tendrá diferentes alternativas en función de si el documento es propiedad o no del nodo que lo está consultando.

En caso de ser propietario, el usuario podrá editar o eliminar el documento; por ello al cargar la información los botones *Guardar* y *Eliminar* se activarán. En caso de no ser propietario sólo podrá consultar su contenido y las opciones de *Guardar* y *Eliminar* se deshabilitarán.

En cualquier caso, el nombre nunca se podrá editar y la aplicación realizará las mismas validaciones del contenido que realiza cuando se da de alta un documento.

La acción de listar documentos se puede realizar tanto estando conectado como desconectado del resto del grupo.

En caso de estar desconectado, la lista que se mostrará será la foto de la última conexión realizada; por lo que podrá haber información desactualizada.

En caso de estar conectado, la lista será con la información actualizada (foto de momento).

4.5.5 MODIFICAR DOCUMENTOS XML

Requisito:REQ_n3

Caso de uso: 2.3

Tipo de aplicación: Nodo Peer

Ya se conoce que el manejo de documentos que se realizará será únicamente de documentos XML y que un nodo sólo podrá modificar documentos que sean de su propiedad (creados por el mismo nodo).

Para acceder al modo edición de un documento, se tendrá que ir a la opción de menú *Ficheros → Listar* y pulsar sobre uno de los documentos propiedad del nodo; en caso de pulsar sobre un documento propiedad de otro nodo, sólo se podrá visualizar su contenido pero no se podrá ni *guardar* ni *eliminar*.

Al pulsar sobre un documento editable, con independencia que se cargue su contenido en la parte de edición del documento, se habilitan los botones *guardar* y *eliminar*.

El usuario podrá modificar el contenido del fichero y una vez lo quiera guardar, pulsará el botón *guardar* y automáticamente se volcarán los datos en el documento físico y se avisará al resto de nodos de que el documento ha sido modificado.

A la hora de modificar el contenido, la aplicación realizará las mismas validaciones que si de una nueva alta se tratara. A parte, no se

permite modificar el nombre del documento (por ello el campo nombre estará deshabilitado).

A nivel de modificaciones del XML se podrán realizar las siguientes:

- No se permitirá cambiar el nombre del nodo raíz. En tal caso se entiende que se quiere modificar todo el documento por lo que habrá que Eliminar y crear uno nuevo.
- Se podrá modificar los parámetros de cualquier nodo, permitiendo añadir, modificar o eliminar parámetros.
- Se podrá modificar el texto de cualquier nodo, permitiendo añadir, modificar o eliminar el texto.
- Se podrá dar de alta nodo en todo el árbol del xml
- Se podrá eliminar nodos en todo el árbol del xml
- Se podrá modificar nombre de cualquier nodo (siempre y cuando no sea el nodo raíz).
- Como en cualquier xml, se podrán repetir nombres de nodos.

La acción de modificar documento se puede realizar tanto estando conectado como desconectado del resto del grupo.

En caso de estar conectado en el momento de modificar el documento, la solución se encargará de informar al resto de compañeros del grupo que el documento ha sido modificado para activar las directrices necesarias de actualización de documentos al resto de nodos del grupo.

En caso de no estar conectado, una vez se conecte se encargará de informar de los cambios al resto de nodos del grupo.

4.5.6 BORRAR DOCUMENTO XML

Requisito:REQ_n4

Caso de uso: 2.4

Tipo de aplicación: Nodo Peer

Ya se conoce que el manejo de documentos que se realizará será únicamente de documentos XML y que un nodo sólo podrá borrar documentos que sean de su propiedad (creados por el mismo nodo).

Para acceder al modo edición de un documento, se tendrá que ir a la opción de menú *Ficheros → Listar* y pulsar sobre uno de los documentos propiedad del nodo; en caso de pulsar sobre un

documento propiedad de otro nodo, sólo se podrá visualizar su contenido pero no se podrá ni *guardar* ni *eliminar*.

Al pulsar sobre un documento editable, con independencia que se cargue su contenido en la parte de edición del documento, se habilitan los botones *guardar* y *eliminar*.

Si el usuario quiere borrar un documento, tendrá que pulsar sobre el botón *eliminar* y la propia aplicación pedirá confirmación de dicha acción. Una vez aceptada, se borrará el documento (no sólo a nivel lógico sino físico).

Al borrar el documento, se actualizará el listado de documentos para poder seguir trabajando con los documentos existentes.

En caso de estar conectada en el momento de borrar el documento, la solución se encargará de informar al resto de compañeros del grupo que el documento ha sido eliminado para activar las directrices necesarias de actualización de documentos al resto de nodos del grupo.

En caso de no estar conectada, una vez se conecte se encargará de informar de los cambios al resto de nodos del grupo.

4.5.7 CONFIGURAR

Requisito:

Caso de uso: 3.1

Tipo de aplicación: Nodo Peer

Para poder acceder al área de configuración de la aplicación, se tendrá que ir a la opción de menú *Sistema → Configurar*.

Al pulsar sobre esta opción de menú se cargará una pantalla donde el usuario podrá un parámetro de la aplicación:

Campo	Descripción
Directorio de trabajo	Path del directorio donde se guardarán los documentos xml
SuperPeer IP	Dirección IP del SuperPeer
SuperPeer Puerto	Puerto del SuperPeer

Respecto al parámetro *Directorio de* trabajo, por defecto, cuando un usuario utiliza la aplicación por primera vez, el sistema crea un directorio llamado *documentos* donde se guardarán todos los

documentos que se puedan crear tanto por el propio nodo como por el resto de nodos el grupo.

Este directorio se creará en el mismo path donde se ejecuta la aplicación.

Si el usuario quiere utilizar otro directorio (y no el de por defecto), accediendo a esta opción puede cambiarlo.

Este cambio sólo será efectivo para los nuevos documentos que a partir de ese momento se creen (bien por el propio nodo o por el resto de nodos del grupo). Los documentos ya existentes no se moverán al nuevo directorio; permanecerán en el directorio anterior.

De igual forma, los parámetros IP y Puerto del *SuperPeer* se establecen por defecto pero siempre se podrán editar por esta opción. Eso sí, si se cambian cuando el usuario está CONECTADO, no tendrán efecto hasta la siguiente conexión.

4.5.8 SALIR

Requisito:

Caso de uso: 3.2

Tipo de aplicación: Nodo Peer

Para poder salir de la aplicación, se tendrá que ir a la opción de menú *Sistema → Salir*.

Al pulsar sobre esta opción se saldrá de forma automática de la aplicación. Internamente se hará un paso previo que es desconectarse del grupo en caso de estar conectado.

4.6 DIAGRAMAS DE ACTIVIDAD

4.6.1 ACTIVIDAD: NODOS PEER & SUPERPEER

Hasta ahora básicamente se ha hablado de funcionalidades y actores (usuarios) que iban a realizar dichas acciones. Por ello, gran protagonismo lo ha tenido la parte del proyecto que hace referencia a nodos p*eer*. No olvidemos que son estos nodos los que disponen de la interficie para que el usuario pueda interactuar con los demás compañeros del grupo.

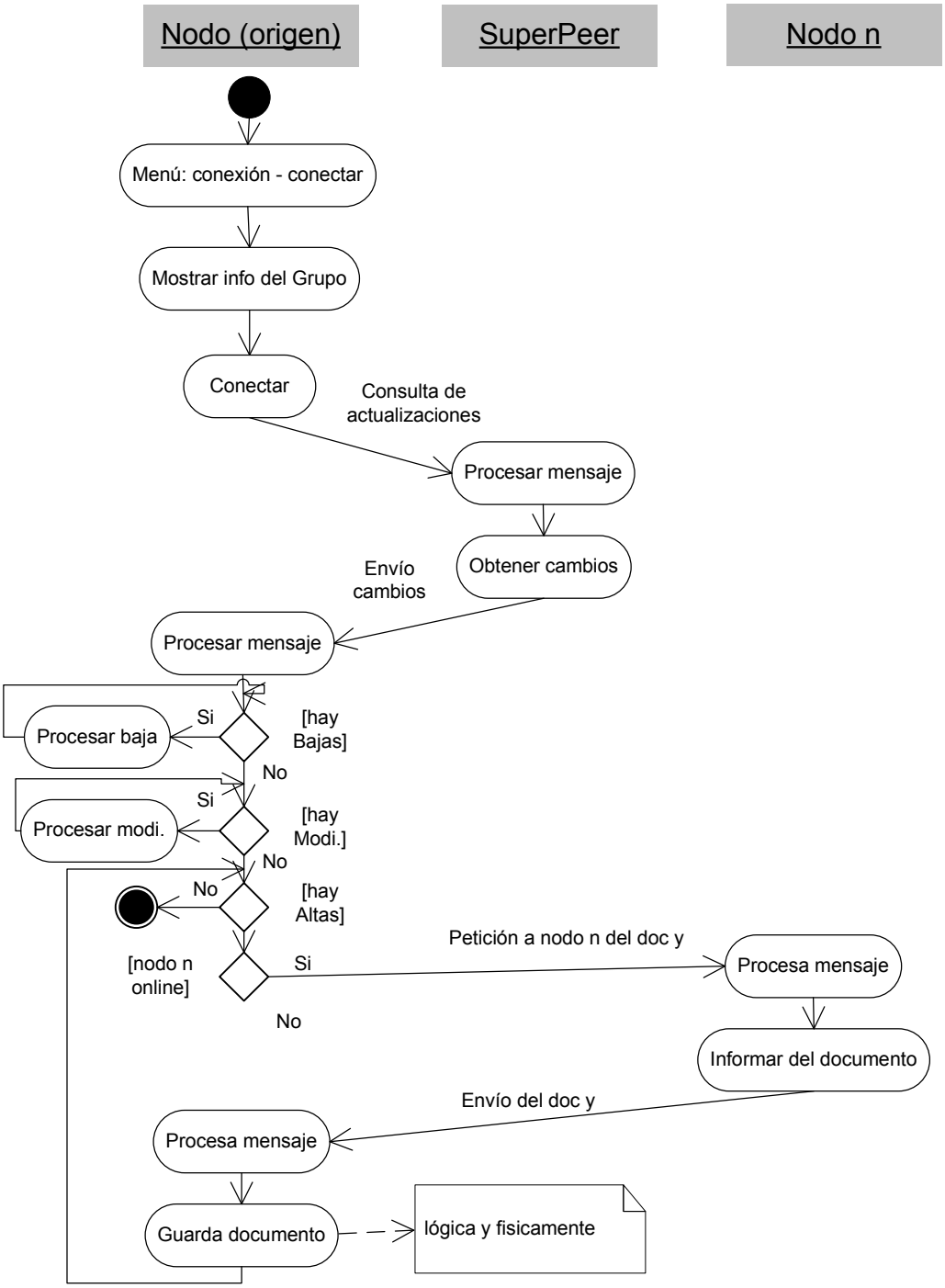
Ahora se hablará de la actividad de estas acciones, y en este momento entrará en juego el otro elemento del proyecto, el nodo s*uperpeer*.

Recordar que este elemento no dispone de una interficie por lo que un usuario no interactúa directamente con él; pero si los nodos p*eer* a la hora de comunicar actualizaciones o de poder saber si un determinado nodo está actualizado.

4.6.2 ACTIVIDAD CONECTAR

Requisito: REQ_a1, REQ_a2, REQ_a3, REQ_a4

La acción de conectar se ha dividido en dos partes. La primera parte la actividad es:



Una vez el usuario ha pulsado el botón *Conectar* la aplicación tiene que informarse de todos aquellos cambios que hayan podido suceder en documentos de otros nodos.

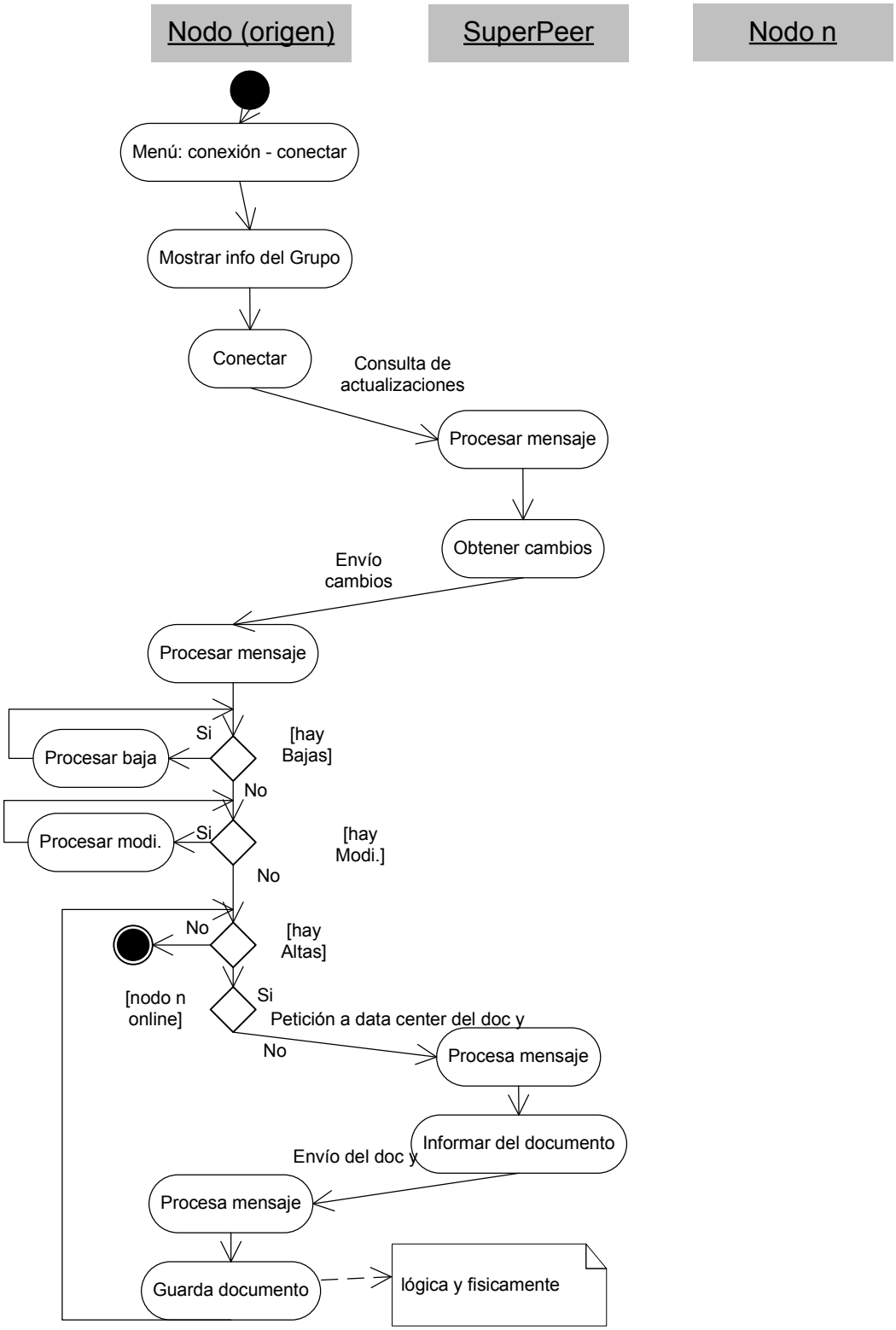
Para ello, envía una petición a *superpeer* para que le informe de todas estas actualizaciones. En este envío, el nodo origen envía todos los documentos no propiedad suya y en qué fecha los tiene actualizados. Con esta información el nodo *superpeer* puede verificar qué cambios ha habido y enviar un único mensaje con toda esta información.

Los cambios que han podido haber serán bajas, con lo que informará del identificador del documento; modificaciones, con lo que informará del documento y modificaciones realizadas (nodos modificados) y altas de nuevos documentos.

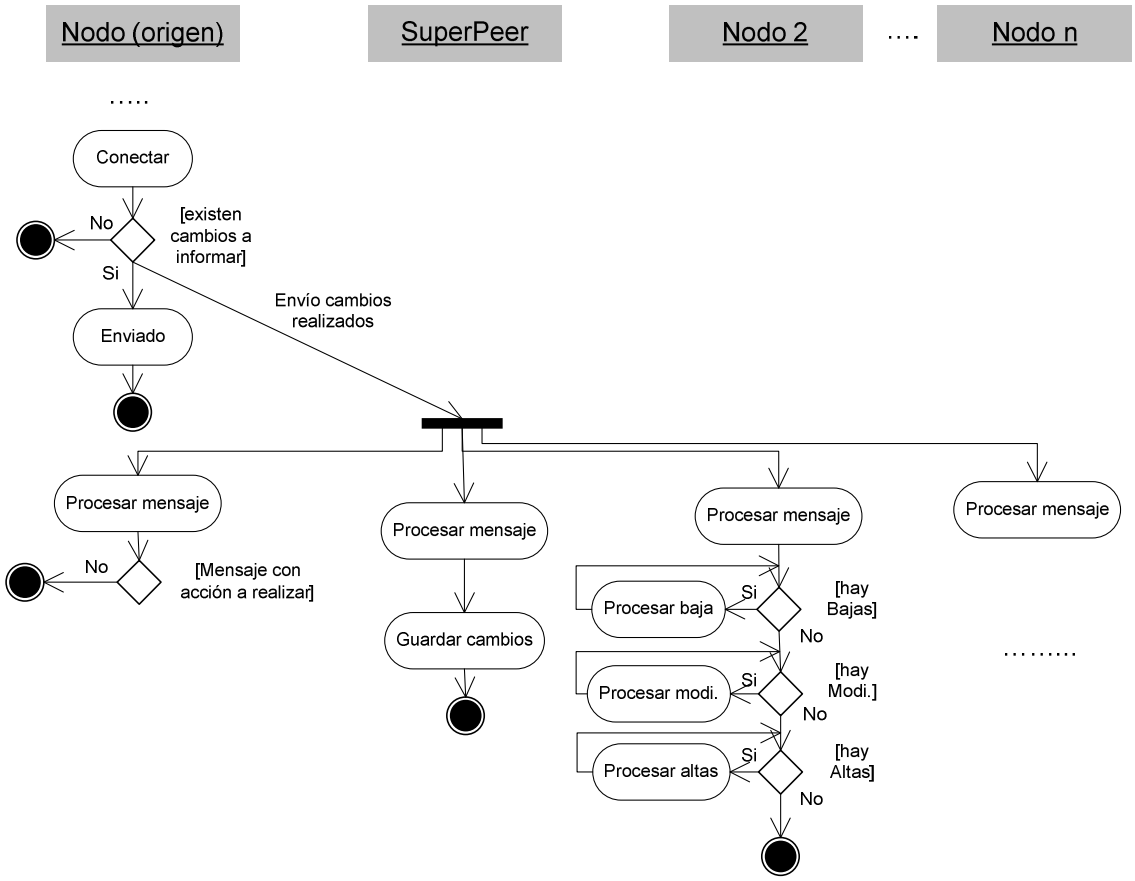
En este último caso se informa del identificador del documento y del propietario (identificador único del nodo creador).

Con toda esta información el nodo que ha realizado la petición procesa todas las bajas, dando de baja aquellos documentos indicados. Todas las modificaciones, aplicando los cambios en los nodos/documentos y por último todas las altas. En este caso, el nodo envía un mensaje al propietario del documento indicando que le envíe el documento que ha creado. Si el nodo propietario esta *online*, le devuelve el mensaje dando el documento xml nuevo (diagrama de actividad indicado con anterioridad).

Pero puede suceder que cuando el nodo pide al nodo propietario el documento, este nodo no esté *online*, por lo que el nodo realizará otra petición al *data center* (*superpeer*) indicando que le dé dicho documento. Esta parte de la actividad está detallada en el siguiente diagrama que es la continuación del anterior completando la salida NO a la pregunta *nodo n online*:



Una vez finalizado este proceso, el nodo tiene que informar al resto del grupo de todos los cambios que haya podido realizar en los documentos de los que es propietario:



Una vez la aplicación recopila todos los cambios que ha realizado mientras estaba sin conexión, envía un mensaje al grupo con todas estas modificaciones.

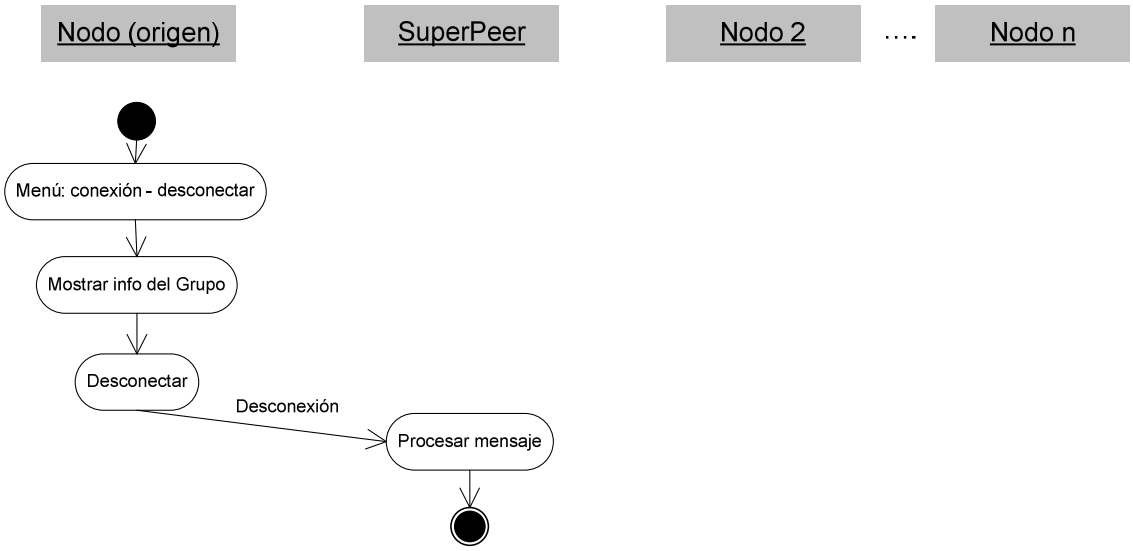
Cuando el mensaje llega al *superpeer*, guarda dichos cambios (altas, bajas y modificaciones) en base de datos y almacena los documentos que son objeto de alta en la unidad de disco. Esto último lo realiza por ser también *data center* del grupo.

Cuando el mensaje llega al resto de nodos *peer* del grupo, procesan el mensaje y realiza los cambios indicados por el nodo origen.

Tener en cuenta que en este mensaje se envía todo, tanto las referencias para las bajas, como los nodos/documentos modificados como los nuevos documentos. No hace falta otra comunicación entre los *peers* destino y el nodo origen para pedir los nuevos documentos.

4.6.3 ACTIVIDAD DESCONECTAR

El siguiente diagrama muestra la actividad cuando un usuario decide desconectarse del grupo (red):



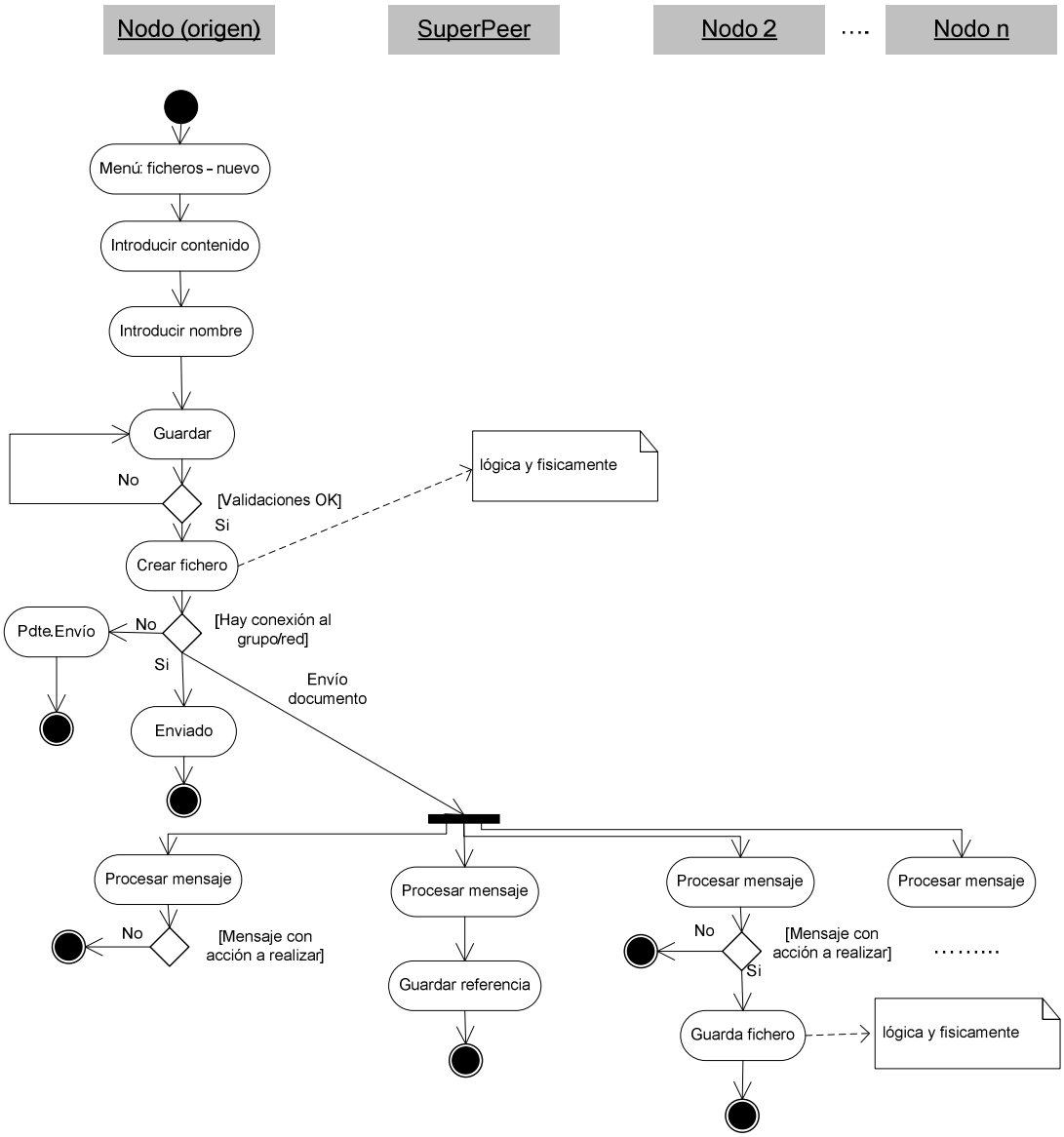
Al pulsar el botón se desconecta la aplicación del grupo.

Esto no significa que se cierre la solución sino que trabaja sin conexión al grupo por lo cualquier cambio que haga el nodo no será enviado al resto (se guardará como cambios pendientes de enviar) y cualquier cambio que se realice en el resto no será informado al nodo origen.

4.6.4 ACTIVIDAD CREAR DOCUMENTO XML

Requisito: REQ_a1, REQ_a3, REQ_a4

El siguiente diagrama de actividad muestra las acciones a realizar cuando un nodo crea un nuevo documento xml:



Nodo origen es el nodo que crea el documento xml. Como se ha podido ver con anterioridad, cualquier nodo (sin contar *SuperPeer*) puede crear xmls e informar al resto de nodos que forman el grupo.

En este caso, una vez creado el documento y guardado en disco local del nodo origen, la aplicación actuará de manera diferente en función de si está conectada al grupo o no lo está.

En caso de no estar conectado al grupo, guardará conforme se ha creado un documento y no se ha informado al grupo.

En caso de estar conectado al grupo, se creará un mensaje que se enviará a todo el grupo, donde indicará que se ha creado un nuevo documento y se informará del documento creado.

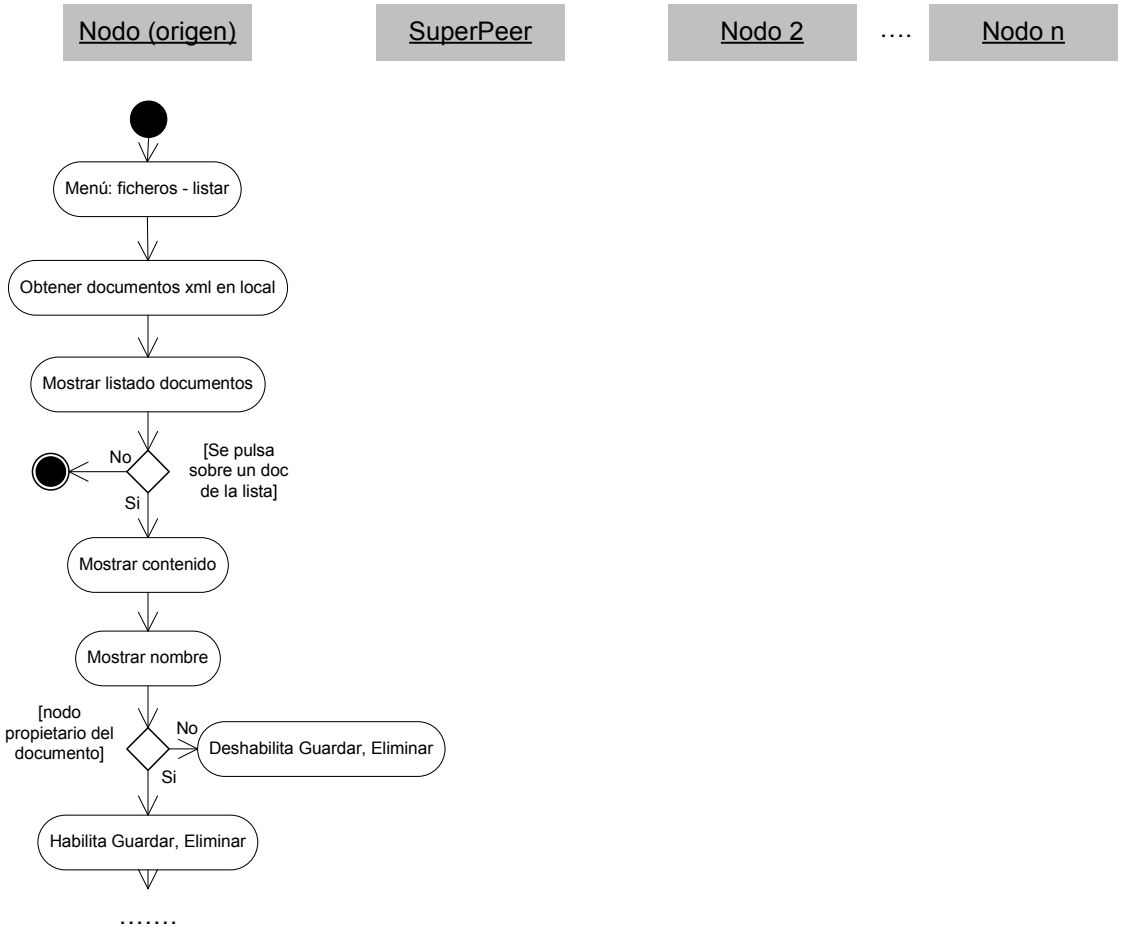
Los nodos tipo *peer* que estén conectados recibirán el mensaje, lo procesarán y guardarán el documento: Lo almacenan en la base de datos y lo guardan en el disco local (lógica como físicamente). El

nodo origen también recibirá el mensaje pero lo descartará por ser él el emisor del mismo.

El nodo *superpeer* procesará el mensaje y guardará la referencia del documento en base de datos para saber quién y qué documento se ha dado de alta. A parte, y por actuar como *data center*, guardará físicamente el documento para poder disponer del mismo en caso de que un nodo lo requiera.

4.6.5 ACTIVIDAD LISTAR DOCUMENTOS XML

El siguiente diagrama de actividad muestra las acciones a realizar cuando un nodo lista los documentos xml de trabajo (propios y del grupo):



La opción de *listar* documentos es independiente al estar o no conectado al grupo (red).

Cuando el usuario accede a la opción de *Listar* dentro del menú de *Ficheros*, el sistema consulta los documentos xml que tiene dados de alta y muestra en forma de lista los nombres de cada uno de ellos.

Con la acción de pulsar sobre cualquiera de ellos, muestra el contenido y el nombre del mismo en el editor de la solución.

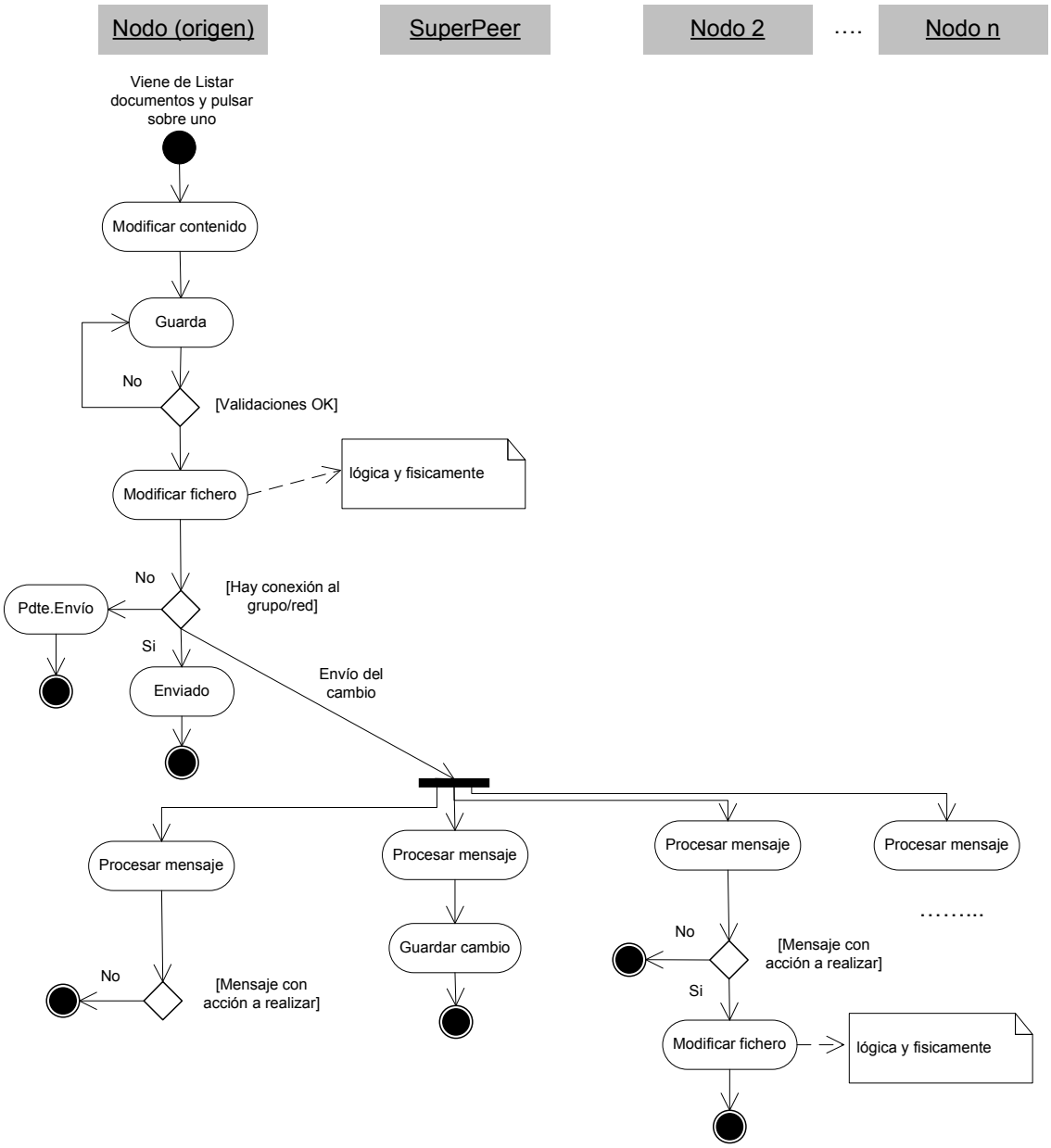
A parte de mostrar el contenido, el sistema verifica si el documento es propiedad del nodo. En caso de no serlo, sólo permite al usuario consultar el contenido, por lo que los botones de *Guardar* y *Eliminar* se deshabilitan. En caso de ser propietario del documento, habilita los botones de *Guardar* y *Eliminar* y permite al usuario modificar el contenido.

Si el usuario pulsase alguna de estas acciones, se enlazaría con la actividad seleccionada.

4.6.6 ACTIVIDAD MODIFICAR DOCUMENTO XML

Requisito: REQ_a1, REQ_a3, REQ_a4

El siguiente diagrama de actividad muestra las acciones a realizar cuando un nodo modifica un documento xml:



Nodo origen es el nodo que modifica el documento xml. Como se ha podido ver con anterioridad, un nodo sólo podrá modificar aquellos documentos creados por él mismo.

Los controles de poder o no modificar un documento se han visto en la actividad *listar documentos xml*.

Una vez modificado el documento el nodo guarda los cambios en el fichero xml original con el orden en el que se detectan en el documento xml y se guarda una referencia de lo(s) cambio(s) realizados (referencia de nodos modificados, fecha y orden).

En caso de no estar conectado al grupo, cuando se conecte enviará dichos cambios.

En caso de estar conectado al grupo, envía estos cambios para que el resto del grupo los tenga.

Al enviarse este mensaje a todos los nodos, cada nodo tendrá que procesar y saber si tiene que hacer algo o no. Únicamente el nodo origen (el que ha modificado el documento) descartará el mensaje a tratarse de un evento creado por él mismo.

El resto de nodos procesarán el mensaje y actualizarán el documento tal cual los cambios enviados por el nodo origen y en el orden indicado.

Por otro lado, el nodo s*uperpeer* se guardará la referencia de los cambios realizados para poder informar a posibles nodos no conectados en ese momento de dichos cambios.

4.6.7 ACTIVIDAD BORRAR DOCUMENTO XML

Requisito: REQ_a1, REQ_a4

El siguiente diagrama de actividad muestra las acciones a realizar cuando un nodo elimina un documento xml:

Nodo origen es el nodo que elimina el documento xml. Como se ha podido ver con anterioridad, un nodo sólo podrá eliminar aquellos documentos creados por él mismo.

Los controles de poder o no eliminar un documento se han visto en la actividad *listar documentos xml*.

Cuando el usuario pulsa sobe el botón *eliminar*, la aplicación pide confirmación de dicha acción. Una vez aceptada, se elimina el documento del disco y de base de datos.

Si la aplicación no está conectada al grupo, se guarda una referencia y al conectarse se informa al grupo de dicha baja.

En caso de estar conectada al grupo, se envía mensaje indicando el documento que se ha dado de baja.

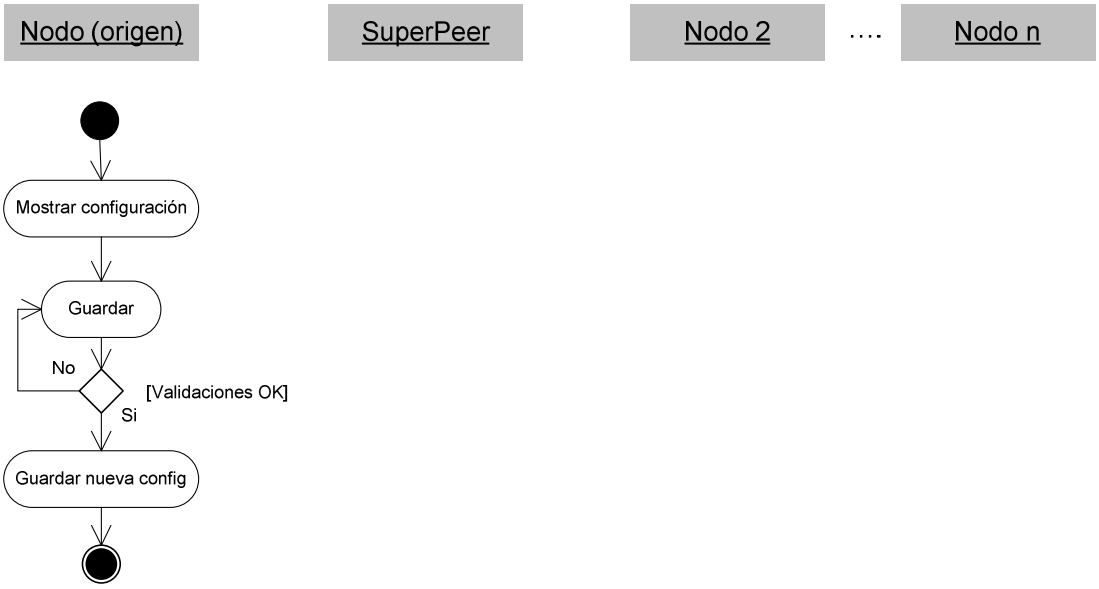
Los nodos *peer* que reciban este mensaje eliminarán el documento tanto física como lógicamente.

El nodo *superpeer* eliminará la referencia de dicho documento y al actuar como *data center* también eliminará el fichero físico.

4.6.8 ACTIVIDAD CONFIGURAR

Requisito:

La actividad de configurar realizará:



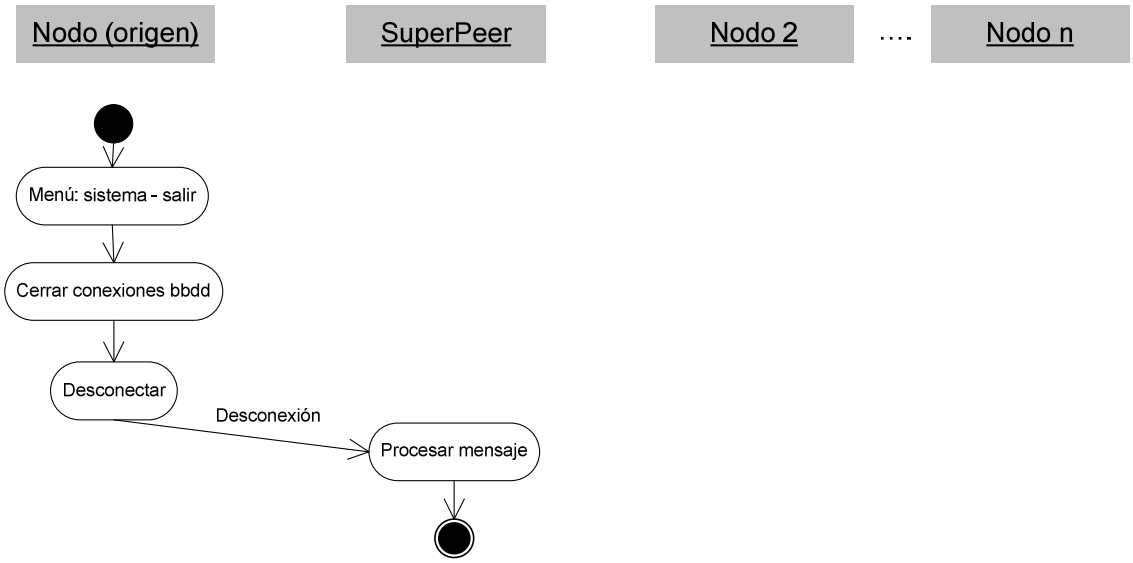
Una vez el usuario ha pulsado el botón *Guardar*, la aplicación realizará las validaciones oportunas en función del parámetro modificado.

En caso de modificar el parámetro que indica el directorio de trabajo, a partir de ese momento cualquier nuevo documento se guardará en esa nueva ubicación.

Si lo modificado es la IP o Puerto del *SuperPeer,* a partir de la siguiente conexión al grupo ya se tendrán en cuenta estos parámetros.

4.6.9 ACTIVIDAD SALIR

El siguiente diagrama muestra la actividad cuando un usuario decide salir de la aplicación:



Antes de cerrarse definitivamente, la aplicación cierra la conexión con la base de datos y realiza la desconexión con el grupo (red).

Con todo esto, la aplicación se cierra definitivamente y el usuario ya no puede trabajar con ella.

4.7 INTERFICIES DE USUARIO

4.7.1 TIPO DE INTERFICIE

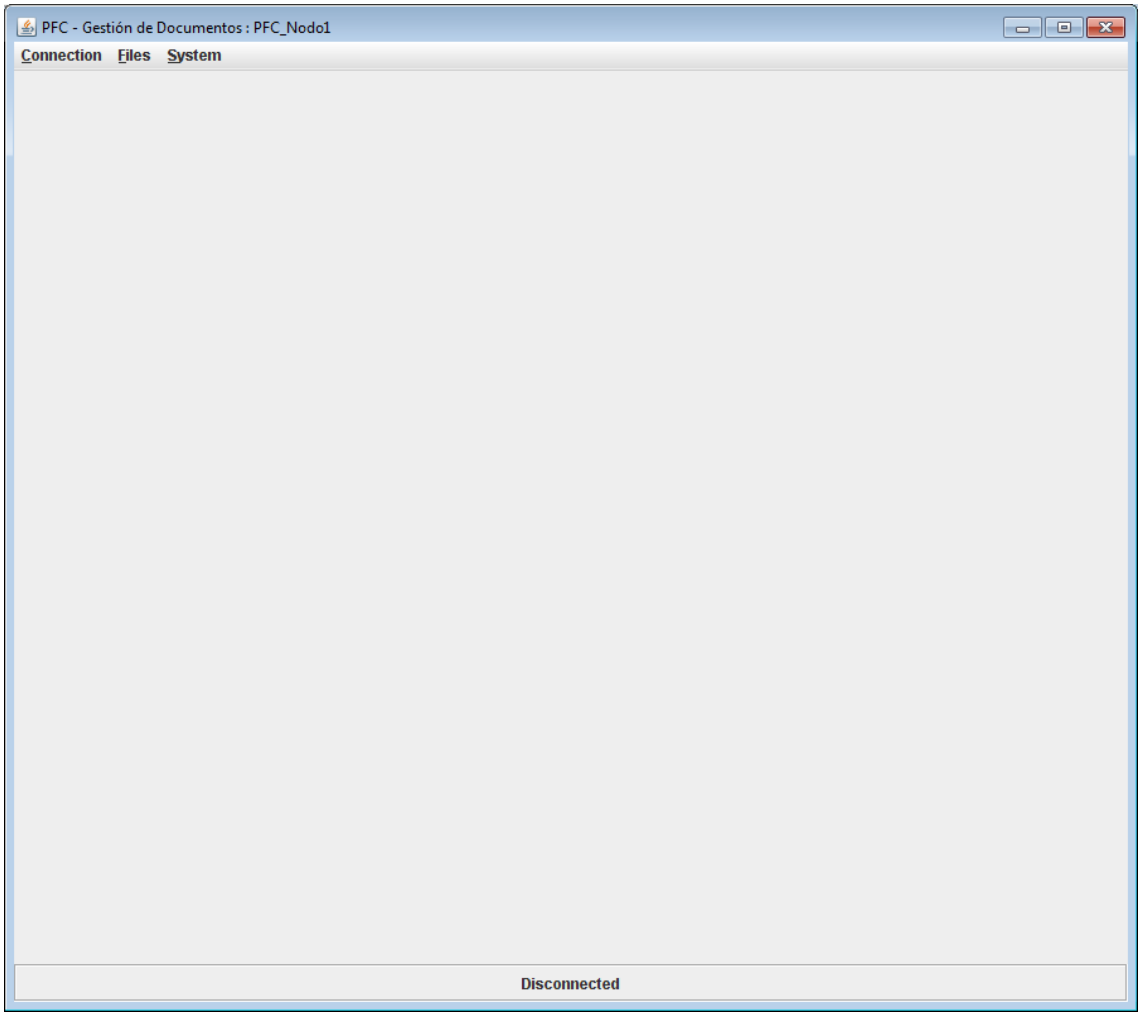
Como se viene indicando a lo largo de todo el documento, el proyecto se basa en dos módulos: Uno sin interficie gráfica (o de usuario) y encargado de orquestar toda la comunicación del grupo; comunicación entre nodos *peer*. Y otro módulo, el propio *peer* que si que dispone de una interficie para que el usuario pueda trabajar con ella.

Se trata de una aplicación pesada de *desktop* basada en una interfice *swing*. La tipología del formato de pantalla es un estilo *look and feel* clásico y el diseño y formato de las pantallas son muy básicas y sencillas de entender. El idioma utilizado es *Inglés*.

El trabajo es intuitivo, basándose toda su operativa en un menú principal y al tratarse de una aplicación tipo Windows, el manejo del marco principal es muy sencillo.

4.7.2 MARCO PRINCIPAL

Cuando se arranca la aplicación lo primero que se carga es el marco principal:



El marco principal se divide en cuatro áreas:

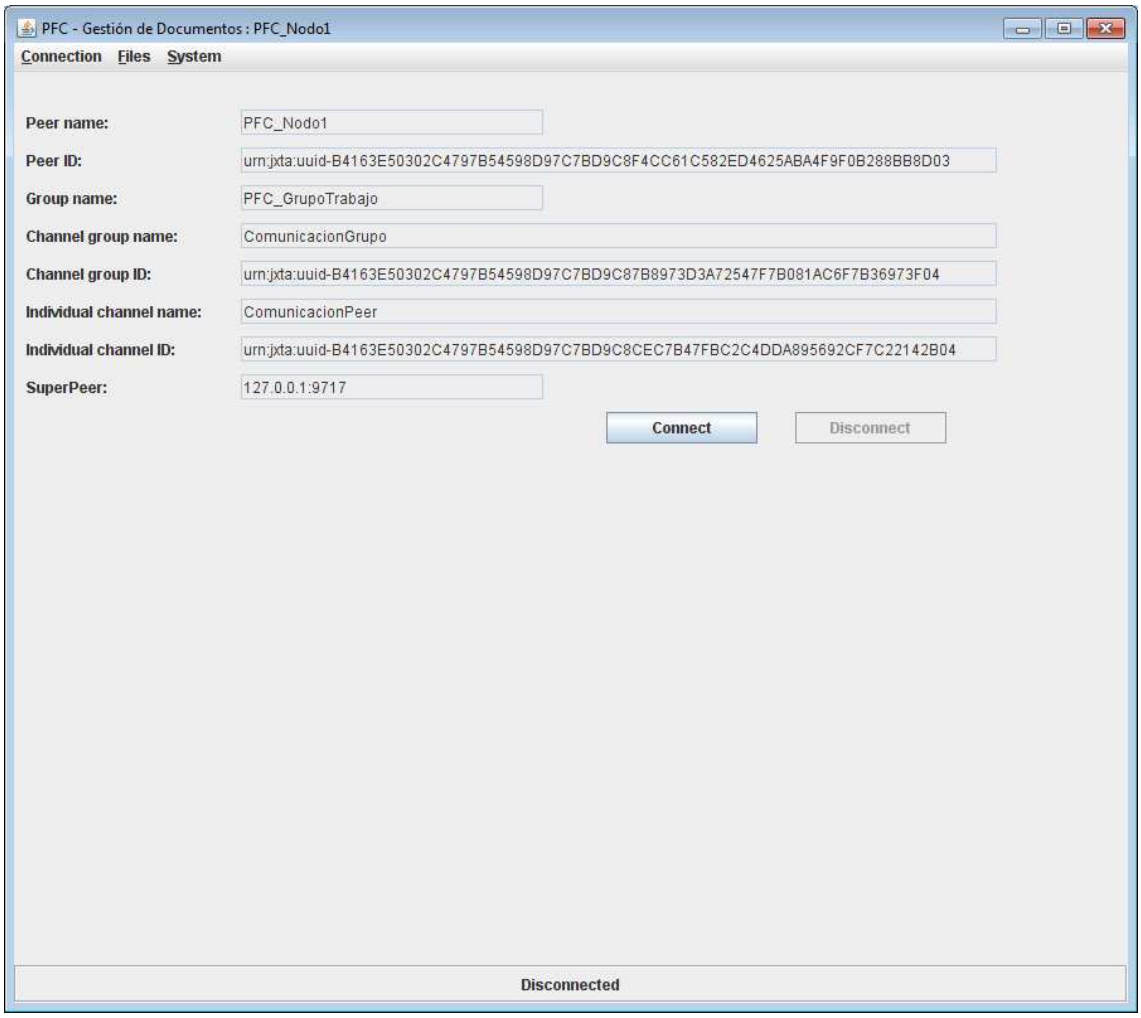
- Barra superior: Tipo *Windows* donde se puede observar el nombre de la aplicación seguido del nombre de nodo conectado. En el ejemplo indicado: PFC – Gestión de documentos : PFC_Nodo1.

  En el extremo derecho las opciones para minimizar, maximizar y cerrar aplicación.

  Esta barra siempre estará visible.
- Menú: El área de menú es un área que también siempre estará visible y desde donde el usuario podrá acceder a las distintas opciones de la aplicación.
- Área de trabajo: Área donde se van cargando las distintas pantallas en función de la opción de menú seleccionada. Es el área central y más grande del marco principal.
- Barra de estado: Barra inferior donde se indica el estado de conexión/desconexión al grupo. Esta barra siempre está visible.

4.7.3 OPCIÓN CONEXIÓN – GESTIONAR

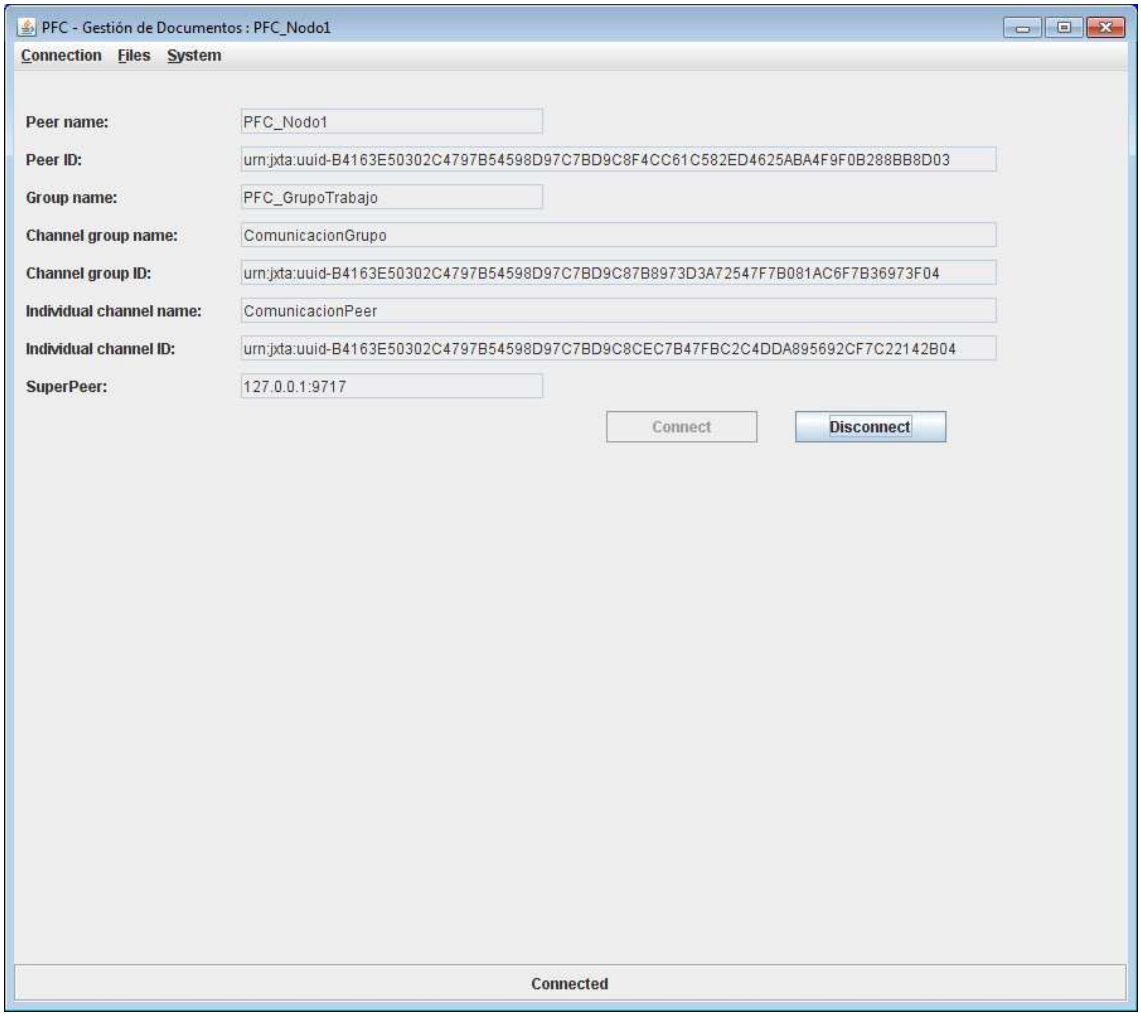
Cuando el usuario accede a esta opción de menú verá la información de conexión al grupo:



Se puede observar que lo único que ha cambiado ha sido el área de trabajo.

Las opciones que tendrá disponible el usuario serán las de *conectar/desconectar* al grupo de trabajo. En este caso, y tal y como se puede observar, la aplicación aún no está conectada al grupo por lo que el botón *desconectar* está desactivado y el estado en *Desconectado*.

Al conectarnos al grupo, la pantalla mostrará:

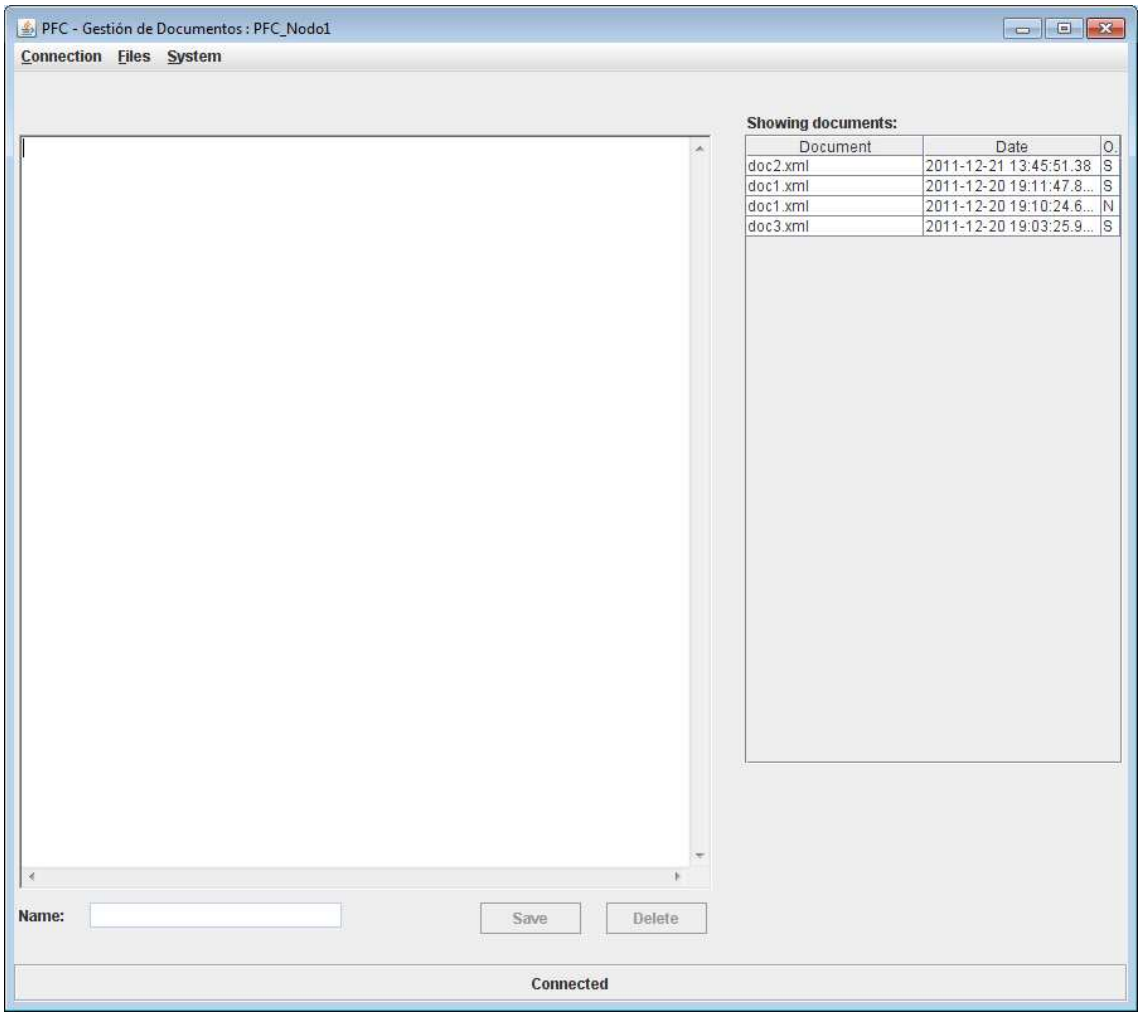


Se puede observar que ahora el estado es *Conectado* y el botón *Conectar* se ha deshabilitado y habilitado el *Desconectar*.

Durante el proceso de conexión la aplicación informará al usuario del grado de avance del mismo:

4.7.4 OPCIÓN FICHEROS – LISTAR

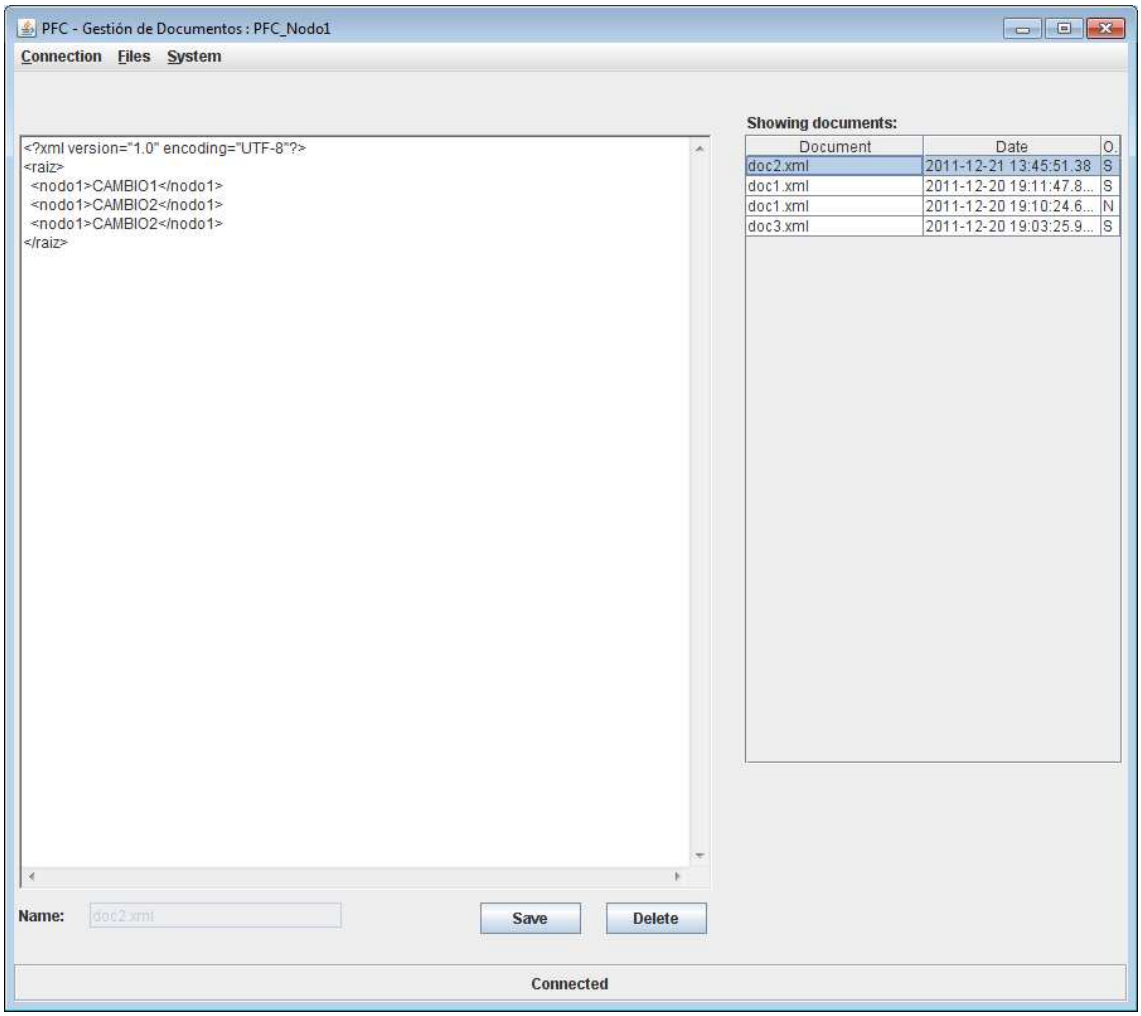
Esta pantalla muestra el listado de todos los documentos disponibles, tanto propiedad como no del nodo.



El área de trabajo está dividida en dos partes:

- Listado de documentos: Se muestran los documentos del grupo. Se diferencia el nombre, fecha de actualización y si es propiedad o no del propio nodo
- Editor: Formado por un área de texto donde al seleccionar un documento se carga el contenido y el usuario puede modificarlo. Por otro lado, cuando se crea uno nuevo, el usuario puede introducir todo el contenido del xml.
- Nombre: Donde se indicará el nombre del documento a editar o el nombre del documento a dar de alta.
- Acciones: Botones de *Guardar* y *Eliminar*.

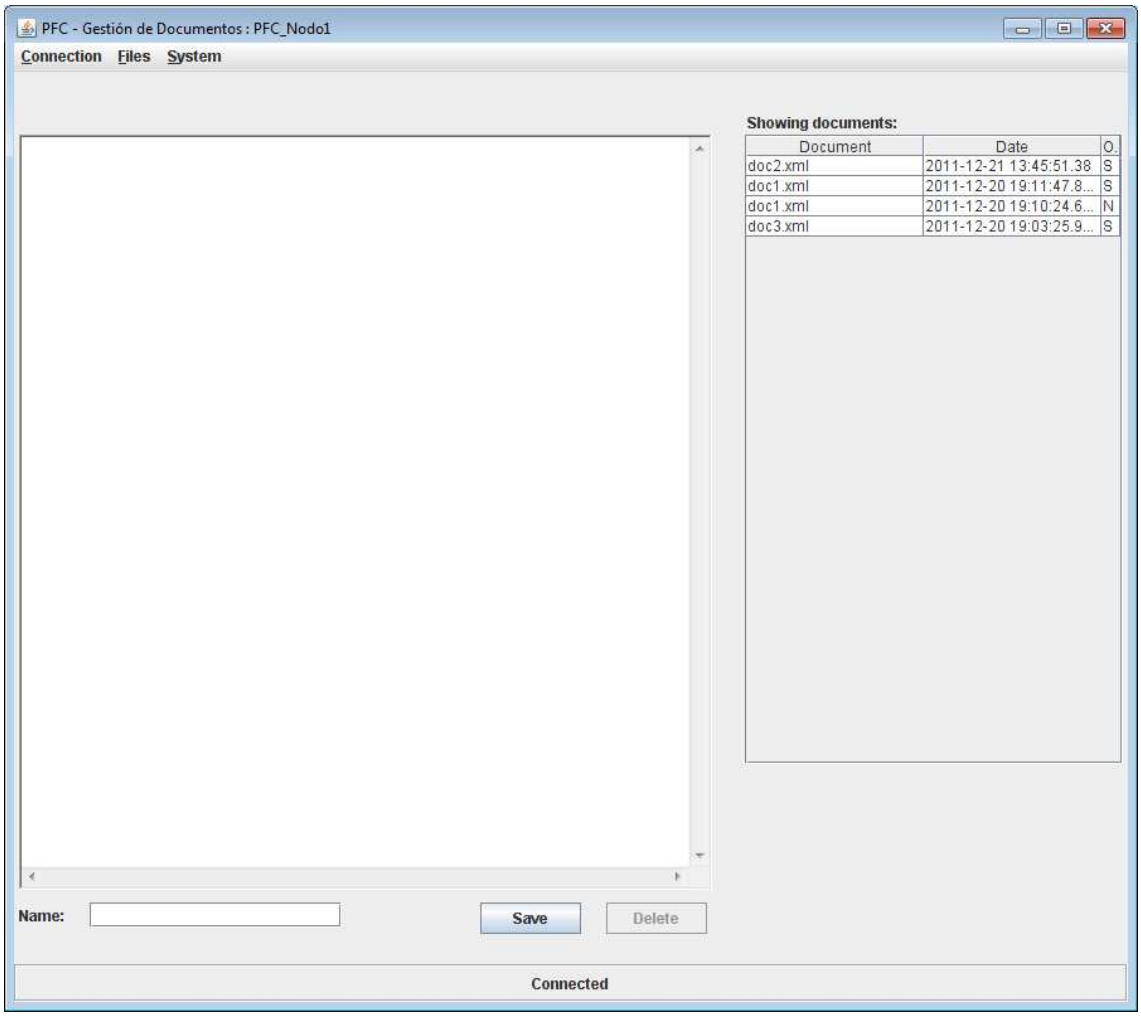
Si se pulsa sobre uno de los documentos del listado:



Se carga el contenido del mismo y se ajustan todas las opciones para poder modificar y eliminar el documento.

4.7.5 OPCIÓN FICHEROS – NUEVO

A nivel de pantalla es la misma que al acceder a la opción de listar con la salvedad que la configuración de las opciones es distinta:

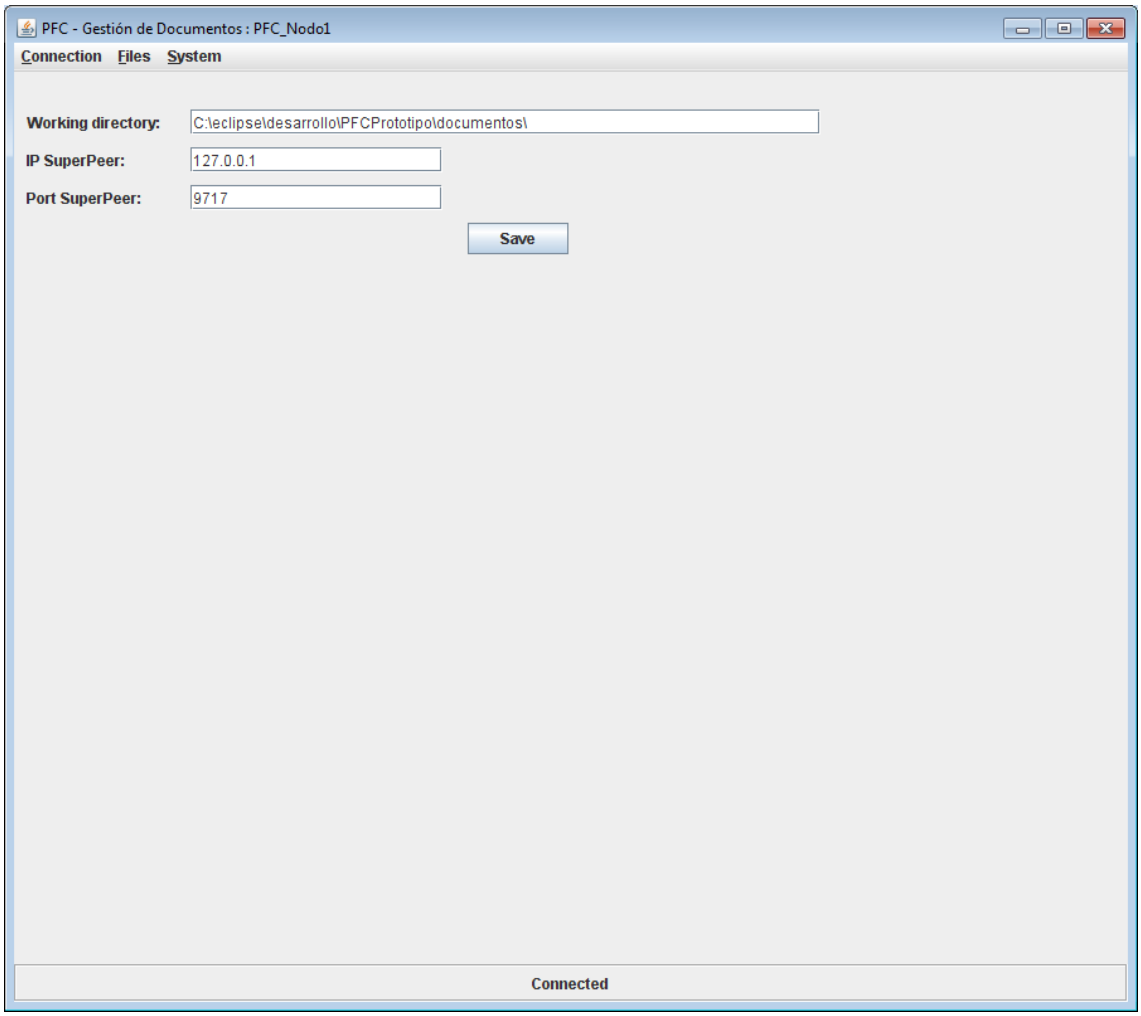


Se habilita tanto el área de texto como el nombre y la opción de *Guardar*.

Cuando el usuario da de alta un nuevo documento, se limpia el área de texto y nombre y se refresca el listado de documentos.

4.7.6 OPCIÓN SISTEMA – CONFIGURAR

Esta pantalla muestra los parámetros de configuración disponibles en la aplicación:



El usuario podrá modificar el directorio de trabajo, IP o Puerto de *SuperPeer* y al pulsar *Guardar* se guardará la nueva configuración.

4.7.7 OPCIÓN SISTEMA – SALIR

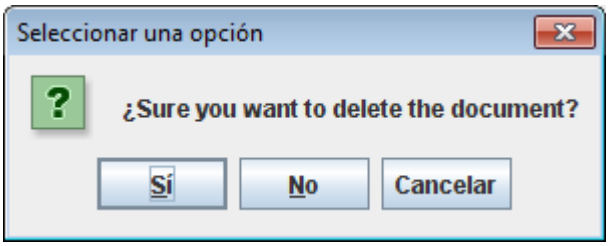
Esta opción de menú no dispone de pantalla. Directamente sale de la aplicación.

4.7.8 AVISOS

La aplicación mostrará una serie de avisos cuando el usuario realice alguna acción no correcta; cuando no se pueda llevar a cabo una determinada acción o simplemente para resaltar o confirmar un determinado mensaje.

Habrá dos tipologías de pantallas de aviso:

- Las de confirmación: Se requiere que el usuario acepte o rechace un determinado mensaje:



- Las de aviso: Únicamente se informa al usuario:

4.8 DISEÑO DE LA BASE DE DATOS

Aunque es cierto que la solución necesitará de un repositorio de datos para almacenar información de documentos, no se trata de una aplicación que necesite de un amplio soporte de base de datos.

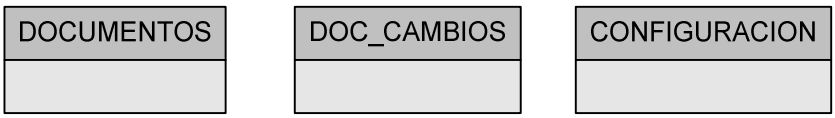
Podemos decir que aunque la base de datos es importante, no es el elemento a resaltar del proyecto y la definición de la misma es muy reducida.

Se sabe que aunque la parte más importante del proyecto es crear un entorno de trabajo colaborativo y establecer las comunicaciones y protocolos necesarios para la compartición de los documentos, se trabajará con un censo de documentos que serán los que los diferentes elementos del grupo compartirán.

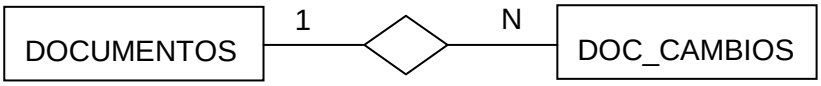
De estos documentos es muy importante saber qué versión del mismo se dispone. Por ello, es vital conocer la fecha de la última actualización. A parte, censar las modificaciones que se hagan en un documento (se trabaja a nivel del nodo del xml) para informar al resto del grupo de dichos cambios (tener en cuenta que los cambios se pueden realizar estando o no conectado a la red).

Visto todo esto, la definición de la base de datos se basa en tres entidades; una que representa el censo de documentos existentes. Otra que identifica los cambios realizados en los documentos en los

que el nodo es propietario. Y una última entidad que guarda la configuración del sistema.



Si plasmamos las relaciones entre tablas (Modelo entidad-relación):



La Entidad CONFIGURACIÓN no tiene relación con ninguna tabla. Únicamente guarda la configuración del propio sistema.

Aunque la solución final dispone de dos elementos, Nodos tipo *peer* y tipo *superpeer*, la definición de la base de datos será la misma; teniendo que administrar las entidades: DOCUMENTOS, DOC_CAMBIOS, CONFIGURACION. Tener en cuenta que el nodo tipo *superpeer* también hace las funciones de *data center* por ello también necesita disponer de la estructura necesaria para almacenar referencia y documentos físicos.

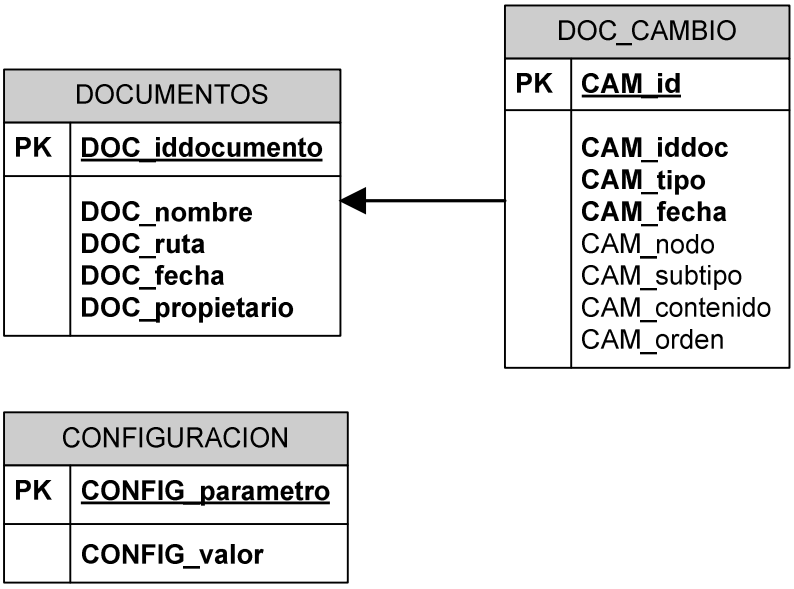
El objetivo de un nodo *superpeer* es conocer el censo de documentos y los cambios que puedan sufrir (por actuar como *superpeer*). Y disponer de las altas de documentos (físicamente) por sus funciones de *data center*.

Por otro lado, el nodo *peer* dispondrá del censo de documentos existentes en el grupo y el censo de cambios que realiza en los documentos de su propiedad mientras ha estado desconectado. En este censo de cambios almacenará el tipo de cambio, la referencia al nodo cambiado y el orden en que se han realizado los cambios así como el contenido del nodo modificado (en caso de baja de un nodo, sólo la referencia).

Tanto el *superpeer* como el *peer*, almacenarán en CONFIGURACIÓN todos los puntos necesarios para el correcto funcionamiento de la aplicación.

4.8.1 DISEÑO LÓGICO

Se define diseño lógico único para los dos elementos de la solución: Nodos tipo *peer* y *superpeer*.



Conocemos que cada nodo puede crear los documentos xml que quiera y que todos ellos los compartirá con el resto de miembros del grupo.

La identificación del documento se realizará mediante el campo *DOC_iddocumento* y se formará mediante:

Id_unico_propietario+ _ + DOC_nombre (sin extensión)

Indicar que *DOC_*propietario contiene el identificador único del nodo que crea el documento. De esta forma se evita que otro nodo haya podido crear un documento con el mismo nombre.

Detalle tabla DOCUMENTOS:

Campo	Tipología	Obligatorio	Descripción
DOC_iddocumento	VARCHAR(111)	Si	Identificador único del documento. Formado por: id_unico_propietario +"_"+DOC_nombre (sin extensión)
DOC_propietario	VARCHAR(80)	Si	Identificador único del

			nodo propietario
DOC_nombre	VARCHAR(35)	Si	Nombre del documento
DOC_ruta	VARCHAR(300)	Si	Ruta físico donde está almacenado el documento
DOC_fecha	TIMESTAMP	Si	Fecha última actualización

Detalle tabla DOC_CAMBIOS:

Campo	Tipología	Obligatorio	Descripción
CAM_id	VARCHAR(63)	Si	Identificador único del cambio. Formado por: nodo +timestamp
CAM_iddoc	VARCHAR(111)	Si	Identificador del documento sobre el que se ha realizado el cambio
CAM_tipo	VARCHAR(1)	Si	Indica el tipo de modificación. Valores: A, B, M
CAM_fecha	TIMESTAMP	Si	Fecha del cambio
CAM_nodo	VARCHAR(50)	No	Path del nodo modificado. Sólo se informa si CAM_tipo = M
CAM_subtipo	VARCHAR(1)	No	Indica el subtipo del cambio. Valores: A, B, M. Sólo se informa si CAM_TIPO = M
CAM_contenido	VARCHAR(1000)	No	Contenido del nodo modificado. Sólo se informa si CAM_TIPO = M y CAM_SUBTIPO = M
CAM_orden	INTEGER	No	Indica el orden en que se han detectado los cambios realizados en el XML

Como se ha visto con anterioridad, el objetivo de esta tabla es guardar todos aquellos cambios que un nodo tipo *peer* pueda realizar en sus documentos mientras éste está desconectado del grupo; y que el *superpeer* almacene todos los cambios que vayan notificando los distintos nodos del grupo.

En la tabla DOC_CAMBIOS se almacenará la información de la siguiente forma:

Alta de un documento (sólo nodo *peer*):

Campo	**Valor**
CAM_id	TIMESTAMP
CAM_iddoc	Identificar del documento dado de alta
CAM_tipo	A
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	Vacío (null)
CAM_subtipo	Vacío (null)
CAM_contenido	Vacío (null)
CAM_orden	0

Este cambio no es almacenado por *superpeer* ya que el control de altas y bajas se realiza de forma directa con la tabla DOCUMENTOS.

Baja de un documento (sólo nodo *peer*):

Campo	**Valor**
CAM_id	TIMESTAMP
CAM_iddoc	Identificar del documento dado de alta
CAM_tipo	B
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	Vacío (null)
CAM_subtipo	Vacío (null)
CAM_contenido	Vacío (null)
CAM_orden	0

Este cambio no es almacenado por *superpeer* ya que el control de altas y bajas se realiza de forma directa con la tabla DOCUMENTOS.

Modificar documento. Alta de un nodo

Nombre del documento: *doc2.xml*

Nombre del nodo:*nodo2*

Contenido del nodo: *<nodo2>cambio 1</nodo2>*

Campo	Valor
CAM_id	/raiz/nodo2[1]1324639676164
CAM_iddoc	B4163E50302C4797B54598D97C7BD9C8F4CC61C582ED4625ABA4F9F0B288BB8D03_doc2
CAM_tipo	M
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	/raiz/nodo2[1]
CAM_subtipo	A
CAM_contenido	<nodo2>cambio 1</nodo2>
CAM_orden	1

Se puede observar que el nodo dado de alta es el primer nodo con nombre *nodo2* (nodo2[1]). Esto se realiza para poder trabajar con múltiples nodos con mismo nombre.

Modificar documento. Baja de un nodo

Nombre del documento: *doc2.xml*

Nombre del nodo:*nodo1[3]*

Campo	Valor
CAM_id	/raiz/nodo1[3]1324641337380
CAM_iddoc	B4163E50302C4797B54598D97C7BD9C8F4CC61C582ED4625ABA4F9F0B288BB8D03_doc2
CAM_tipo	M
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	/raiz/nodo1[3]
CAM_subtipo	B
CAM_contenido	Vacío (null)
CAM_orden	1

Se puede observar que el nodo dado de baja es el tercer nodo con nombre *nodo1* (nodo1[3]).

Modificar documento. Modificar nodo, cambio texto

Nombre del documento: *doc2.xml*

Nombre del nodo:*nodo2[1]*

Nuevo contenido nodo: <nodo2>PRUEBAS</nodo2>

Campo	Valor

CAM_id	/raiz/nodo2[1]1324641544593
CAM_iddoc	B4163E50302C4797B54598D97C7BD9C8F4CC61C582ED4625ABA4F9F0B288BB8D03_doc2
CAM_tipo	M
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	/raiz/nodo2[1]
CAM_subtipo	M
CAM_contenido	<nodo2>PRUEBAS</nodo2>
CAM_orden	1

Se modificar nodo2[1] cambiando el texto. En este caso se obtiene todo el nodo modificado. Si este nodo tuviese hijos, igualmente se cogería todo el contenido del nodo con sus hijos incluidos.

Modificar documento. Modificar nodo, cambio parámetros

Nombre del documento: *doc2.xml*

Nombre del nodo:*nodo2[1]*

Nuevo contenido nodo: <nodo2 id="1">PRUEBAS</nodo2>

Campo	**Valor**
CAM_id	/raiz/nodo2[1]1324641759578
CAM_iddoc	B4163E50302C4797B54598D97C7BD9C8F4CC61C582ED4625ABA4F9F0B288BB8D03_doc2
CAM_tipo	M
CAM_fecha	AAAA-MM-DD hh:mm:ss.mmm
CAM_nodo	/raiz/nodo2[1]
CAM_subtipo	M
CAM_contenido	<nodo2 id="1">PRUEBAS</nodo2>
CAM_orden	1

Se modificar nodo2[1] cambiando parámetros (en este caso se da de alta uno). Se obtiene todo el nodo modificado. Si este nodo tuviese hijos, igualmente se cogería todo el contenido del nodo con sus hijos incluidos.

Detalle tabla CONFIGURACIÓN:

Campo	**Tipología**	**Obligatorio**	**Descripción**
CONFIG_parametro	VARCHAR(20)	Si	Identificador del parámetro de

			configuración
CONFIG_VALOR	VARCHAR(300)	Si	Valor del parámetro

4.9 DISEÑO DEL PLAN DE PRUEBAS

Definición del plan de pruebas:

	Caso de uso:	**1.1 Conectar**	
	Grupo de pruebas:	**Conectar**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Conexión – Gestión*	Se tiene que cargar una pantalla donde se mostrará la siguiente información: Nombre peer, Identificador del peer, Nombre del grupo, nombre del canal Multicast, Identificador único del canal grupo, Nombre del canal Unicast, identificador del canal Unicast, información del SuperPeer (IP : puerto)	
F2	Botones Conectar y Desconectar. La aplicación está en modo DESCONECTADO	El botón *Conectar* tiene que estar activo y el botón *Desconectar* tiene que estar desactivado. En la barra de estado tiene que indicar: DESCONECTADO	
F3	Pulsar *Conectar* sin haber realizados cambios en los documento mientras ha estado desconectado y sin que otro *peer haya realizado cambios en sus documentos.*	El nodo tiene que poder conectarse al *superpeer* y al grupo de trabajo y no tiene que sufrir ningún cambio en los documentos. La barra de estado tiene que cambiar a CONECTADO y el botón *Conectar* se tiene que deshabilitar y habilitar el botón *Desconectar*	
F4	Pulsar *Conectar* habiendo hecho cambios en los documentos mientras ha estado desconectado y sin que otro *peer haya realizado cambios en sus documentos.*	El nodo tiene que poder conectarse al *superpeer* y al grupo de trabajo y ha tenido que transmitir todos los cambios realizados en los documentos; por lo que los otros *peer* y *superpeer* tienen que ver reflejados dichos cambios. La barra de estado tiene que cambiar a CONECTADO y el botón *Conectar* se tiene que deshabilitar y habilitar el botón *Desconectar*	
F5	Pulsar *Conectar* sin haber realizados cambios en los documento mientras ha estado desconectado y realizar cambios en los documentos de otro *peer.*	El nodo tiene que poder conectarse al *superpeer* y al grupo de trabajo y tiene que ver reflejados en los documentos del *peer* que ha hecho cambios, las modificaciones realizadas. La barra de estado tiene que cambiar a CONECTADO y el botón *Conectar* se tiene que deshabilitar y habilitar el botón *Desconectar*	
	Caso de uso:	**1.1 Conectar**	
	Grupo de pruebas:	**Conectar**	

	Tipo de prueba:	**Técnica**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
T1	Pulsar *Conectar* sin haber arrancado el nodo *superpeer*	El nodo no tiene que poder conectarse al *superpeer* y debe informar de tal hecho. La barra de estado tiene que mantener el estado DESCONECTADO y el botón *Conectar* tiene que estar activo y el *Desconectar* deshabilitado.	
T2	Modificar la IP o puerto del *superpeer* y pulsar el botón *conectar*	El nodo no tiene que poder conectarse al *superpeer* y debe informar de tal hecho. La barra de estado tiene que mantener el estado DESCONECTADO y el botón *Conectar* tiene que estar activo y el *Desconectar* deshabilitado.	

	Caso de uso:	**1.2 Desconectar**	
	Grupo de pruebas:	**Desconectar**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Conexión – Gestión* con la aplicación en estado *CONECTADA*	Se tiene que cargar una pantalla donde se mostrará la siguiente información: Nombre peer, Identificador del peer, Nombre del grupo, nombre del canal Multicast, Identificador único del canal grupo, Nombre del canal Unicast, identificador del canal Unicast, información del *superpeer* Ip : puerto). A parte, el botón *Conectar* tiene que estar deshabilitado y el *desconectar* habilitado	
F2	Pulsar *Desconectar*.	La aplicación se tiene que desconectar del grupo. La barra de estado tiene que indicar DESCONECTADO y el botón *Conectar* se ha tenido que activar y el botón *Desconectar* se ha tenido que desactivar.	
F3	Una vez realizada la prueba **F2** del caso de uso 1.2, volver a pulsar el botón *Conectar*.	El nodo tiene que poder conectarse al *superpeer* y al grupo de trabajo (en caso de cambios se habrán enviado o habrán sido recibidos). La barra de estado tiene que cambiar a CONECTADO y el botón *Conectar* se tiene que deshabilitar y habilitar el botón *Desconectar*	

	Caso de uso:	**2.1 Crear documento XML**	
	Grupo de pruebas:	**Crear documento XML**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Ficheros – Nuevo*.	Se tiene que cargar en el área de trabajo una pantalla donde en la parte derecha tienen que salir el listado de documentos del grupo (tanto los del propio *peer* como los del resto de *peers* del grupo). En la parte central un editor de texto y en la parte inferior un espacio para introducir el	

		nombre del fichero y dos botones. El botón *Guardar* estará habilitado y el *Eliminar* estará deshabilitado	
F2	Con la pantalla cargada y sin introducir ningún dato, pulsar botón *Guardar*	La aplicación tiene que avisar que no ha introducido el nombre del documento.	
F3	Con la pantalla cargada sólo introducir el nombre del fichero pero sin extensión xml; pulsar botón *Guardar*	La aplicación tiene que informar que el nombre del documento o extensión no es correcta.	
F4	Introduzca un nombre de fichero correcto (doc1.xml) pero no inserte el contenido. Pulse *Guardar*.	La aplicación le tiene que avisar que falta por introducir el contenido del fichero.	
F5	Introduzca un nombre de fichero correcto (doc3.xml), introduzca contenido pero que no sea un xml correcto.	La aplicación le tiene que informar que el contenido del fichero no es el correcto.	
F6	Introducir un nombre de fichero correcto (doc3.xml) y un contenido que sea xml y pulsar botón *Guardar*. A parte, la aplicación tiene que estar en modo DESCONECTADO.	La aplicación tiene que crear un nuevo documento con nombre y contenido indicado. En el directorio de trabajo se ha tenido que crear el fichero y tiene que existir referencia en base de datos. El listado de documento se habrá actualizado con el nuevo documento y tanto el cuadro de nombre como el editor de texto se han tenido que limpiar. El botón *Guardar* seguirá activado y el *Eliminar* desactivado. Al estar DESCONECTADO, no se habrá enviado el alta al grupo y se habrá dado de alta un registro indicando que hay un cambio (alta) por notificar.	
F7	Introducir un nombre de fichero correcto (doc4.xml) y un contenido que sea xml y pulsar botón *Guardar*. A parte, la aplicación tiene que estar en modo CONECTADO.	La aplicación tiene que crear un nuevo documento con nombre y contenido indicado. En el directorio de trabajo se ha tenido que crear el fichero y tiene que existir referencia en base de datos. El listado de documento se habrá actualizado con el nuevo documento y tanto el cuadro de nombre como el editor de texto se han tenido que limpiar. El botón *Guardar* seguirá activado y el *Eliminar* desactivado. Al estar CONECTADO se habrá enviado el alta al grupo y todos los miembros del grupo que estuviesen conectados tienen que ver el nuevo documento.	
F8	Introducir un nombre de fichero que ya exista (doc1.xml) y pulsar botón *Guardar*.	La aplicación le tiene que indicar que no se ha podido dar de alta el documento.	

Caso de uso:	**2.2 Listar documentos XML**
Grupo de pruebas:	**Listar documentos XML**
Tipo de prueba:	**Funcional**

#	Prueba	Resultado esperado	Resultado real
F1	Pulsar opción de menú *Ficheros – Listar*.	Se tiene que cargar en el área de trabajo una pantalla donde en la parte derecha tienen que salir el listado de documentos del grupo (tanto los del propio *peer* como los del resto de *peers* del grupo). En la parte central un editor de texto y en la parte inferior un espacio para introducir el nombre del fichero que estará deshabilitado y dos botones. El botón *Guardar* el *Eliminar* estarán deshabilitados	
F2	Pulsar opción de menú *Ficheros – Listar* y verificar documentos propietarios y no propietarios. Para ello, disponer de un documento creado por el *peer* de ejecución y otro documento creado por otro *peer*.	En el listado de documentos, todos los documentos propiedad del *peer* que se está ejecutando tiene que mostrarse una S en la columna *Propietario*, y una N en aquellos documentos propiedad de otro *peer*.	
F3	Pulsar opción de menú *Ficheros – Listar* y pulsar sobre un documento propietario	Se tiene que cargar el contenido del documento en el editor, se tiene que cargar el nombre en el cuadro *Nombre* y se tienen que habilitar los botones *Guardar, Eliminar*. A parte, el nombre NO tiene que poder ser editado y el contenido SI.	
F4	Pulsar opción de menú *Ficheros – Listar* y pulsar sobre un documento NO propietario	Se tiene que cargar el contenido del documento en el editor, se tiene que cargar el nombre en el cuadro *Nombre* y se tienen que deshabilitar los botones *Guardar, Eliminar*. A parte, el nombre NO tiene que poder ser editado y el contenido SI (aunque no se puede *Guardar*).	

Caso de uso:		**2.3 Modificar documento XML**	
Grupo de pruebas:		**Modificar documentos XML**	
Tipo de prueba:		**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Modificar el contenido del fichero para hacer que el XML no sea correcto	La aplicación te tiene que avisar que no es un formato correcto y no guardará los cambios.	
F2	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Modificar el contenido y dejarlo en blanco	La aplicación te tiene que avisar que debe introducir el contenido y no guardará los cambios.	
F3	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Modificar el contenido por un contenido xml válido.	La aplicación guardará los cambios en el fichero en disco; actualizará el registro de base de datos del documento para establecer la nueva fecha de actualización y enviará los cambios al resto de *peers* del grupo (incluido *superpeer*).	

	La aplicación estará en estado CONECTADO.	A parte, refrescará el listado y se podrá observar que la fecha ha cambiado.	
F4	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Modificar el contenido por un contenido xml válido. La aplicación estará en estado DESCONECTADO.	La aplicación guardará los cambios en el fichero en disco; actualizará el registro de base de datos del documento para establecer la nueva fecha de actualización y guardará los cambios realizados en base de datos para poder informar de ellos una vez se conecte al grupo. A parte, refrescará el listado y se podrá observar que la fecha ha cambiado.	

	Caso de uso:	**2.4 Borrar documento XML**	
	Grupo de pruebas:	**Borrar documentos XML**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Pulsar *Eliminar*. El estado de la aplicación CONECTADO	Al pulsar el botón *Eliminar* la aplicación pedirá confirmación de dicha acción. Al pulsar SI, se borrará el documento en base de datos, en disco y se enviará dicho evento al resto de *peers* del grupo (incluido *superpeer*) para que realicen la acción oportuna. A parte, actualizará listado y este documento ya no tiene que aparecer.	
F2	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Pulsar *Eliminar* y cuando pida confirmación, pulsar NO. El estado de la aplicación CONECTADO	Al usuario pulsar NO en la confirmación, la aplicación no tiene que hacer nada; por lo que no eliminará el documento.	
F3	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Pulsar *Eliminar* y cuando pida confirmación, pulsar cancelar. El estado de la aplicación CONECTADO	Al usuario pulsar NO en la confirmación, la aplicación no tiene que hacer nada; por lo que no eliminará el documento.	
F4	Pulsar opción de menú *Ficheros – Listar*, cargar un documento que sea propiedad del *peer*. Pulsar *Eliminar* y luego Si. El estado de la aplicación DESCONECTADO	Se borrará el documento en base de datos, en disco y se guardará dicho cambio (Baja) en base de datos para poder informar al resto de miembros del grupo una vez se conecte el *peer*. A parte, actualizará listado y este documento ya no tiene que aparecer.	

	Caso de uso:	**3.1 Configurar**	
	Grupo de pruebas:	**Configurar**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Sistema – Configurar.*	Se mostrará en el área de trabajo un nuevo panel donde se reflejarán los datos	

		configurables. Se mostrará *Directorio de trabajo* con el actúalo directorio donde la aplicación guarda los documentos del grupo.La IP y puerto del *superpeer*.	
F2	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro directorio dejándolo vacío y pulsar *guardar*.	La aplicación tiene que avisar que es obligatorio rellenar el directorio. No modificará nada.	
F3	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro directorio indicando un nuevo directorio de trabajo. Pulsar *guardar*.	La aplicación avisará que el directorio de trabajo se ha modificado, modificará el registro en base de datos y a partir de ese momento, cuando se dé de alta un nuevo documento (bien al crear un nuevo xml o porque otro *peer* nos indique que ha creado uno nuevo), se tiene que guardar en el nuevo directorio. A los documentos existentes NO se le modificará la ruta.	
F4	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro IP *SuperPeer* poniendo una IP no válida o dejarlo vacío. Pulsar *guardar*.	La aplicación avisará que el valor del parámetro no es válido. No modificará nada.	
F5	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro IP *SuperPeer.* Pulsar *guardar*.	La aplicación avisará que la IP de conexión con superpeer se ha modificado, modificará el registro en base de datos, modificará configuración jxta local y a partir de ese momento, cuando se vuelva a conectar al grupo, utilizará este valor para conectar con el *superpeer*.	
F6	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro Puerto *SuperPeer* poniendo un puerto no válido, inferior a 9701 o superior a 9799 o dejarlo vacío. Pulsar *guardar*.	La aplicación avisará que el valor del parámetro no es válido. No modificará nada.	
F7	Pulsar opción de menú *Sistema – Configurar.* Modificar parámetro Puerto *SuperPeer.* Pulsar *guardar*.	La aplicación avisará que el Puerto de conexión con superpeer se ha modificado, modificará el registro en base de datos, modificará configuración jxta local y a partir de ese momento, cuando se vuelva a conectar al grupo, utilizará este valor para conectar con el *superpeer*.	

	Caso de uso:	**3.2 Salir**	
	Grupo de pruebas:	**Salir**	
	Tipo de prueba:	**Funcional**	
#	**Prueba**	**Resultado esperado**	**Resultado real**
F1	Pulsar opción de menú *Sistema – Salir.*	La aplicación cerrará comunicaciones y se cerrará.	

5 ESTUDIO TÉCNICO

5.1 INTRODUCCIÓN

El estudio técnico estará organizado de la siguiente forma:

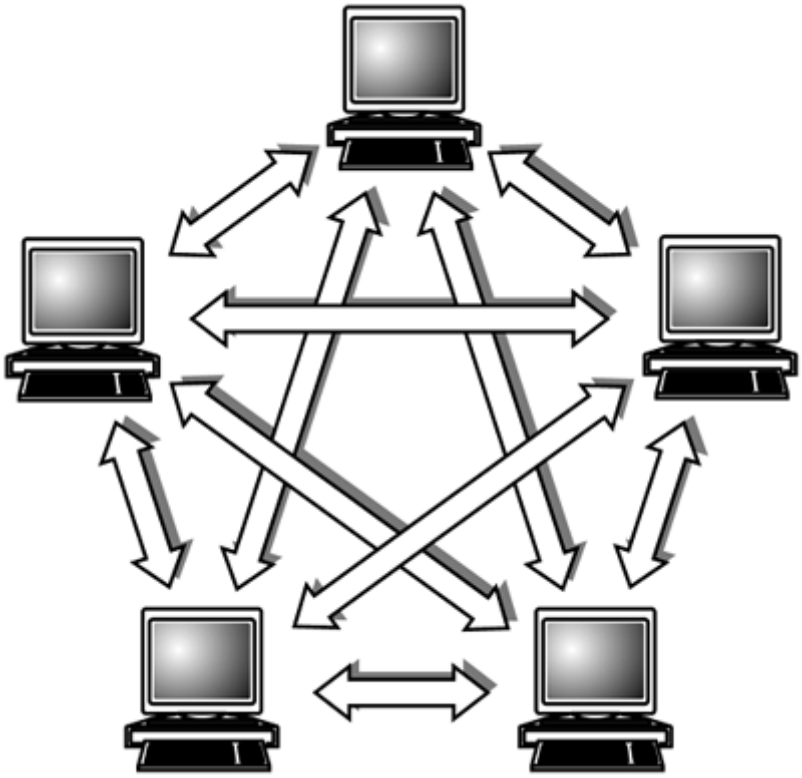
- Necesidades de Arquitectura: Introducción global de las necesidades de arquitectura.
- Tecnologías: Explicación de las decisiones tecnológicas tomadas para llevar a cabo este proyecto.
- Diseño de arquitectura: Arquitectura a utilizar.
- Definición de objetos principales. Ordenación y estructuración del código: Definición de los principales objetos (clases). Decisiones de cómo se estructurará el código y la nomenclatura a utilizar.
- Operaciones de negocio: Descripción de cada una de las operaciones de negocio con sus diagramas de secuencia. Son acciones iniciadas por el usuario
- Lógica de negocio: Descripción de los distintos mensajes que tanto los nodos *peer* como el nodo *superpeer* gestiona y son provocados por alguna de las operaciones de negocio definidas
- Puesta en marcha de la solución: Pasos a realizar para poner en marcha la aplicación.

5.2 NECESIDADES DE ARQUITECTURA

El objetivo principal del proyecto es crear un entorno colaborativo para poder compartir documentos de trabajo.

Este entorno colaborativo huye de la utilización de la clásica arquitectura cliente-servidor y se adentra en una arquitectura conocida como *red peer to peer*, *red de pares o red entre iguales*.

Una *red peer to peer* es una red de computadoras en la que todos o algunos aspectos funcionan sin clientes ni servidores fijos, sino una serie de nodos que se comportan como iguales entre sí. Es decir, actúan simultáneamente como clientes y servidores respecto a los demás nodos de la red. Las redes P2P permiten el intercambio directo de información, en cualquier formato, entre los ordenadores interconectados.



Las redes *peer-to-peer* aprovechan, administran y optimizan el uso del ancho de banda de los demás usuarios de la red por medio de la conectividad entre los mismos, y obtienen así más rendimiento en las conexiones y transferencias que con algunos métodos centralizados convencionales, donde una cantidad relativamente pequeña de servidores provee el total del ancho de banda y recursos compartidos para un servicio o aplicación.

(ref: wikipedia)

En el caso que nos ocupa, se montará un pequeña *red peer-to-peer* en un entorno de trabajo con el fin de poder compartir los documentos xml que se generan en los distintos nodos.

5.3 TECNOLOGÍAS UTILIZADAS

Conocemos qué tiene que hacer la solución y con qué objetivos. Se ha visto qué arquitectura se utilizará para poder disponer del entorno colaborativo que buscamos y se sabe que dispondremos de un frontal pesado para permitir al usuario llevar a cabo todas las acciones necesarias. Por último, veremos qué elementos técnicos se utilizarán para poder plasmar de forma real todo lo que se persigue.

5.3.1 JAVA COMO ELEMENTO COMÚN

Todos los elementos que se verán a partir de ahora y todas las tecnologías utilizadas para llevar a cabo el proyecto tienen una base común: J2EE.

Java es el lenguaje de programación escogido para el desarrollo íntegro de la solución y todas las librerías, protocolos y herramientas utilizadas están pensados para trabajar con Java.

La versión de java utilizada es:

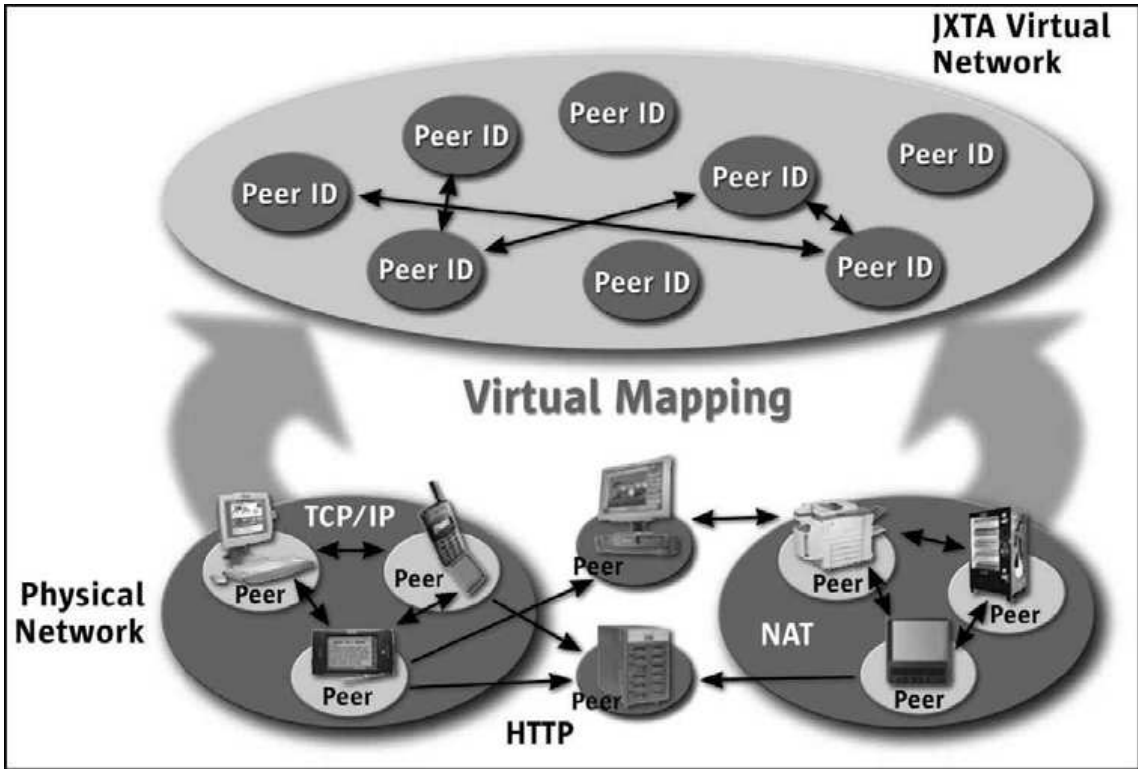
```
java version "1.6.0_24"
Java(TM) SE Runtime Environment (build 1.6.0_24-b07)
Java HotSpot(TM) Client VM (build 19.1-b02, mixed mode, sharing)
```

5.3.2 JXTA Y LA RED PEER TO PEER

5.3.2.1 Introducción

El proyecto JXTA estandariza un grupo de protocolos para construir redes virtuales P2P. Los protocolos JXTA definen los requerimientos de red mínimos para que los pares formen y se unan a una red virtual mediante una capa de red genérica. El proyecto JXTA posibilita a los programadores y desarrolladores de aplicaciones, no solo a los administradores de red, diseñar la topología de la red para adecuase a los requerimientos específicos de la aplicación. Se pueden crear múltiples redes virtuales Ad Hoc y dinámicamente mapearlas en una única red física.

La tecnología JXTA consiste en un conjunto de simples protocolos abiertos (open source) peer to peer que permiten a los dispositivos de la red comunicarse, colaborar y compartir recursos.



(esquema de cómo los pares JXTA crean una red virtual sobre la red existente, ocultando así su complejidad).

En esta red virtual JXTA cualquier par puede interactuar con otros pares a pesar de su ubicación, tipo de dispositivo o sistema operativo; Inclusive cuando algunos pares y recursos están ubicados detrás de un firewall o trabajan sobre un tipo de transporte de red diferente. Así el acceso a los recursos de red no está limitado por incompatibilidades de plataforma o los permisos de una arquitectura jerárquica cliente servidor. La tecnología JXTA funda sus objetivos en la ubicuidad, la independencia de plataformas, la interoperatividad y la seguridad. Puede correr sobre cualquier dispositivo que acceda a la red, incluyendo teléfonos móviles, PDAs, sensores electrónicos, computadoras de escritorio y servidores. Al estar basado en tecnologías probadas y estandarizadas tales como HTTP, TCP/IP, y XML la tecnología JXTA no depende de un lenguaje de programación en particular ni de la plataforma de red ni del sistema operativo y por eso puede trabajar con cualquier combinación de ellos.

5.3.2.2 Arquitectura JXTA

La arquitectura software de JXTA está dividida en tres capas:

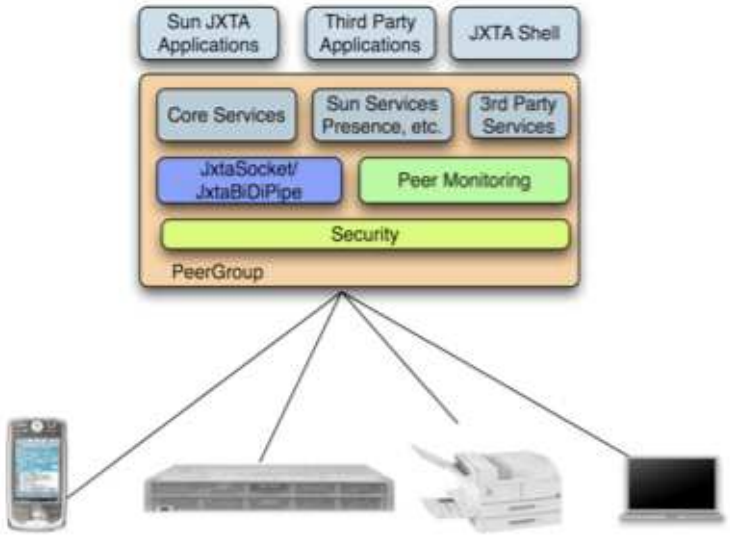


Figure 2: JXTA Software Architecture

5.3.2.3 Peers / SuperPeer

Un igual es cualquier entidad de red que implemente uno o más de los protocolos de JXTA. Los iguales pueden residir en sensores, teléfonos, y PDAs, también como en PCs, servidores, y supercomputadores. Cada igual opera independientemente y asíncronamente de todos los otros iguales y es identificado únicamente por un ID de igual.

Tipología de Peers: Se pueden categorizar dentro de tres tipos principales:

- **Minimal-Edge peers**: Iguales que implementan solo los servicios requeridos del corazón (core) de JXTA y pueden contar con otros iguales para que actúen como su proxy para otros servicios para participar completamente en una red JXTA. Los iguales proxy actúan como un proxy para los servicios que no son del corazón (non-core services). Los iguales Minimal-Edge incluyen típicamente sensores y dispositivos de automatización de casas.
- **Full-Edge peers**: Iguales que implementan todos los servicios del corazón y estándar de JXTA y pueden participar en todos los protocolos JXTA. Estos iguales conforman la mayoría de iguales en una red JXTA y pueden incluir teléfonos, PC's, servidores, etc.
- **Super-Peer**: Iguales que implementan y proveen recursos para soportar el despliegue y operación de una red JXTA. Hay

tres funciones principales en un Super-Peer. Un igual simple puede implementar una o más de estas funciones

- **Relay**(receive, reinforce, and retransmit): Usada para almacenar y reenviar mensajes entre iguales que no tienen conectividad directa debido cortafuegos o NATs. Solamente los iguales que no están en capacidad de recibir conexiones de otros iguales requieren un relay.
- **Rendezvous**: Mantiene los índices globales de publicidad y ayuda a iguales extremos(edge) y proxys con el anuncio de búsquedas. También se ocupa de la radiodifución de mensajes
- **Proxy**: Utilizado por los Minimal-Edge peers para tener acceso a todas las funcionalidades de la red JXTA. El igual proxy, traduce y resume las peticiones, responde a las consultas y proporciona soporte para la funcionalidad de los Minimal-Edge peers.

5.3.2.4 Grupo de Peers

Un grupo de iguales, es una colección de iguales que se han puesto de acuerdo en un conjunto común de servicios o intereses. Los iguales se autorganizan en grupos de iguales, cada uno de los cuales se idéntica unívocamente por un ID de grupo de iguales. Cada grupo de iguales establece su propia política de membresía incluyendo abierta (cualquier igual puede afiliarse) hasta la muy segura y protegida (que requiere credenciales para poder ser miembro).

Los iguales pueden pertenecer a más de un grupo de iguales simultáneamente. Por defecto, el primer grupo que se instancia es el Network Peer Group. Todos los iguales pertenecientes al Network Peer Group, pueden optar por unirse a otros grupos de iguales en cualquier momento.

(ref: JXSE_ProgGuide_v2.5)

5.3.2.5 Cómo trabajar con JXTA

Para poder trabajar con los protocolos establecidos en JXTA es necesario importar los siguientes componentes:

jxta.jar (http://download.java.net/jxta/)

5.3.2.6 Mapa de componentes

Se han creado los siguiente objetos para gestionar toda la creación y gestión de la *red peer to peer*.

PFC_Peer.java: Objeto principal que crea y gestiona todo lo necesario de un nodo tipo *peer*.

PFC_PeerComun.java: Elementos comunes de trabajo dentro de elementos *Peers*.

PFC_PGestorComunicacion.java: Objeto que gestiona todas las comunicación tipo Unicast entre los distintos elementos del grupo (tanto comunicación individuales con otro *peers* como la comunicación individual que pueda haber con el *superpeer*).

PFC_PGestorComunicacionMC.java: Objeto que gestiona todas las comunicación tipo Multicast dentro del grupo de trabajo. Son mensajes enviados al grupo.

PFC_SuperPeer.java: Objeto principal que crea y gestiona todo lo necesario de un nodo tipo *superpeer*.

PFC_SPGestorComunicacion.java: Objeto que gestiona todas las comunicación tipo Unicast entre el *superpeer* y los distintos *peers* del grupo de trabajo.

PFC_SPGestorComunicacionMC.java: Objeto que gestiona todas las comunicación tipo Multicast dentro del grupo de trabajo. Son mensajes enviados al grupo.

PFC_JxtaID.java: Gestiona la creación de identificador únicos.

PFC_JxtaUtilidades.java: Funciones de utilidad vinculadas con Jxta y utilizadas por cualquier otro objeto.

PFC_Mensaje.java: Encapsula cualquier mensaje que circula en el grupo creado.

5.3.3 SWING COMO FRONTAL

5.3.3.1 Introducción

Al tratarse de una aplicación de desktop pesada se ha desarrollado todo el *core* e interficie gráfica de los nodos tipo *peer* con tecnología SWING.

No es objetivo de este proyecto explicar en detalle dicha tecnología pero se hace una breve explicación para luego detallar el mapa y estructuras de marcos y ventanas.

SWING es un framework MVC para desarrollar interfaces gráficas para Java con independencia de la plataforma. Sigue un simple modelo de programación por hilos, y posee las siguientes características principales:

- Independencia de plataforma.
- Extensibilidad: es una arquitectura altamente particionada: los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.
- Personalizable: dado el modelo de representación programático del framework de swing, el control permite representar diferentes estilos de apariencia "look and feel" (desde apariencia MacOS hasta apariencia Windows XP pasando por apariencia GTK+, IBM UNIX o HP UX entre otros). Además, los usuarios pueden proveer su propia implementación de apariencia, que permitirá cambios uniformes en la apariencia existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación

(ref: wikipedia)

5.3.3.2 Cómo trabajar con SWING

Para poder trabajar con la librería gráfica de Java es necesario importar lo siguiente:

```
Java.swing.*;
Java.awt.*;
```

Si bien es cierto que en nuestro caso, cuando lo ejecutamos por línea de comandos bajo un sistema Windows no es necesario importar ningún jar específico.

En caso de ejecutarlo bajo sistema Linux, quizá sea necesario importar esas librerías gráficas.

5.3.3.3 Mapa de componentes

Toda la interficie de los nodos tipo *peer* son ventanas creadas con swing. El mapa de componentes creado es el siguiente:

PFC_PanelPrincipal.java: Objeto contenedor del marco principal de la aplicación.

Define la estructura de menú, área de trabajo y pie de estado.

PFC_PanelConectarConexion.java: Objeto que define el panel de trabajo para poder conectar o desconectar al grupo. Accesible desde la opción de menú *Conexión – Gestión*.

PFC_PanelListarFicheros.java: Objeto que define el panel de trabajo para crear un nuevo documento o listar los documentos existentes. Se accede desde cualquier opción del menú *Ficheros*.

PFC_PanelConfigurar.java: Objeto que define el panel de trabajo para configurar los parámetros de la aplicación. Accesible desde la opción de menú *Sistema – Configurar*.

5.3.4 BASE DE DATOS DERBY

En la primera entrega del proyecto se hizo un estudio de elección de base de datos. En ese estudio se decidió utilizar la base de datos *Derby (JavaDB)* ya que era 100% java y permitía una forma de trabajo *empotrado* que ofrecía un servicio de repositorio de datos transparente para el usuario; no necesita de una instalación previa y en caliente se van creando las tablas a utilizar.

Como se ha visto en el apartado de apartado *Diseño de la base de datos*, la aplicación trabaja con tres entidades y con un diseño muy sencillo.

5.3.4.1 Cómo trabajar con Derby

Para poder trabajar con *Derby* en una determinada aplicación se tiene que importar la librería:

derby.jar (http://db.apache.org/derby/index.html)

Esto permite a cualquier código java crear los puentes necesarios para tratar con una base de datos *derby*.

A parte también se ha utilizado un aplicación de comandos para poder manipular/consultar los datos de una determinada base de datos. Esta aplicación de comandos es ***ij*** y se puede encontrar con la distribución de *derby*.

5.3.4.2 Configuración de la base de datos

En cuanto a base de datos, el funcionamiento de las dos aplicaciones, por lo que se describe a continuación sirve tanto para los nodos tipo *peer* como para el nodo tipo *superpeer*.

Cuando arranca la aplicación no existe base de datos por lo que la primera vez se creará. El nombre de la base de datos que se crea es: *PFCGrupoBD*. Esto se puede hacer con la ejecución del comando:

Jdbc:derby:PFCGrupoBD;create=true

Con lo que si la base de datos no existe la crea; si existe únicamente se conecta a ella.

Una vez hecho esto, *Derby* crea un directorio con el mismo nombre dado a la base de datos (en este caso PFCGrupoBD) y en el path donde se está ejecutando la aplicación, donde residirá toda la base de datos.

El siguiente paso que hace la aplicación es verificar si existe la tabla: CONFIGURACION.

La primera vez que se ejecuta la solución esta tabla no existirá, por lo que el sistema creará todo el modelo de datos (CONFIGURACION, DOCUMENTOS, DOC_CAMBIOS) e inicializará la tabla CONFIGURACION con los datos de configuración por defecto.

A partir de ahí, las posteriores veces que se acceda a la aplicación, el sistema detectará que ya existe la tabla CONFIGURACION y lo que hará será cargar los datos de configuración en memoria.

5.3.4.3 Mapa de componentes

El objeto encarga de gestionar todo lo referente a la base de datos es:

PFC_BDGestion.java: Objeto que se crea al inicio del programa y se va utilizando en las distintas clases. Al salir de la aplicación se desconecta de base de datos y se destruye.

5.3.5 TRATAMIENTO DOCUMENTOS XML CON JDOM

La creación y tratamiento de documentos XML se realizar con la librería libre JDOM.

JDOM son unas librerías java pensadas para el tratamiento de documentos XML que permiten trabajar con ese tipo de ficheros de una forma rápida y sencilla.

5.3.5.1 Cómo trabajar con JDOM

Para poder trabajar con esta librería es necesario importar lo siguiente:

Jdom.jar

Jaxen.jar

(http://www.jdom.org/downloads/index.html)

5.3.5.2 Mapa de componentes

Los objetos creados para tratar con documentos xml son:

PFC_GestorXML.java: Se encarga de la creación de xml, carga de contenido y búsqueda de diferencias entre dos documentos xml.

5.3.6 SISTEMA DE TRAZAS CON LOG4J

La aplicación dispone de un sistema de logs para poder debugar o disponer de trazas en tiempo de ejecución.

El sistema utilizado con este fin es la conocida librería java *log4j*.

Log4j es una biblioteca open source desarrollada en Java por la Apache Software Foundation que permite a los desarrolladores de software elegir la salida y el nivel de granularidad de los mensajes o "logs" (data logging) a tiempo de ejecución y no a tiempo de compilación como es comúnmente realizado.

(ref: wikipedia)

5.3.6.1 Cómo trabajar con LOG4J

Para poder trabajar con esta librería es necesario importar lo siguiente:

log4j-1.2.16.jar

(*http://logging.apache.org/log4j/*)

5.3.6.2 Mapa de componentes

En esta ocasión no existe un objeto que gestiona esta funcionalidad. En cada clase que se quiera utilizar el sistema de trazas se tiene que definir lo siguiente:

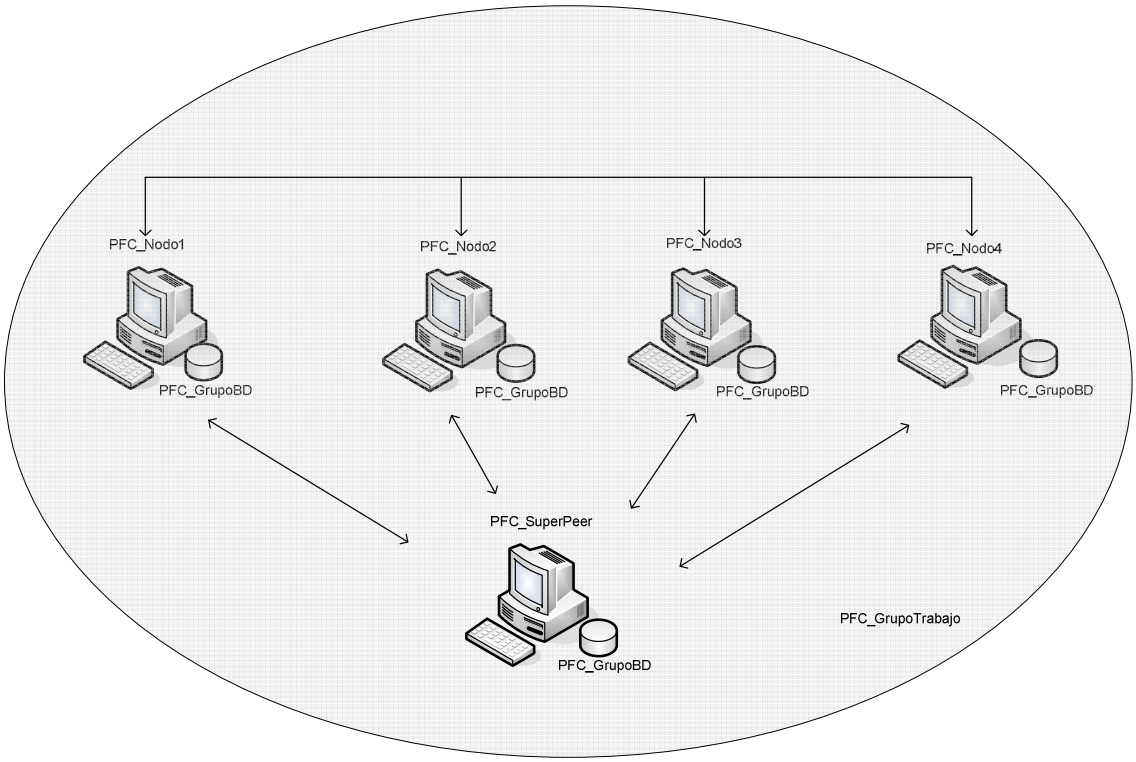
```
private final static Logger moLog = Logger.getLogger(PFC_GestorXML.class);
```

De esta forma, la clase PFC_GestorXML dispondrá de un sistema de tarzas siempre y cuando se invoque al objeto *moLog*.

Inicialmente, y al arranque de la aplicación, se establece la configuración del sistema de trazas. Se trata de cargar el fichero *log4j.properties*.

5.4 DISEÑO DE ARQUITECTURA

Una vez se han visto las necesidades y tecnologías a utilizar, se detalla el mapa final de la arquitectura que se creará para montar un sistema distribuido capaz de gestionar documentos xml creando un grupo de trabajo donde cada nodo dispondrá de todos los documentos que se creen en dicho grupo.



Se crea un grupo de trabajo con nombre **PFC_GrupoTrabajo** mediante el cual cada uno de los 4 nodos tipo *peer* y el nodo tipo *supepeer* estarán conectados.

Aunque la representación ha sido que cada nodo estará en una máquina, es posible disponer todos los nodos en una misma máquina siempre y cuando los directorios de instalación sean distintos.

Como se puede observar el nombre de los nodos *peer* son:

- PFC_Nodo1
- PFC_Nodo2
- PFC_Nodo3
- PFC_Nodo4

Y el nombre del nodo *superpeer*:

- PFC_SuperPeer

A su vez cada uno de los 5 nodos tendrá su propia base de datos con nombre **PFCGrupoBD**.

5.5 OBJETOS PRINCIPALES. ESTRUCTURACIÓN

El código fuente de la aplicación se estructurará de la siguiente forma:

pfc.prototipo: package base. De este package cuelga objetos comunes y todas la clases creadas para la gestión de ventanas.

pfc.prototipo.bbdd: Contiene la clase encargada de gestionar todo lo relacionado con la base de datos.

pfc.prototipo.jxta: package base de los objetos que gestionan la comunicación de la red *peer to peer*. Contiene clases comunes.

pfc.prototipo.jxta.peer: Contiene los objetos relacionados con la comunicación *peer to peer* de un *peer*.

pfc.prototipo.jxta.superpeer: Contiene los objetos relacionados con la comunicación *peer to peer* del *superpeer*.

pfc.prototipo.rendimiento: Contiene el objeto encargado de realizar el log en fichero del tiempo de proceso.

pfc.prototipo.xml: Contiene el objeto encargado de manejar documentos xml utilizando la librería *JDOM*.

5.5.1 CONSTANTES

Para gestionar todas las constantes de la aplicación o funciones comunes que puedan ser estáticas se ha creado el objeto:
pfc.prototipo.PFC_Constantes

5.5.2 MANEJO DE FICHEROS

Para manejar ficheros a bajo nivel, pudiendo convertir de un arrayByte a fichero y viceversa. O eliminar directamente de un path un documento se ha creado el objeto:
pfc.prototipo.PFC_ManejoFicheros

5.5.3 MARCO PRINCIPAL

Los nodos *peer* disponen de una interficie gráfica creada mediante las librerías swing que permite al usuario interactuar con la aplicación.

Para gestionar el marco principal (ventana) se ha creado el objeto:

pfc.prototipo.PFC_PanelPrincipal

5.5.4 GESTIÓN DE LA BASE DE DATOS

Toda la gestión de la base de datos se realiza mediante un solo objeto. En esta clase se crea la base de datos y se realizar todas las operaciones necesarias que vayan contra el repositorio de datos. Sólo se crea una instancia por aplicación:
pfc.prototipo.bbdd.PFC_BDGestion

5.5.5 CREAR IDENTIFICADOR ÚNICOS

La comunicación en una red *peer to peer* requiere de identificadores únicos (uuid) para poder identificar cada uno de los elementos que forman parte de dicha red. No sólo necesitan identificadores únicos los *peers* (tanto *superpeer* como *peer*), también los canales de comunicación utilizados. Para crear todos estos identificadores únicos se utiliza el objeto: **pfc.prototipo.jxta.PFC_JxtaID**

5.5.6 MENSAJE

Toda la comunicación entre los componentes del grupo se encapsula con un objeto creado para contener la distinta tipología de mensajes creados con la diferente información necesaria en cada momento. Para gestionar todo esto se ha creado el objeto:
pfc.prototipo.jxta.PFC_Mensaje

Este objeto tiene censado los distintos mensajes definidos:

Tipo	Descripción
TIPO_NUEVO (NUEVO)	Contiene un documento dado de alta por un determinado *peer*.
TIPO_BAJA (BAJA)	Contiene los datos de un documento que ha sido dado de baja por su nodo creador
TIPO_CAMBIO (CAMBIO)	Contiene los cambios realizados sobre un documento. Estos cambios afectan a posibles Alta, Baja o Modificación de nodos del xml.

TIPO_PETICION_ACTUALIZACION (PETICION_ACTUALIZACION)	Mensaje enviado por un *peer* pidiendo que el *superpeer* le envíe todas las actualizaciones que se hayan realizado mientras ha estado desconectado.
TIPO_ACTUALIZACION (ACTUALIZACION)	Mensaje respuesta al tipo anterior donde se informa de todos los cambios que ha habido
TIPO_AVISO_ALTA (AVISO_ALTA)	De entre todas la actualizaciones recibidas, esta indica que tiene que realizar una petición de nuevo documento a un *peer* determinado
TIPO_PETICION_ALTA (PETICION_ALTA)	Mensaje enviado a un determinado *peer* (o al *data center*) donde se indica que necesita que le envíe un determinado documento
TIPO_RESPUESTA_ALTA (RESPUESTA_ALTA)	Mensaje respuesta al anterior donde se informa y adjunta el nuevo documento dado de alta
TIPO_ENVIO_ACTUALIZACIONES (ENVIO_ACTUALIZACIONES)	Mensaje enviado por un determinado *peer* cuando se conecta informando de todos los cambios que ha realizado mientras estaba desconectado.

5.5.7 PEER

La gestión de la comunicación de un *peer* dentro de la red *peer to peer* se realiza con el objeto: **pfc.prototipo.jxta.peer.PFC_Peer**

5.5.8 COMUNICACIÓN UNICAST

Para recibir mensajes del tipo Unicast se han creado dos objetos, uno para *peer* y otro para *superpeer* que son lanzados mediante *Thread* y que su ejecución es infinita mientras corre la aplicación. Estos objetos son: **pfc.prototipo.jxta.peer.PFC_PGestorComunicacion, pfc.prototipo.jxta.superpeer.PFC_SPGestorComunicacion**

5.5.9 COMUNICACIÓN MULTICAST

Para recibir mensajes del tipo Multicast se han creado dos objetos, uno para *peer* y otro para *superpeer* que son lanzados mediante *Thread* y que su ejecución es infinita mientras corre la aplicación. Estos objetos son:
pfc.prototipo.jxta.peer.PFC_PGestorComunicacionMC, pfc.prototipo.jxta.superpeer.PFC_SPGestorComunicacionMC

5.5.10 SUPERPEER

La gestión global del *superpeer* se realiza con el objeto: **pfc.prototipo.jxta.superpeer.PFC_SuperPeer**

5.5.11 DOCUMENTOS XML

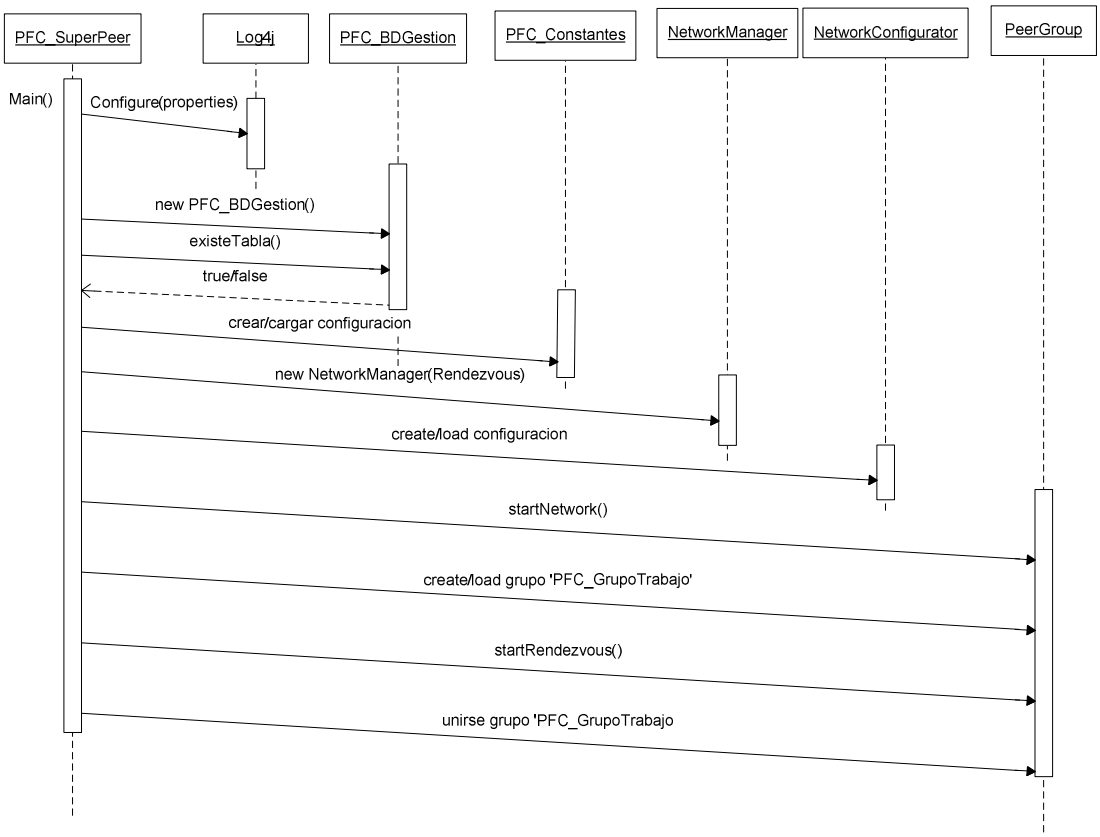
La gestión de documentos xml se hace utilizando la librería *JDOM*. Para ello, se ha creado un objeto que es el encargado de manipular este tipo documentos: **pfc.prototipo.xml.PFC_GestorXML**

5.6 OPERACIONES DE NEGOCIO

5.6.1 INICIAR LA APLICACIÓN – NODO SUPERPEER

La aplicación *superpeer* se definirá como un proceso java que correrá sin frontal y que estará a la espera de recibir peticiones de los nodos *peer*.

El arranque partirá de un método main dónde en una primera fase se realiza la carga/creación de la configuración:



El objeto a lanzar mediante main es la clase **PFC_SuperPeer**. Al arrancar será necesario que:

- Inicialice sistema de trazas con *log4j*. Para ello habrá que configurar e indicar fichero *log4j.properties*.
- Crear base de datos y conexión necesaria.
- Verificar si existe configuración creada. Si es la primera vez que se ejecuta la solución no tendrá configuración por lo que se crearán todas las tablas y se cargará en base de datos la configuración por defecto siguiente:

Parámetro	**Valor**
CONF_NombrePeer	PFC_SuperPeer
CONF_IdPeer	Crear identificador único en base al nombre del peer y al nombre del grupo
CONF_NombreGrupo	PFC_GrupoTrabajo
CONF_IdPipeMulticast	Crear identificador único en base al nombre del PipeMulticast(ComunicacionGrupo) y al nombre del grupo
CONF_IdPipeUnicast	Crear identificador único en base al nombre del PipeUnicast(ComunicacionPeer) y al nombre del

	grupo
CONF_SuperPeerIP	127.0.0.1
CONF_SuperPeerPuerto	9717
CONF_Directorio	Será el path de ejecución de la aplicación +"\documentos\"

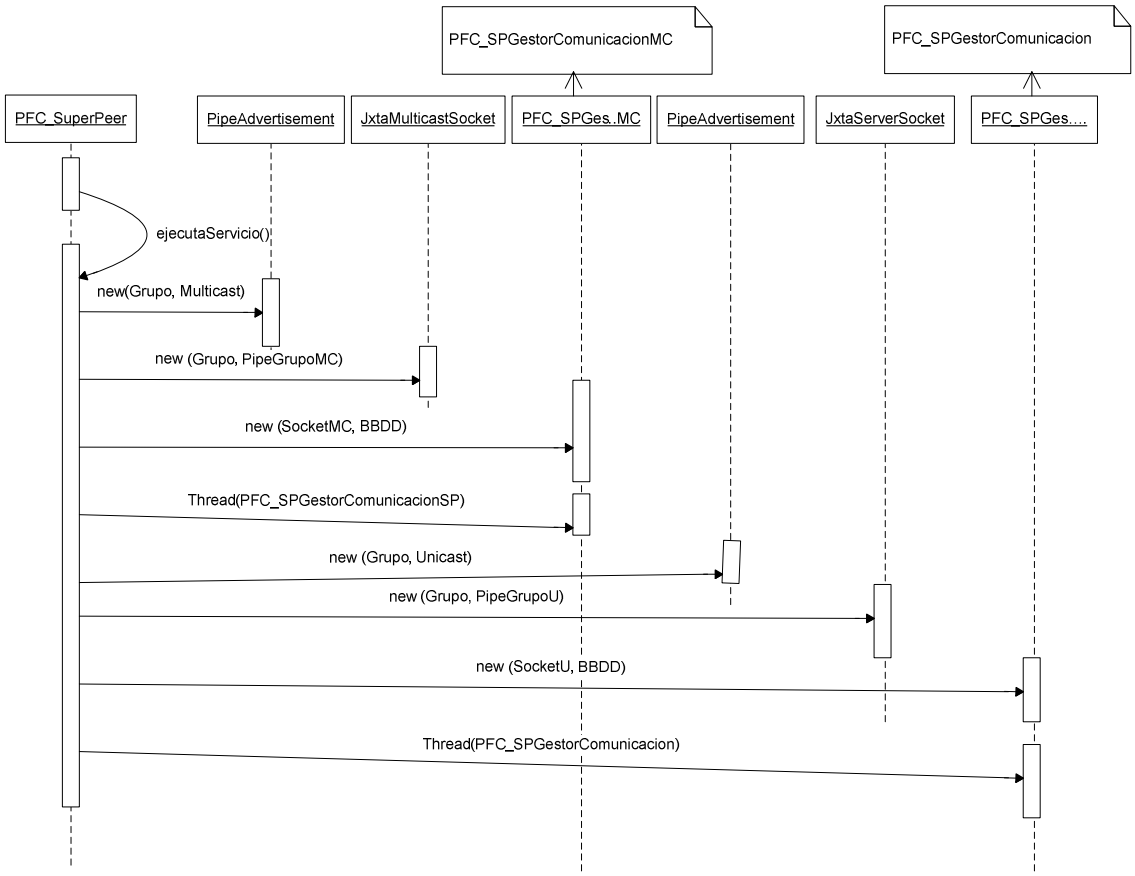
- Si no es la primera vez que se ejecuta la aplicación, cargar en memoria la configuración existente en base de datos
- Crear un peer del tipo: *RENDEZVOUS_RELAY*, indicando el grupo de trabajo *PFC_GrupoTrabajo* y el nombre del propio peer *PFC_SuperPeer*
- Cargar configuración del Peer local si existe.
- Si no existe, crear configuración indicando la siguiente configuración:

Parámetro	**Valor**
Nombre	Parámetro de configuración CONF_NombrePeer
ID	Parámetro de configuración CONF_IdPeer
TCP Puerto	Parámetro de configuración CONF_SuperPeerPuerto
TCP Activado	True
Multicast activado	True

- Si existe la configuración local, cargarla.
- Iniciar la red
- Verificar si el Grupo se ha creado. En caso negativo, se crea. En cualquier caso, se une al Grupo.

Cabe la posibilidad de arrancar el *superpeer* pasando por parámetro el número de puerto por donde escuchará. En tal caso, a la hora de crear la configuración u obtener la ya creada, actualizará este parámetro con el valor introducido a la hora de arrancar la aplicación.

Con esta primera fase de carga de configuración e inicialización de la red, se inicia una segunda fase que se encarga de arrancar los canales de comunicación:



El *superpeer* tendrá que inicializar los canales específicos para poder comunicarse con el resto de miembros del grupo. Para ello:

- Se creará un Pipe (canal de comunicación) del tipo Multicast que se utilizará para cualquier comunicación tipo Multicast (propagar) entre los miembros del grupo.
- Se creará un socket tipo JxtaMulticastSocket para recibir mensajes del tipo Multicast
- Se lanzará el gestor de comunicaciones (PFC_SPGestorComunicacionesMC) que centralizará estas peticiones y se encargará de atender los mensajes recibidos por ese canal. Este proceso es un Thread que se ejecuta siempre y que se queda a la espera de recibir mensajes.
- Se creará un Pipe (canal de comunicación) del tipo Unicast que se utilizará para cualquier comunicación tipo Individual entre dos miembros del grupo.
- Se creará un socket tipo JxtaServerSocket para recibir mensajes del tipo Unicast
- Se lanzará el gestor de comunicaciones (PFC_SPGestorComunicaciones) que centralizará estas peticiones y se encargará de atender los mensajes recibidos por

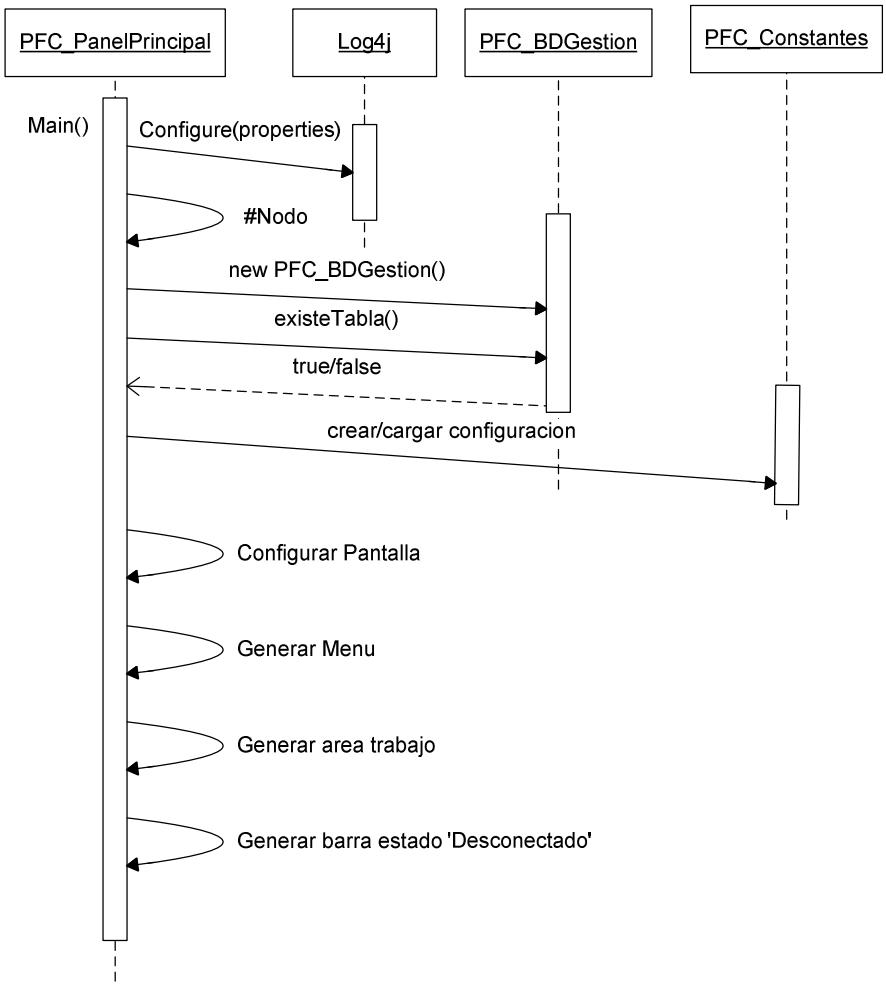
ese canal. Este proceso es un Thead que se ejecuta siempre y que se queda a la espera de recibir mensajes.

5.6.2 INICIAR LA APLICACIÓN – NODO PEER

A diferencia del nodo *superpeer*, el nodo *peer* si que dispone de interficie gráfica con la que interactuará el usuario.

De igual forma, se iniciará la aplicación ejecutando el main del objeto **PFC_PanelPrincipal** y obteniendo por parámetro un entero que indica el número de nodo que lanzamos. De tal forma que si se quiere lanzar el nodo 1: PFC_PanelPrincipal 1.

Los pasos para iniciar la aplicación serán:



Los pasos a seguir en el arranque de la aplicación serán:

- Inicializar sistema de trazas con *log4j*. Para ello habrá que configurar e indicar fichero *log4j.properties*.

- Crear base de datos y conexión necesaria.
- Verificar si existe configuración creada. Si es la primera vez que se ejecuta la solución no tendrá configuración por lo que se crearán todas las tablas y se cargará en base de datos la configuración por defecto siguiente:

Parámetro	Valor
CONF_NombrePeer	PFC_Nodox (donde x es el parámetro recibido en main)
CONF_IdPeer	Crear identificador único en base al nombre del peer y al nombre del grupo
CONF_NombreGrupo	PFC_GrupoTrabajo
CONF_IdPipeMulticast	Crear identificador único en base al nombre del PipeMulticast(ComunicacionGrupo) y al nombre del grupo
CONF_IdPipeUnicast	Crear identificador único en base al nombre del PipeUnicast(ComunicacionPeer) y al nombre del grupo
CONF_SuperPeerIP	127.0.0.1
CONF_SuperPeerPuerto	9717
CONF_Directorio	Será el path de ejecución de la aplicación +"\documentos\"

- Si no es la primera vez que se ejecuta la aplicación, cargar en memoria la configuración existente en base de datos

Tener en cuenta que el sistema trabaja sin un fichero de configuración propio. Al arrancar por primera vez la aplicación se crea una configuración por defecto.

A parte, se ha creado una pantalla de configuración donde el usuario puede cambiar el directorio de trabajo, la IP y puerto de conexión del *superpeer*.

Si el usuario modifica el Directorio de trabajo, la aplicación actualizará dicha información en la configuración de base de datos y a partir de ese momento, cualquier alta que se realice trabajará con dicho directorio. Puntualizar que sólo se utilizará el nuevo directorio para las nuevas altas; para documentos ya existentes no se modificará este parámetro.

Si el usuario modifica la IP o Puerto de conexión con *superpeer*, cuando se realice una nueva conexión a la red (grupo de trabajo), el nodo trabajará con estos valores para conectarse al nodo tipo

superpeer. La IP y puerto del *superpeer* sólo se utiliza para realizar la primera conexión con *superpeer*; a partir de ese momento, cualquier comunicación que haya con él se realizará utilizando el identificador único del *superpeer*.

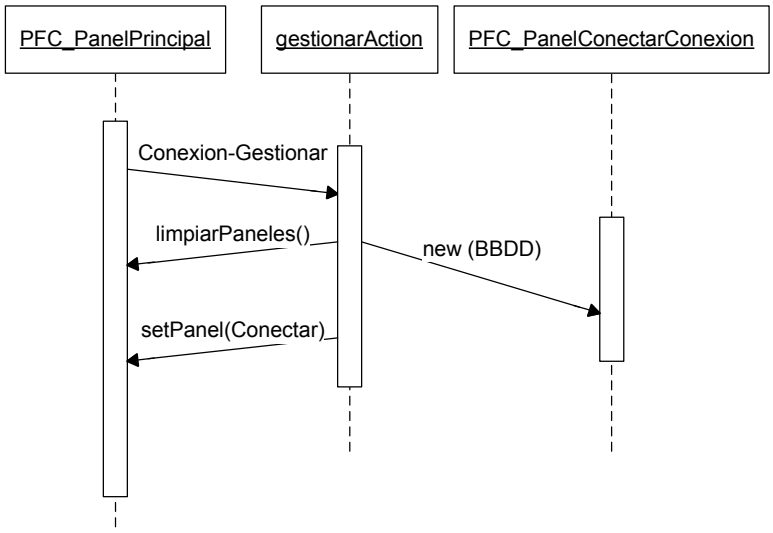
En caso de que el usuario modifique la IP o puerto del *superpeer*, la aplicación actualizará configuración almacenada en base de datos y también actualizará la configuración de comunicación *peer to peer* propia del nodo.

5.6.3 CONECTAR NODO PEER

Una vez la aplicación se ha iniciado, se queda a la espera que el usuario realice alguna acción.

Cuando el usuario pulse la opción de menú *Conexión –*

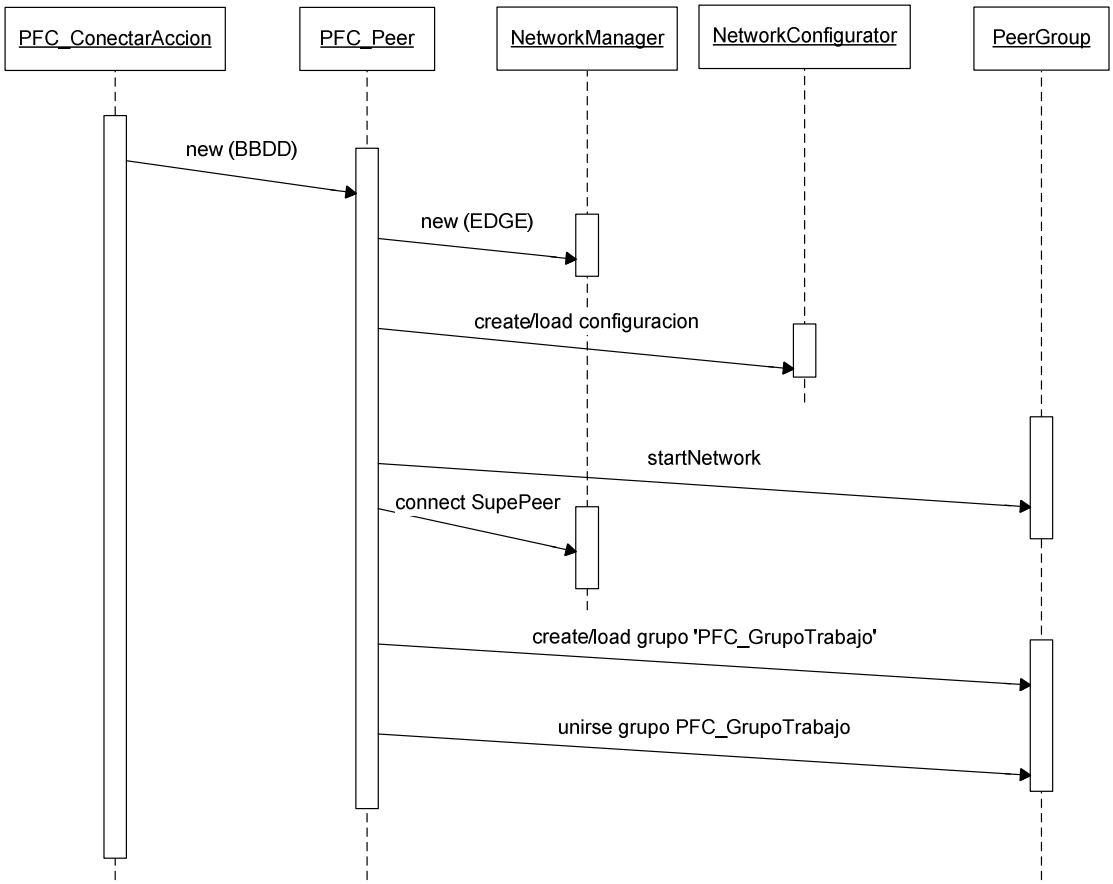
Gestionar en primer lugar se realizará:



Se debe cargar en el área de trabajo del marco principal la pantalla PFC_PanelConectarConexion y realizar:

- Obtener datos necesarios a mostrar en la pantalla
- Si la aplicación no está conectada, habilitar botón conectar y deshabilitar botón desconectar
- Si la aplicación está conectada, habilitar botón desconectar y habilitar botón conectar.

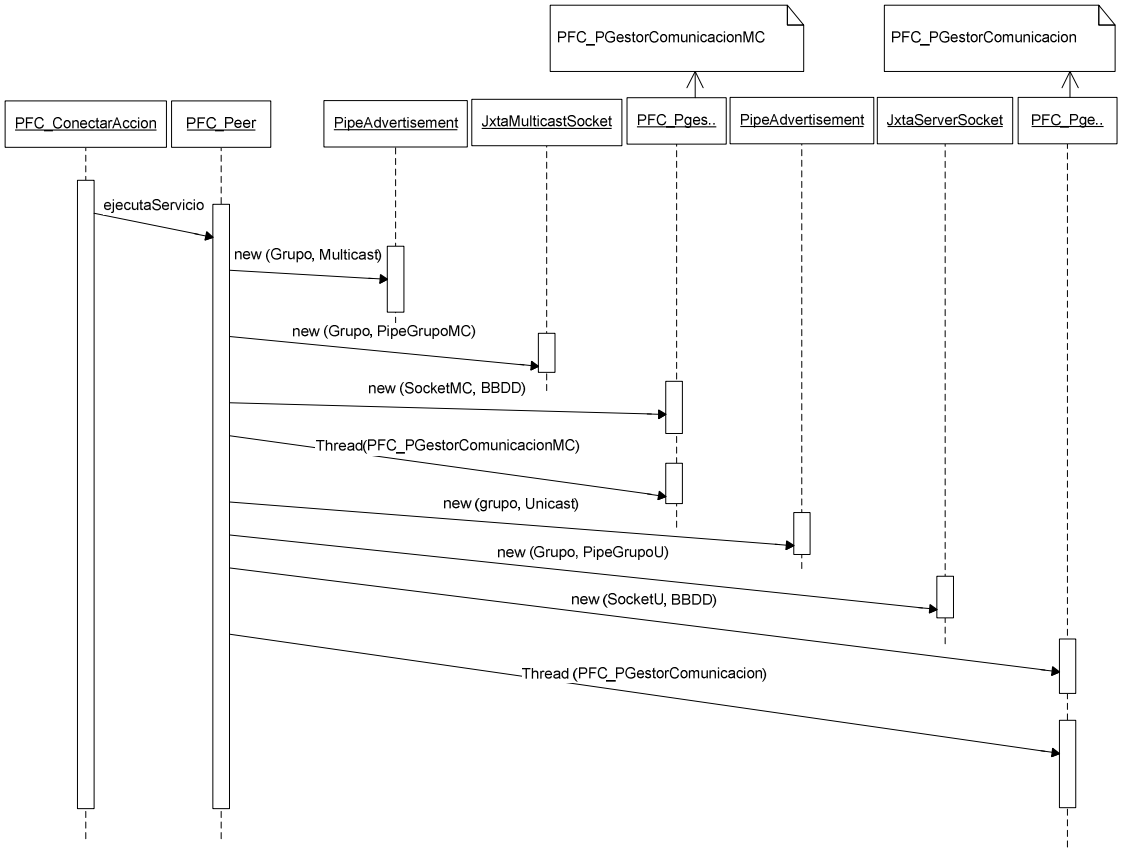
Una vez la pantalla cargada, cuando el usuario pulse el botón conectar se iniciará una primera fase de inicialización de las comunicaciones:



En esta fase se tiene que:

- Instanciar el objeto PFC_Peer que es el encargado de gestionar todas las comunicaciones con el grupo de trabajo
- El PFC_Peer creará una instancia del NetworkManager indicándole el tipo de conexión (EDGE), el nombre (PFC_NODOx) y el grupo al que pertenece.
- Se creará/cargará configuración local de la conexión.
- Se iniciará conexión a la red
- Se realizará conexión con el s*uperpeer*. Tener en cuenta que por configuración se ha establecido la IP y Puerto del *superpeer*.
- Se creará/cargará configuración de grupo
- Se unirá al grupo de trabajo para poder conectar con el resto de nodos.

Una vez finalizada esta fase, se iniciará la fase de arranque de los canales de comunicación del *peer*:



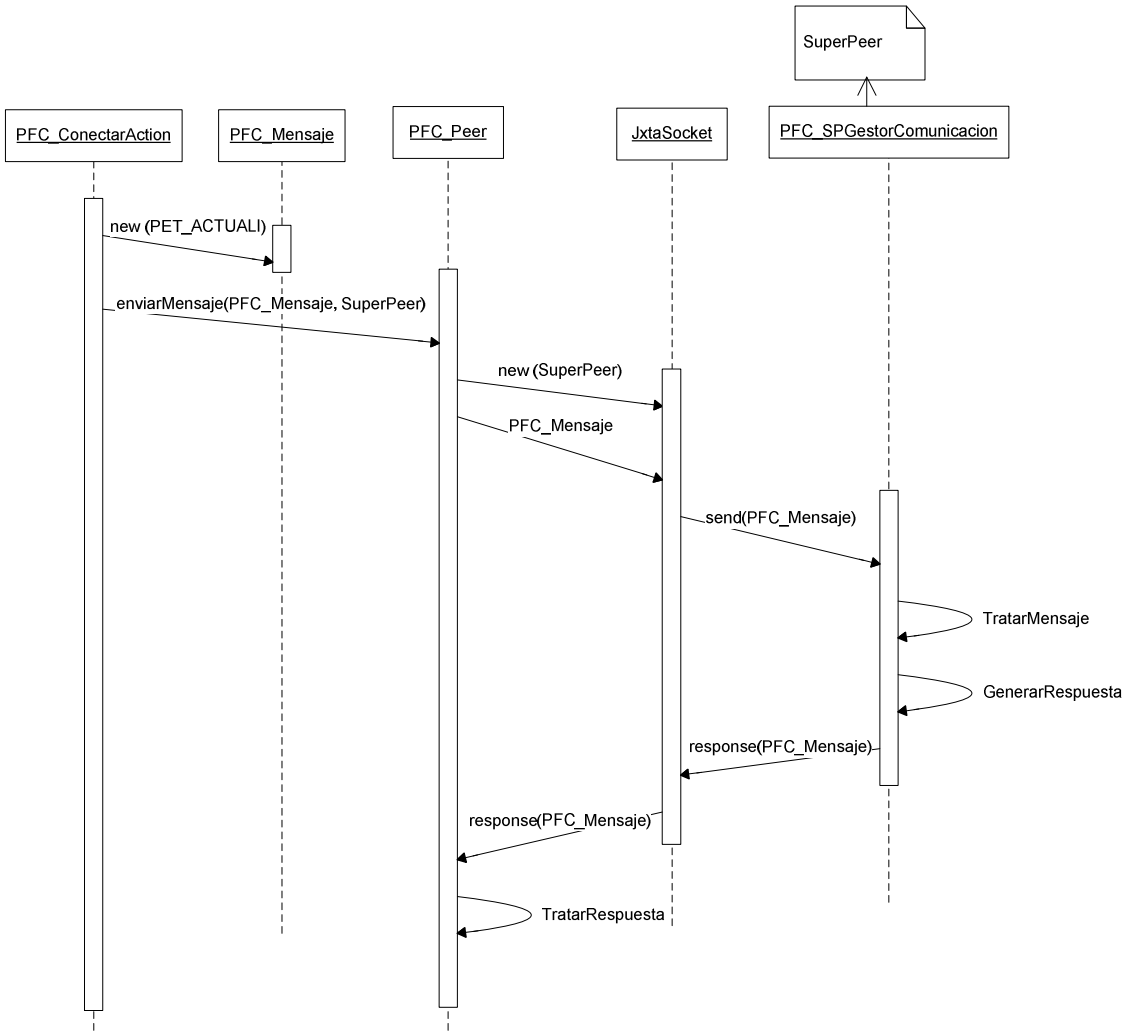
El *peer* tendrá que inicializar los canales específicos para poder comunicarse con el resto de miembros del grupo. Para ello:

- Se creará un Pipe (canal de comunicación) del tipo Multicast que se utilizará para cualquier comunicación tipo Multicast (propagar) entre los miembros del grupo.
- Se creará un socket tipo JxtaMulticastSocket para recibir mensajes del tipo Multicast
- Se lanzará el gestor de comunicaciones (PFC_PGestorComunicacionesMC) que centralizará estas peticiones y se encargará de atender los mensajes recibidos por ese canal. Este proceso es un Thread que se ejecuta siempre y que se queda a la espera de recibir mensajes.
- Se creará un Pipe (canal de comunicación) del tipo Unicast que se utilizará para cualquier comunicación tipo Individual entre dos miembros del grupo.
- Se creará un socket tipo JxtaServerSocket para recibir mensajes del tipo Unicast

- Se lanzará el gestor de comunicaciones (PFC_PGestorComunicaciones) que centralizará estas peticiones y se encargará de atender los mensajes recibidos por ese canal. Este proceso es un Thread que se ejecuta siempre y que se queda a la espera de recibir mensajes.

En este punto se establece que la aplicación ya está conectada a la red. Por ello, su estado pasa a ser *conectado*.

Se inicia una fase de comunicación directa con el *superpeer* para pedirle las actualizaciones que haya podido haber mientras ha estado desconectado:

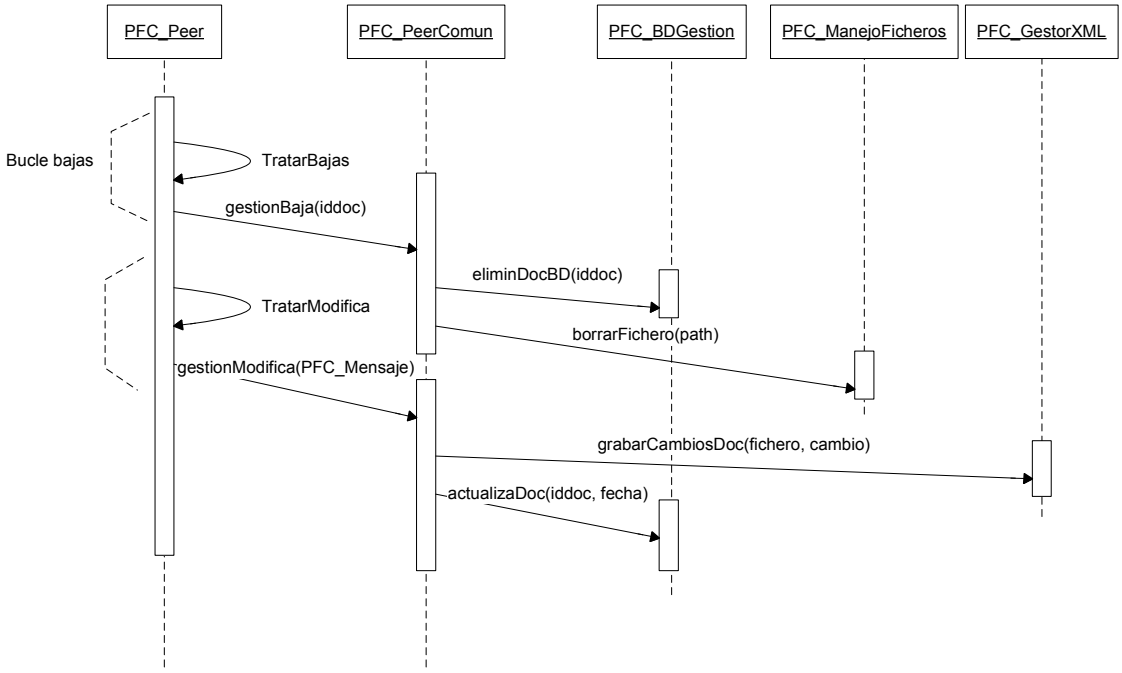


Los pasos a seguir para pedir las actualizaciones disponibles al *superpeer* son:

- Obtener los documentos no propiedad del nodo que en la actualidad tiene censados.

- Construir un PFC_Mensaje tipo PETICION_ACTUALIZACION incluyendo el censo de documentos disponibles que no son propiedad del nodo (se entiende que de los que es propietario ya dispone de la última versión).
- Enviar mensaje al *superpeer* por el canal de comunicación *unicast* (mensaje uno a uno). Esta comunicación se realiza utilizando el identificador único del *superpeer*.
- El *superpeer* recibe el mensaje, procesa que es una PETICION_ACTUALIZACION y obtiene todos los cambios/altas/modificaciones comparando el censo que tiene con el que le ha enviado el *peer*.
- Crea un PFC_Mesanje tipo ACTUALIZACION con los cambios detectados
- Responde por el mismo socket abierto.
- El *peer* recibe el mensaje, detecta que es del tipo ACTUALIZACIÓN y proceso todos los cambios enviados.

En la secuencia anterior se ha visto cómo un *peer* pide al *superpeer* las actualizaciones que haya podido haber. El *superpeer* contesta con todos los cambios y el nodo *peer* los procesa. En la siguiente secuencia veremos con detalle cómo procesa el *peer* todos los cambios indicados por el *superpeer* que son del tipo Baja o Modificación:

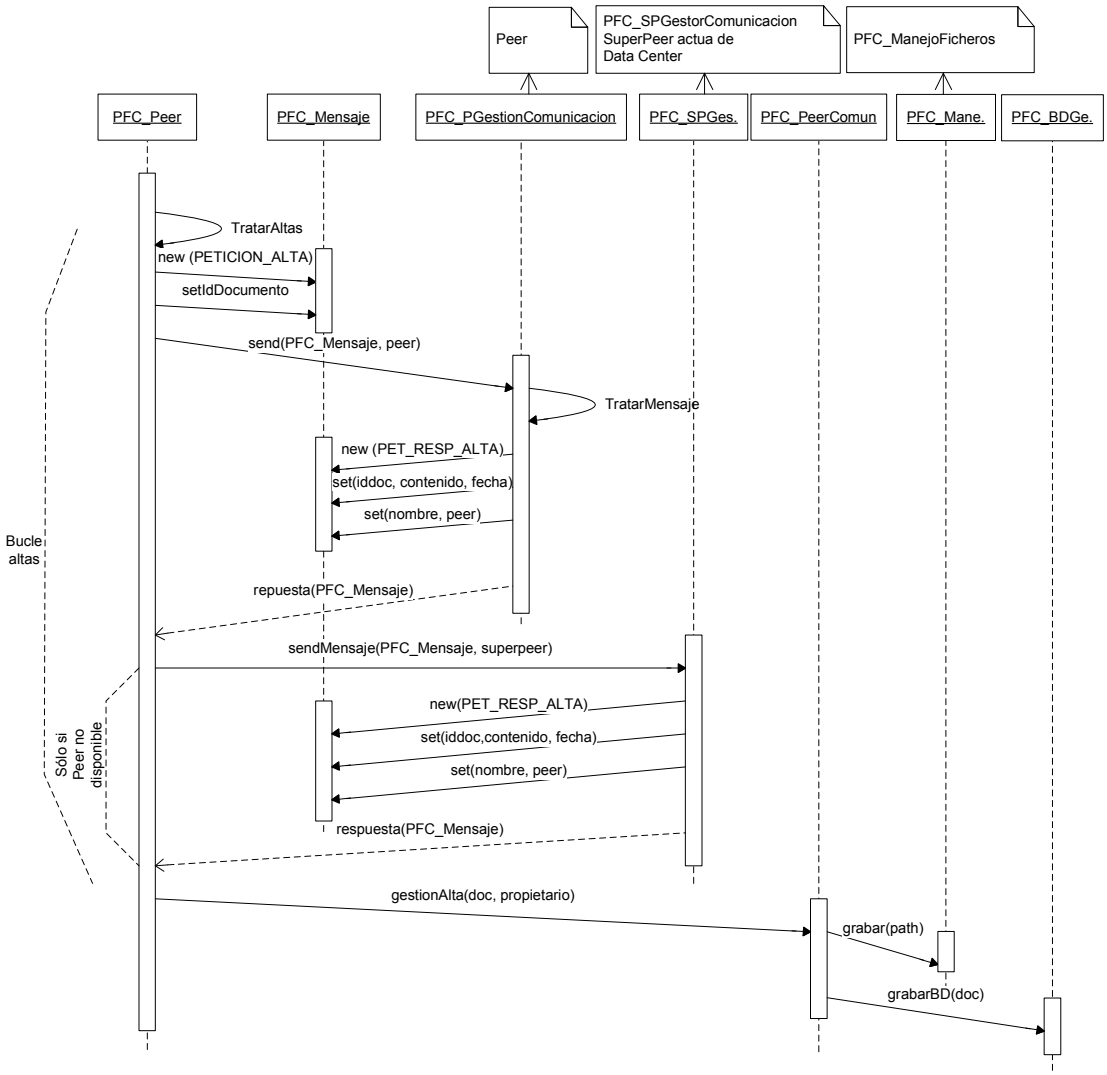


Los pasos a seguir serán:

- Partimos que el *superpeer* ha informado al *peer* de todos los cambios
- *Peer* procesa el mensaje de respuesta. Al tratar las bajas (en caso de que haya), borrará el registro en base de datos y borrará el documento físico
- Si el *superpeer* ha informado de modificaciones, obtendrá la relación de documento/nodo modificado y actualizará el documento únicamente el nodo(s) indicado con el contenido que nos haya informado. Acto seguido actualizará el registro en base de datos del documento para indicar la fecha de la última actualización (esta fecha también viene informada por el *superpeer*). Tener en cuenta que en la modificación se contempla tanto al alta como la baja de un nodo. En tal caso se eliminará o se dará de alta el nodo indicado.

Una vez tratadas las Bajas y Modificaciones, se tratan las Altas.

Las altas son un caso especial, ya que el *superpeer* únicamente te informa del documento y propietario que lo ha dado de alta; pero no del documento físico. Por ello:

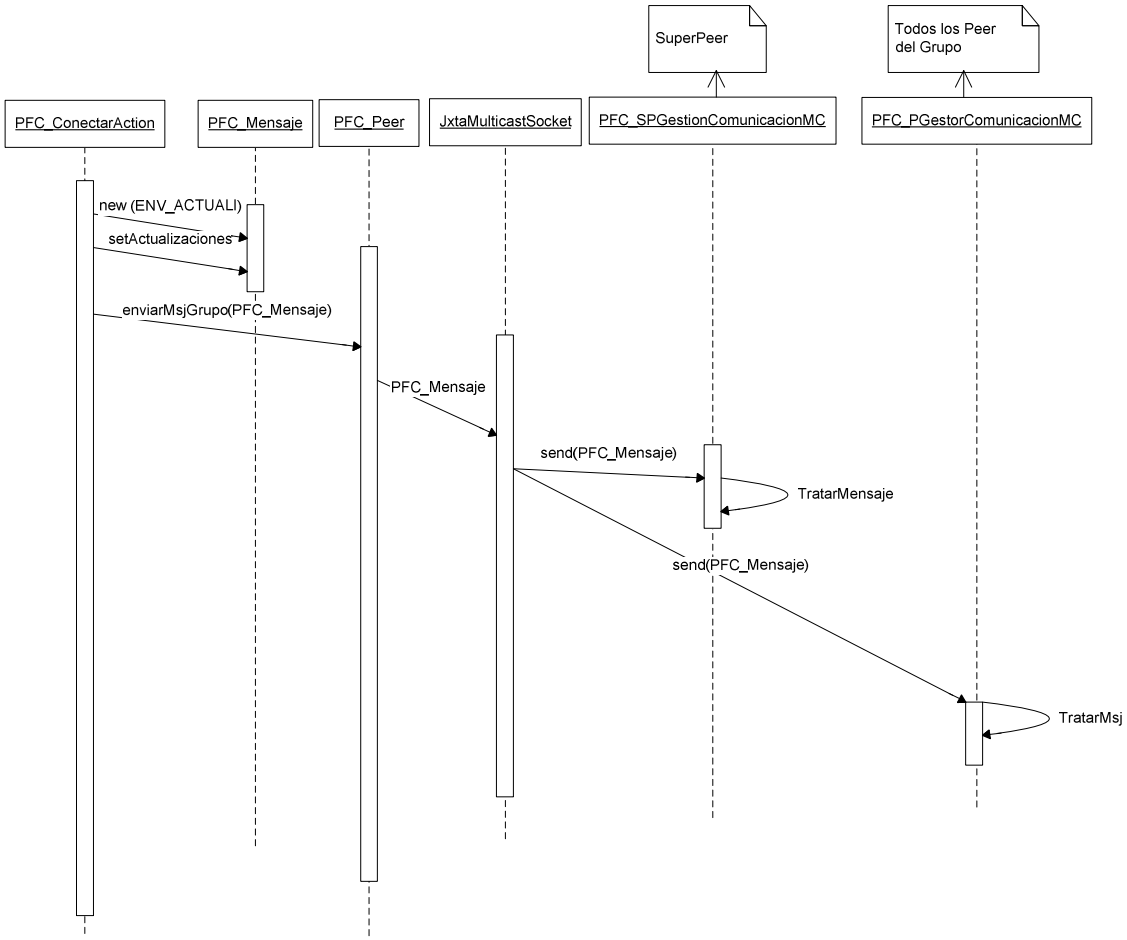


Los pasos a seguir serían:

- Para cada alta que nos hayan informado, se obtendrá el iddocumento y *peer* propietario.
- Con esa información, se monta un PFC_Mensaje tipo TIPO_PETICION_ALTA que se enviará por el canal *Unicast* al *peer* propietario.
- Cuando el *peer* propietario recibe la petición, obtiene el fichero y crea un PFC_Mensaje tipo TIPO_RESPUESTA_ALTA que lo devuelve por el mismo socket de la petición.
- En *peer* que ha realizado la petición, procesa mensaje y dar de alta en base de datos la referencia y graba en el directorio de trabajo el documento.
- Puede suceder que el *peer* propietario del documento no esté disponible. En tal caso, se enviará un mensaje al *data center* (que en nuestro caso es el *superpeer*).

- El *data center (superpeer)* procesará la petición y devolverá un PFC_MENSAJE TIPO_RESPUESTA_ALTA con el documento.
- A partir de ahí se realizarán los pasos para guardar en base de datos y disco.

La última fase de la acción de *Conectar* es informar al resto del grupo de los cambios que haya podido hacer mientras ha estado desconectado:



Los pasos a seguir serían:

- El *peer* que se acaba de conectar obtiene todos los cambios que ha realizado mientras ha estado desconectado y que tiene censados en base de datos. Como cambios también se contemplan las altas y bajas realizadas.
- Crea un PFC_Mensaje TIPO_ENVIO_ACTUALIZACIONES que a su vez tendrá tantos PFC_Mensajes como cambios haya habido por documento. Estos *n* PFC_Mensajes que se crearán serán del tipo NUEVO, BAJA o CAMBIO; en función de si se ha dado de alta, baja o ha modificado un documento.

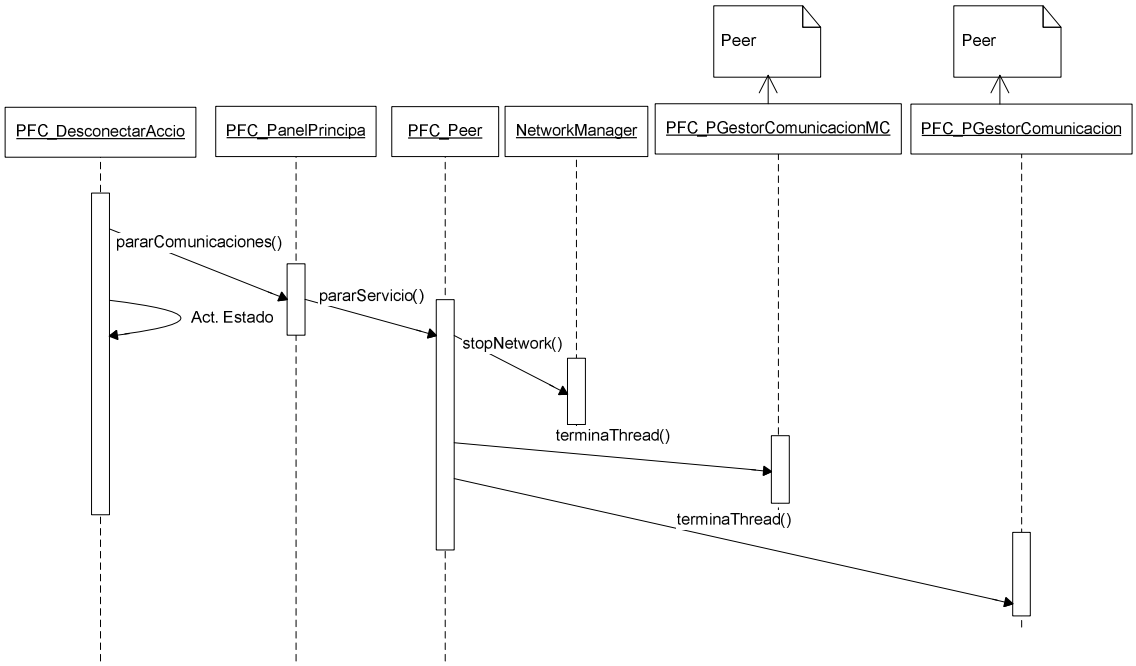
- Utilizando el canal Multicast, se envía el mensaje a todos los miembros del grupo (tanto nodo *superpeer* como nodos *peer*).
- Cuando el *superpeer* recibe el mensaje, obtiene toda la información de cambios y actualiza la base de datos de cambios, de altas y de bajas (borra los registros que se hayan podido indicar).
- A su vez todos los nodo *peer* del grupo (que estén conectado) reciben el mensaje.
- Tratará cada uno de los sub-mensaje como si de un alta, baja o modificación de documento se tratase.

5.6.4 DESCONECTAR – NODO PEER

La forma de llegar a la opción *Desconectar* es la misma que la indicada en el punto anterior, por lo que la forma de cómo cargar la pantalla no se vuelve a explicar.

Se entiende que la aplicación está en modo CONECTADO por lo que el botón de *Conectar* está deshabilitado y el botón *Desconectar* está habilitado.

Cuando se pulse *Desconectar*, la secuencia será:



Los pasos a seguir serían:

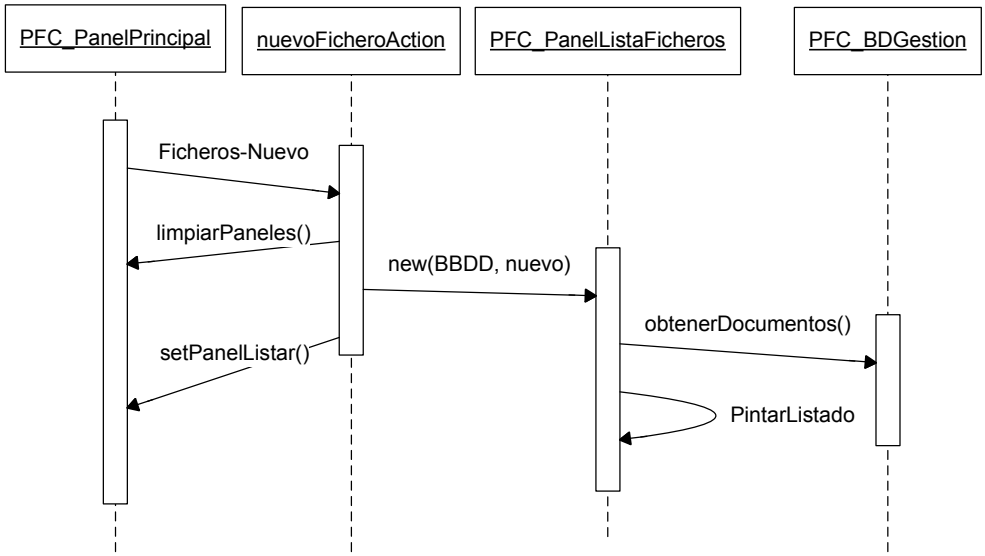
- Al pulsar el botón *desconectar* se indicará al objeto PFC_Peer que cierre las comunicaciones establecidas con el grupo. En primer lugar se desvincula del grupo y cierra sockets de

comunicación abiertos (comunicación tupo *Unicast y Multicast*). Por otro lado, para la comunicación con la red.

- Este mismo objeto también se encargará de terminar el proceso infinito (Thread) de comunicaciones del canal Multicast.
- Este mismo objeto también se encargará de terminar el proceso infinito (Thread) de comunicaciones del canal Unicast.
- Por último actualizará el estado de la aplicación indicando que está en modo DESCONECTADO y habilitará botón *Conectar* y deshabilitará botón *Desconectar*.

5.6.5 CREAR DOCUMENTO XML – NODO PEER

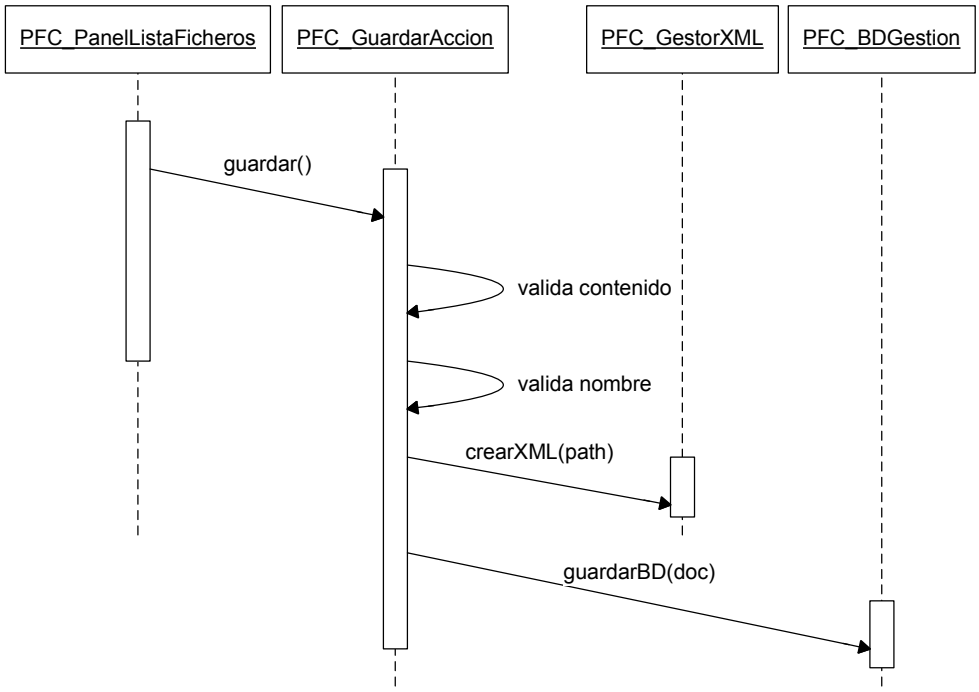
En primer lugar se tiene que realizar la carga de la pantalla una vez el usuario pulse la opción de menú *Ficheros – Nuevo*:



Se debe cargar en el área de trabajo del marco principal la pantalla PFC_PanelListarFicheros y realizar:

- Obtener los documentos censados y mostrarlos en la lista de documentos.
- Habilitar el botón *guardar*
- Deshabilitar el botón *eliminar*

Una vez la pantalla cargada y el usuario haya pulsado el botón *guardar*, se iniciará el proceso de alta de un nuevo documento:

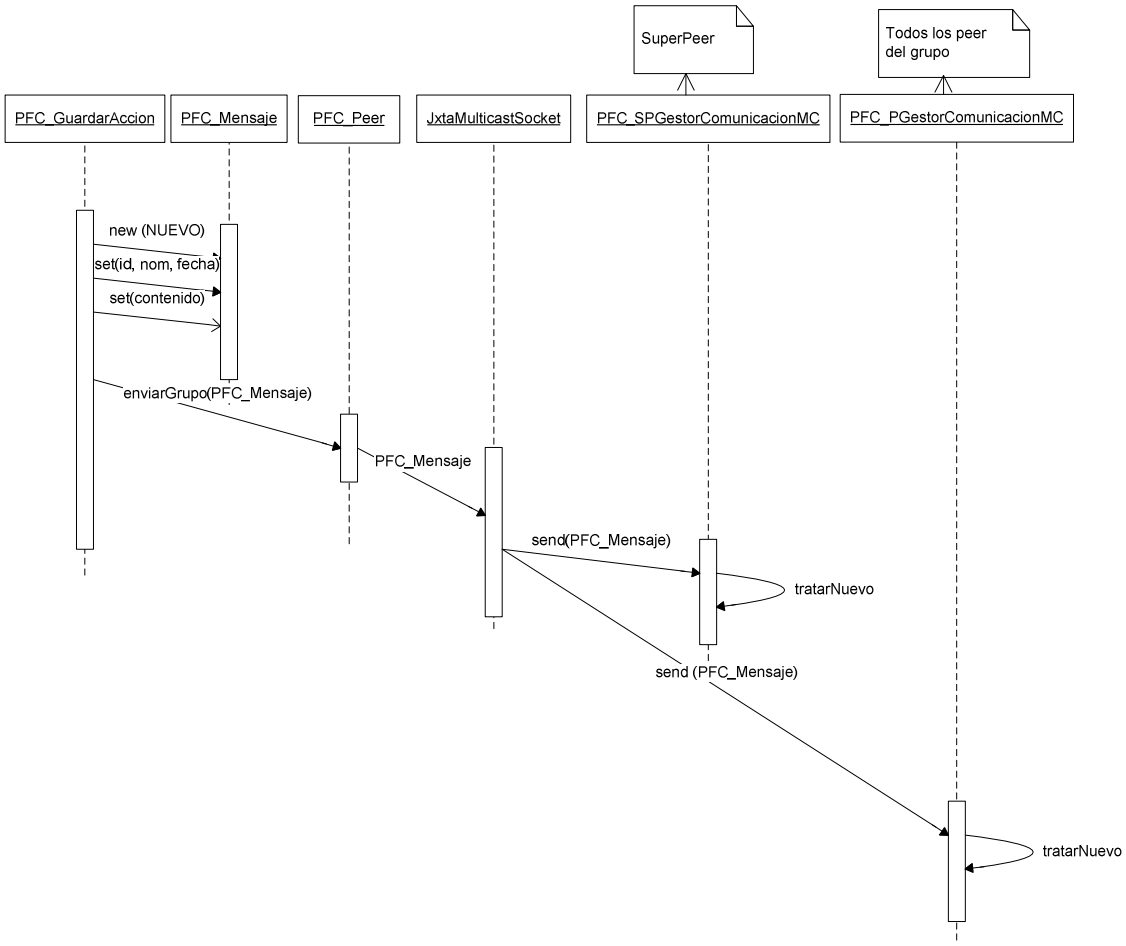


Los pasos a realizar son:

- Validar que el contenido introducido es correcto
- Validar que el nombre introducido es correcto y con extensión xml
- Verificar que ese fichero no existe
- En tal caso, crear el fichero en disco
- Crear el registro en base de datos

A partir de aquí se abren dos caminos: Que el *peer* esté CONECTADO o que no lo esté.

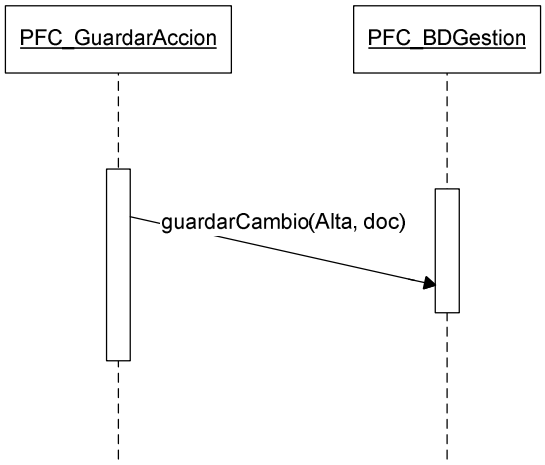
En primer lugar veremos qué se tiene que hacer cuando una vez llegado a este punto, el nodo está CONECTADO:



Los pasos a seguir serían:

- Se crea un PFC_Mensaje del TIPO_NUEVO. Se informa del iddoc, nombre, fecha y contenido del fichero.
- Este mensaje es enviado mediante el canal Multicast al resto de miembros del grupo.
- Cuando el *superpeer* recibe este mensaje, guarda la referencia en base de datos. Como el *superpeer* también actúa de *data center*, a parte de guardar la referencia en base de datos, guarda el fichero en disco.
- Cuando el *peer* recibe este tipo de mensaje, guarda la referencia en base de datos y guarda el fichero en disco.

En segundo lugar vemos qué pasa si el *peer* no está DESCONECTADO:



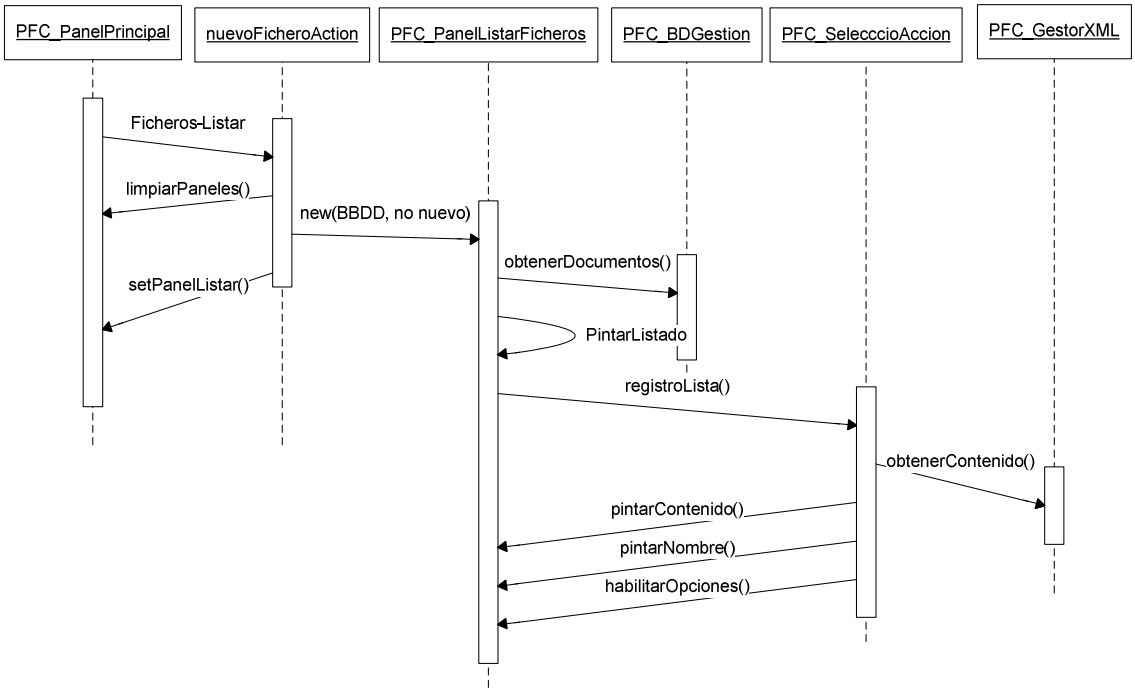
Los pasos a seguir serían:

- Guardar en base de datos los cambios realizados. En este caso se trata de un alta de documento.
- Esto servirá para cuando el nodo se conecte al grupo, saber que existe una alta nueva e informar al grupo de lo mismo.

5.6.6 LISTAR DOCUMENTO – NODO PEER

La pantalla a la que se accede es la misma que en el apartado anterior con la excepción que se indica que entramos en modo no Nuevo, por lo que los botones *guardar y eliminar* estarán deshabilitados (de entrada).

De tal forma que cuando el usuario pulsa la opción *Ficheros – Listar*:

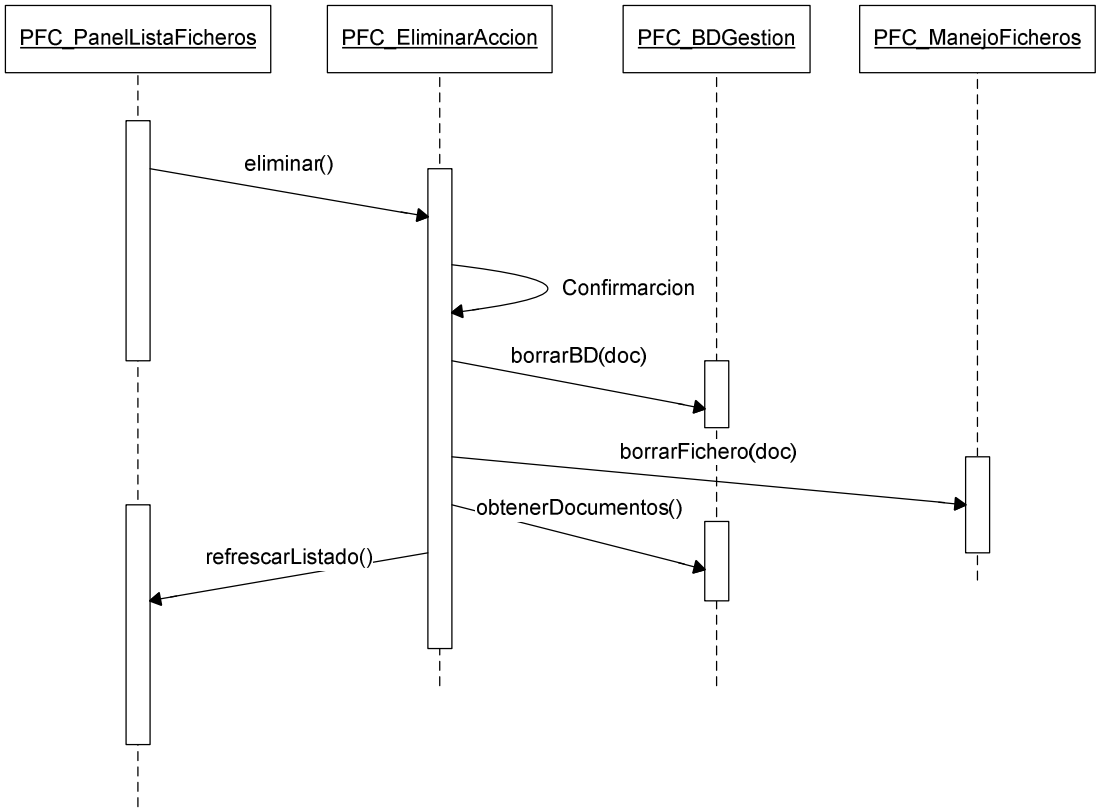


Los pasos a seguir son:

- Se carga pantalla Listar ficheros tal cual se ha visto en el apartado anterior.
- Una vez se han obtenido los documentos censados en base de datos y pintados en el listado, el usuario pulsará sobre uno.
- Se obtiene contenido y nombre del fichero y se vuelcan en el área de edición.
- Se habilitan los botones *guardar y eliminar*.

5.6.7 BAJA DOCUMENTO XML – NODO PEER

Para desencadenar esta acción estamos en el punto que se ha cargado el listado de documentos y seleccionado uno (documento propiedad del nodo). El contenido y nombre se ha volcado en el área de trabajo. A partir de ahí:

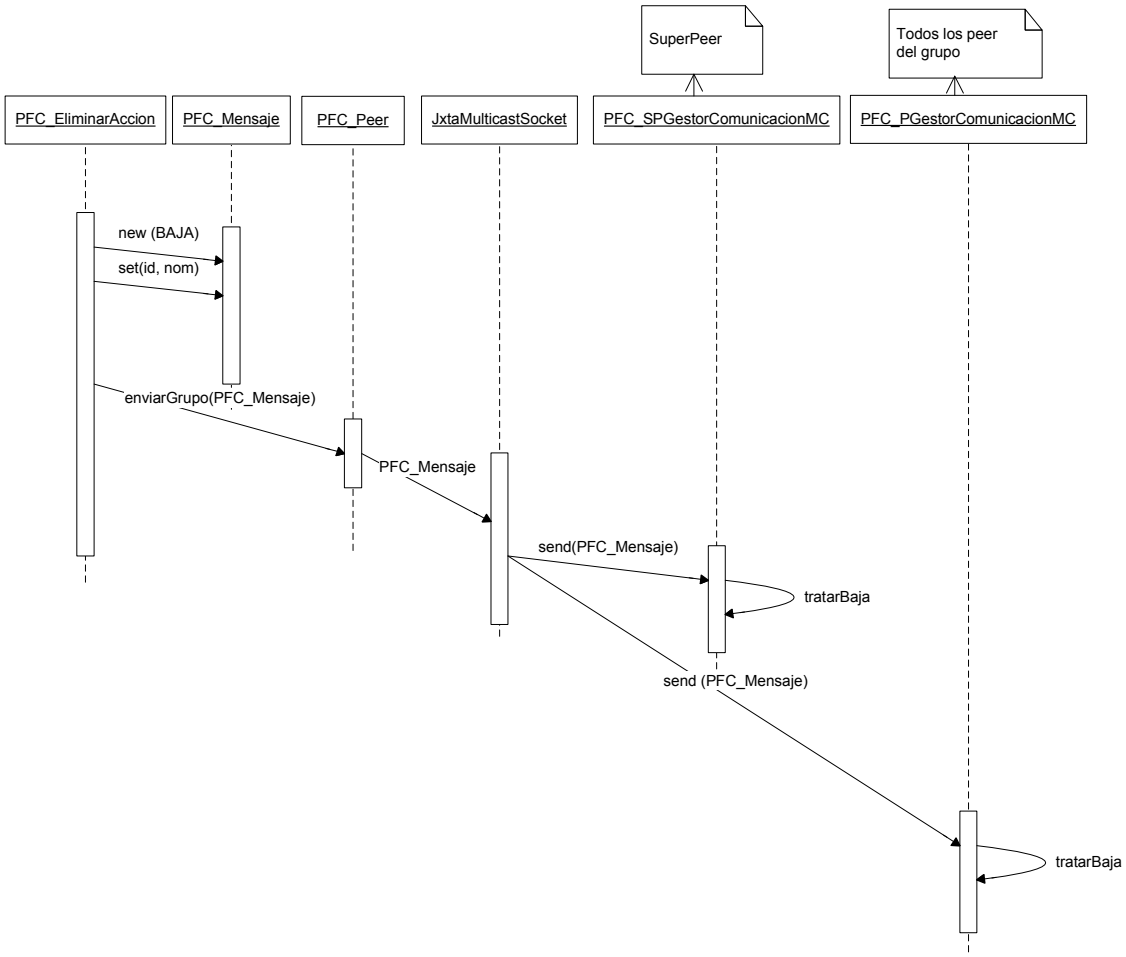


Los pasos a seguir son:

- Una vez pulsado botón *eliminar*, pedir confirmación por parte del usuario
- Con confirmación, borrar registro de base de datos
- Borrar documento del disco
- Obtener de nuevo documentos censados para refrescar listado

A partir de aquí se abren dos caminos: Que el *peer* esté CONECTADO o que no lo esté.

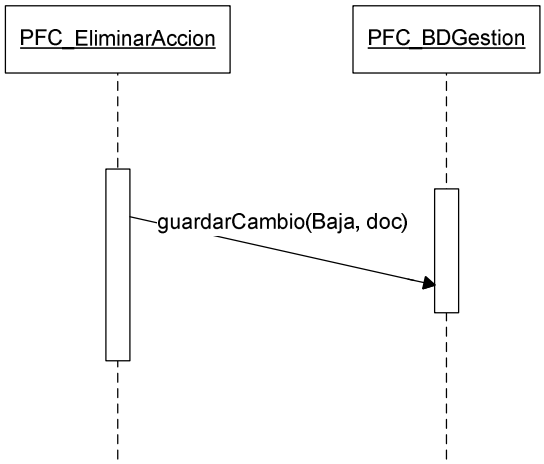
En primer lugar veremos qué se tiene que hacer cuando una vez llegado a este punto, el nodo está CONECTADO:



Los pasos a seguir serían:

- Se crea un PFC_Mensaje del TIPO_BAJA. Se informa del iddoc y nombre del fichero.
- Este mensaje es enviado mediante el canal Multicast al resto de miembros del grupo.
- Cuando el *superpeer* recibe este mensaje, elimina la referencia en base de datos. A parte, y al actuar como *data center*, borra el fichero de disco.
- Cuando el *peer* recibe este tipo de mensaje, elimina la referencia en base de datos y elimina el fichero en disco.

En segundo lugar vemos qué pasa si el *peer* no está DESCONECTADO:

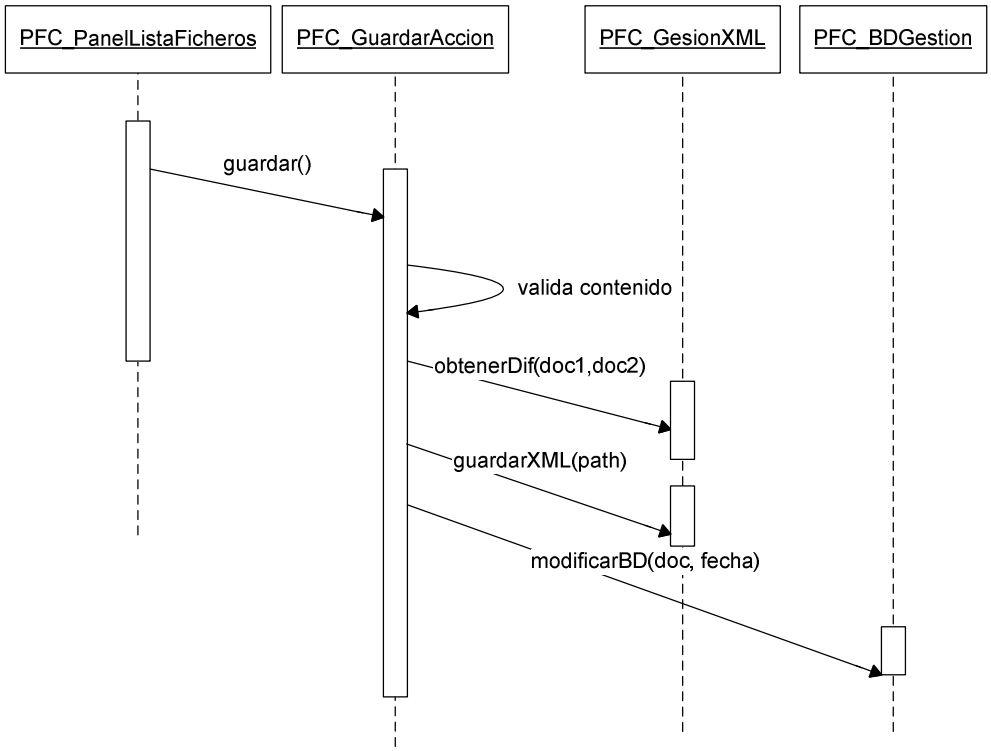


Los pasos a seguir serían:

- Guardar en base de datos los cambios realizados. En este caso se trata de una baja de documento.
- Esto servirá para cuando el nodo se conecte al grupo, saber que existe una baja e informar al grupo de lo mismo.

5.6.8 MODIFICAR DOCUMENTO XML – NODO PEER

Para desencadenar esta acción estamos en el punto que se ha cargado el listado de documentos y seleccionado uno. El contenido y nombre se ha volcado en el área de trabajo. A partir de ahí:

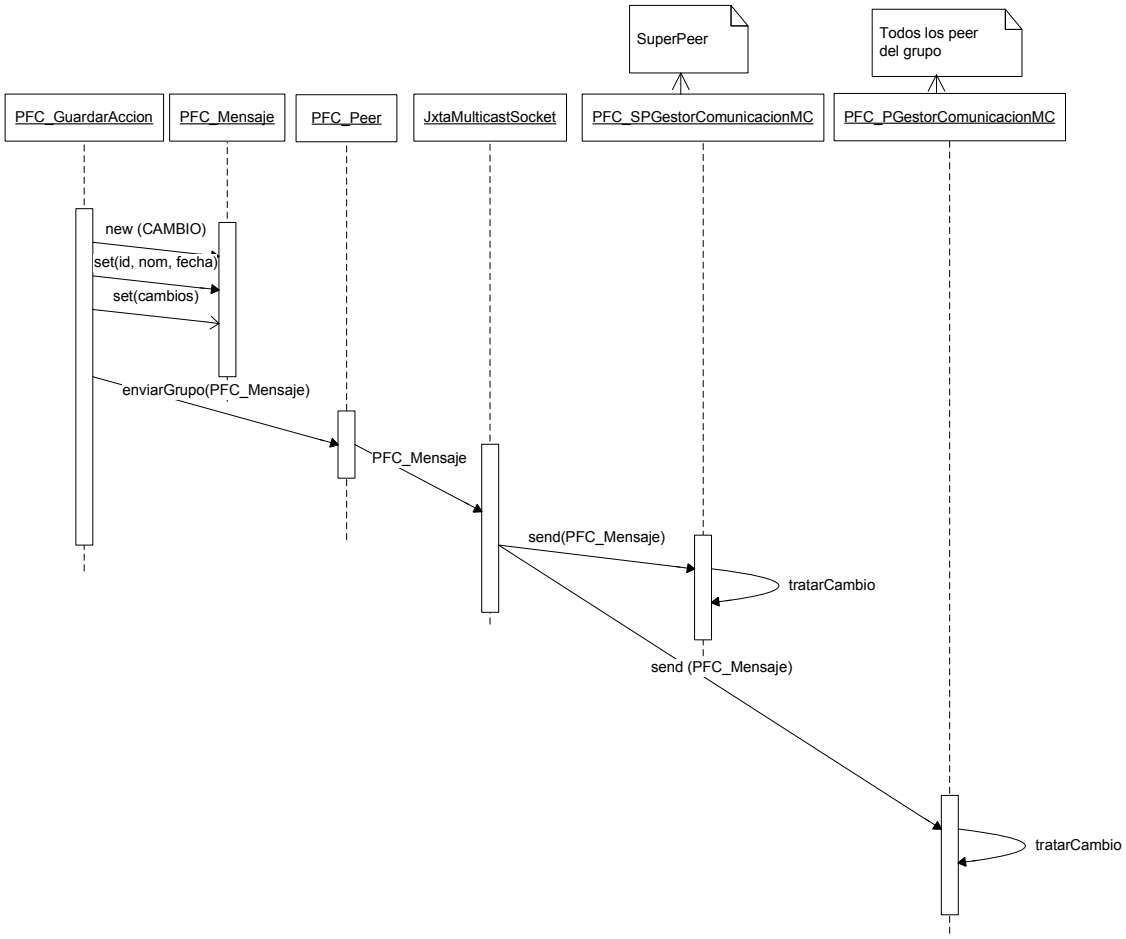


Los pasos a realizar son:

- Validar que el contenido introducido es correcto
- Obtener las diferencias entre el documento original y las modificaciones realizadas. Como diferencias se contemplarán nodos nuevos, nodos dados de baja, modificación del texto de nodos o modificación de los parámetros de los propios nodos.
- Modificar el fichero en disco
- Modificar registro de base de datos indicando la nuevo fecha de actualización

A partir de aquí se abren dos caminos: Que el *peer* esté CONECTADO o que no lo esté.

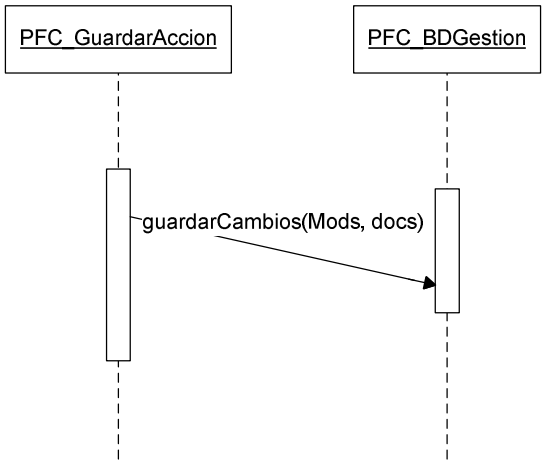
En primer lugar veremos qué se tiene que hacer cuando una vez llegado a este punto, el nodo está CONECTADO:



Los pasos a seguir serían:

- Se crea un PFC_Mensaje del TIPO_CAMBIO. Se informa del iddoc, nombre, fecha y cambios realizados. Los cambios vienen indicados con el path del nodo modificado y si se trata de un Alta(A), Baja(B) o Modificación(M) del nodo. En caso de ser alta o Modificación, se informa del contenido del nodo.
- Este mensaje es enviado mediante el canal Multicast al resto de miembros del grupo.
- Cuando el *superpeer* recibe este mensaje, guarda los cambios recibidos y actualiza la fecha del documento.
- Cuando el *peer* recibe este tipo de mensaje, actualiza la fecha del documento y guarda los cambios indicados en el fichero.

En segundo lugar vemos qué pasa si el *peer* no está DESCONECTADO:

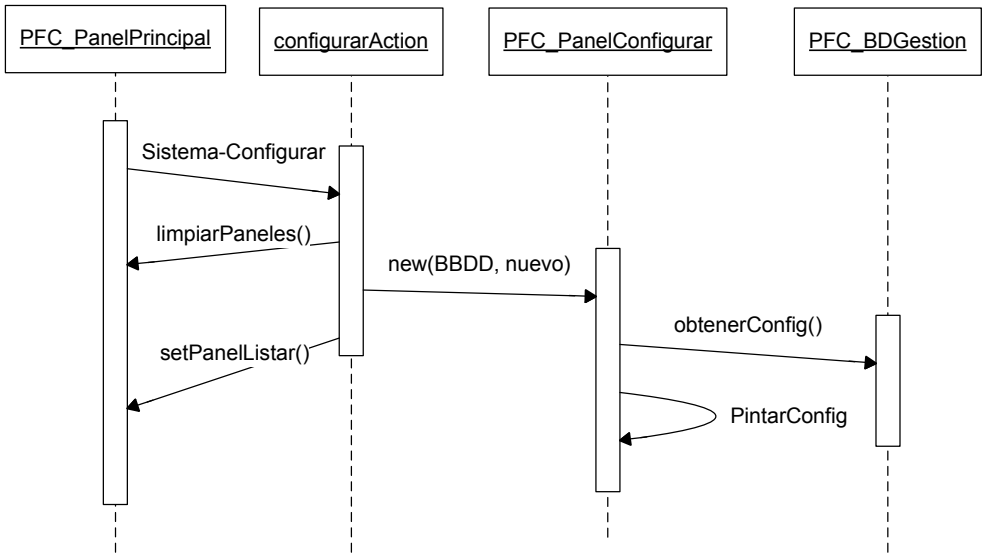


Los pasos a seguir serían:

- Guardar en base de datos los cambios realizados. En este caso se trata de una serie de cambios realizados en el contenido del fichero. Estos cambios se almacenan con referencia el path del nodo e indicando si es una Alta(A), Baja(B) o Modificación(M) del nodo en cuestión. Si se trata de una Alta o Modificación a parte se almacena el contenido del nodo.

5.6.9 CONFIGURACIÓN – NODO PEER

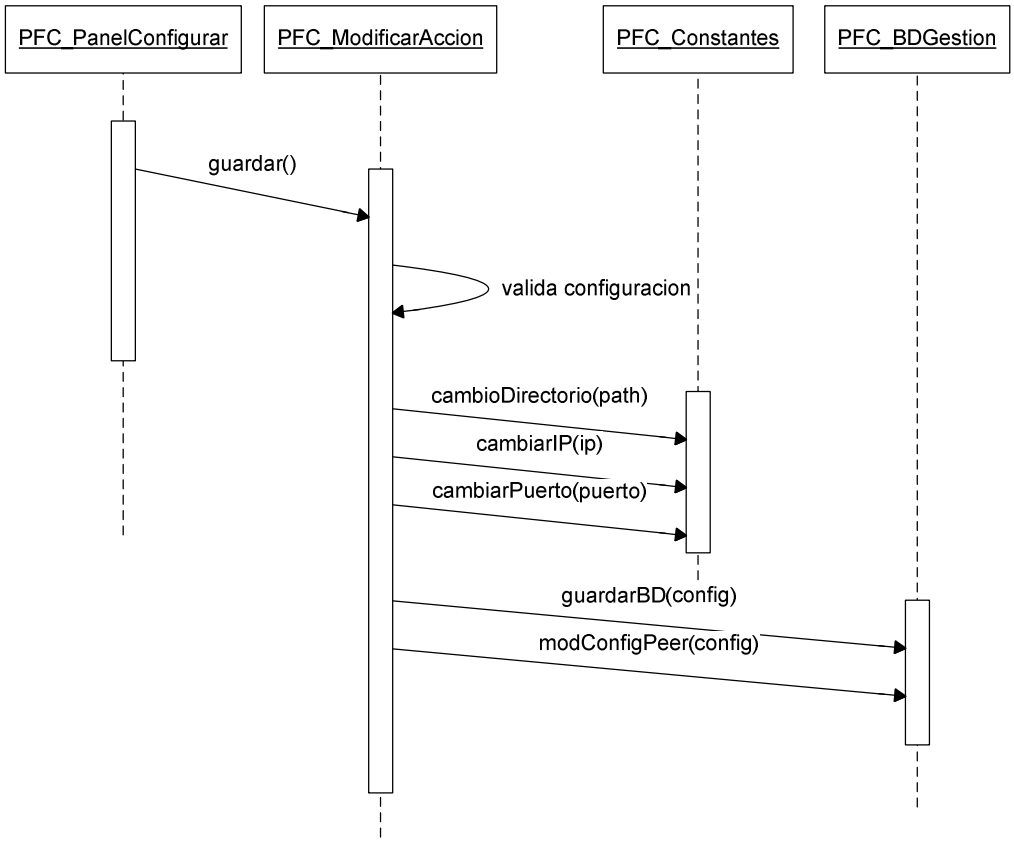
En primer lugar se tiene que realizar la carga de la pantalla una vez el usuario pulse la opción de menú *Sistema – Configurar*:



Se debe cargar en el área de trabajo del marco principal la pantalla PFC_PanelConfigurar y realizar:

- Obtener la configuración y mostrarla el área de trabajo.

Una vez la pantalla cargada y el usuario haya pulsado el botón *guardar*, se iniciará el proceso de modificación de los parámetros de configuración:



Los pasos a seguir son:

- Verificar si se ha modificado el parámetro que indica el directorio de trabajo. En tal caso, verificar que se ha introducido un valor correcto y actualizar este parámetro en la configuración de la aplicación. Verificar si existe el nuevo directorio y en caso contrario, crearlo.
- Verificar si se ha modificado la IP del *superpeer*. En tal caso, comprobar que sea una IP con formato correcto. Actualizar este parámetro en la configuración de la aplicación y activar el flag que indica que la próxima vez que el nodo se conecte a la red (grupo), tiene que actualizar la configuración local comunicación *peer to peer* para tratar la nueva IP de enlace con *superpeer*.
- Verificar si se ha modificado el puerto del *superpeer*. En tal caso, comprobar que sea un puerto con formato correcto y que esté en el rango 9701 - 9799. Actualizar este parámetro en la configuración de la aplicación y activar el flag que indica que la próxima vez que el nodo se conecte a la red (grupo), tiene que actualizar la configuración local comunicación *peer to peer* para tratar el nuevo puerto de enlace con *superpeer*.

5.6.10 SALIR – NODO PEER

Esta acción lo que hará es enlazar con el *desconectar* e inmediatamente cerrar la aplicación.

5.7 LÓGICA DE NEGOCIO

5.7.1 INTRODUCCIÓN

Lo descrito en operaciones de negocio son básicamente todas aquellas acciones que requieren de un punto inicial por parte del usuario.

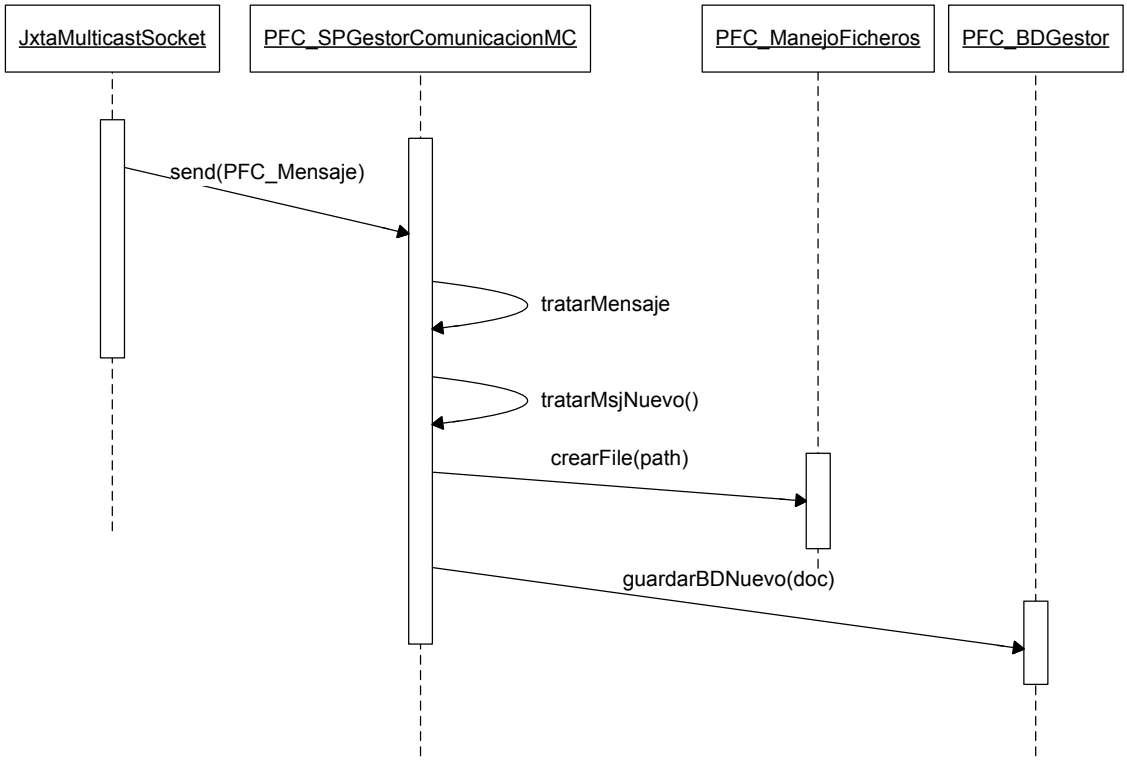
En Lógica de negocio se detallarán todas aquellas operaciones que aunque también están vinculadas a una acción realizada por el usuario, su vinculación es algo más indirecta.

En este punto veremos cómo se tratan los distintos mensajes que son enviados al grupo pero sin tener en cuenta la acción del nodo origen que la genera; o sea, todos los otros nodos del grupo que están a la espera de recibir mensajes donde se les indique que determinado documento ha sido modificado.

5.7.2 RECEPCIÓN ALTA DOCUMENTO – NODO SUPERPEER

Operación de negocio vinculada: Crear documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte del nodo *superpeer*.



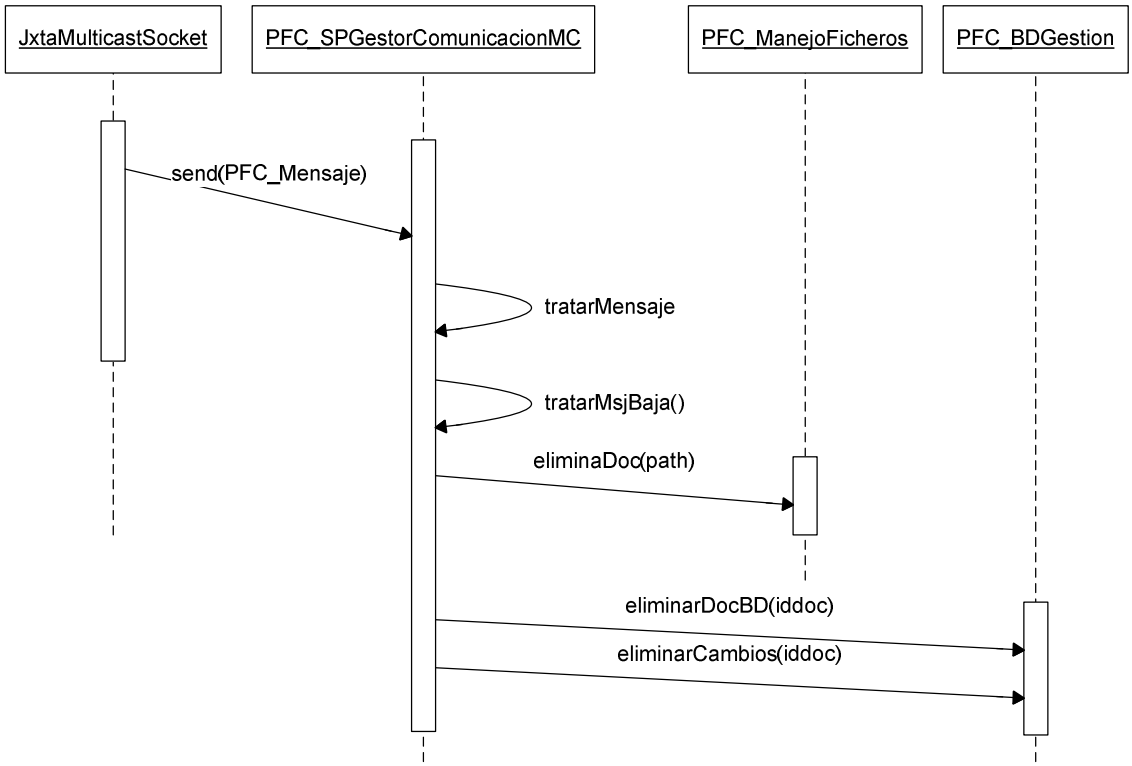
Los pasos a seguir son:

- El *superpeer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_NUEVO
- Se da de alta el registro en base de datos indicando iddoc, nombre, propietario y fecha de creación.
- Al trabajar como *data center*, también almacena el fichero en disco

5.7.3 RECEPCIÓN BAJA DOCUMENTO – NODO SUPERPEER

Operación de negocio vinculada: Baja documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte del nodo *superpeer*.



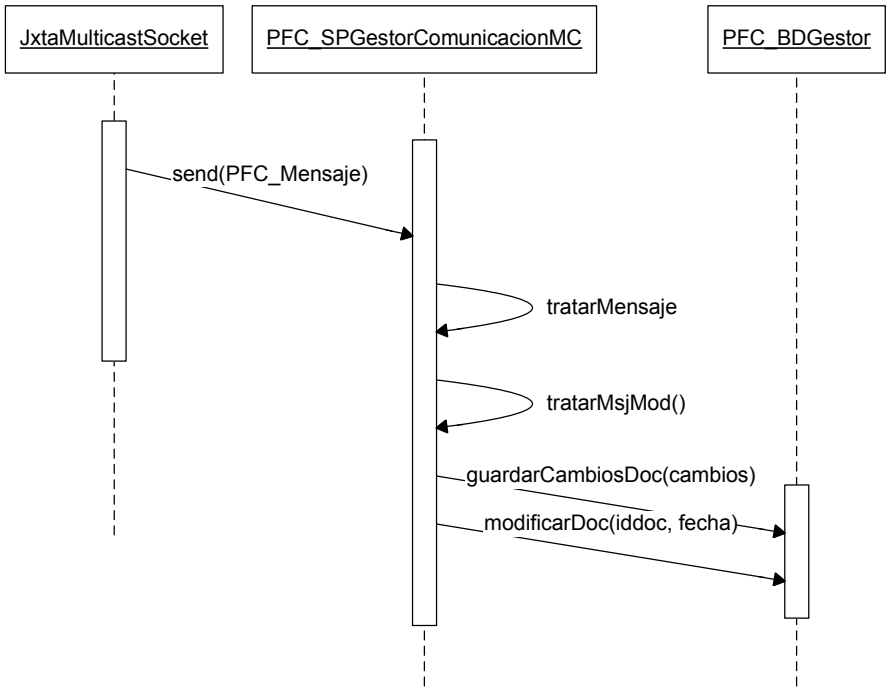
Los pasos a seguir son:

- El *superpeer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_BAJA
- Al trabajar como *data center*, eliminar el documento de disco.
- Elimina el registro del documento en base de datos.
- Elimina los registros de cambios (DOC_CAMBIOS) que pudiese tener.

5.7.4 RECEPCIÓN MODIFICAR DOCUMENTO – NODO SUPERPEER

Operación de negocio vinculada: Modificar documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte del nodo *superpeer*.



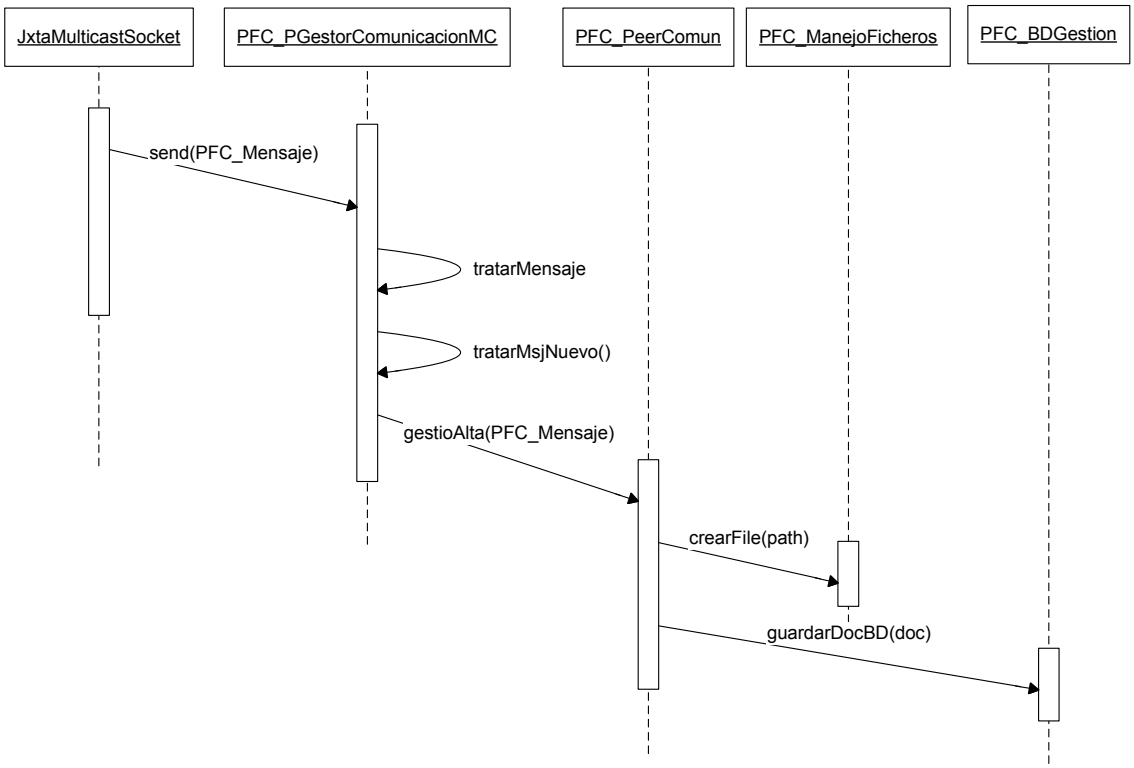
Los pasos a seguir son:

- El *superpeer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica la tipología del mensaje. En este caso el mensaje recibido es TIPO_CAMBIO
- Guarda en base de datos todos los cambios realizados sobre el documento (DOC_CAMBIOS).
- Modifica el registro del documento (DOCUMENTOS) para establecer la nueva fecha de actualización informada.

5.7.5 RECEPCIÓN ALTA DOCUMENTO – NODO PEER

Operación de negocio vinculada: Crear documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte de un nodo *peer*.



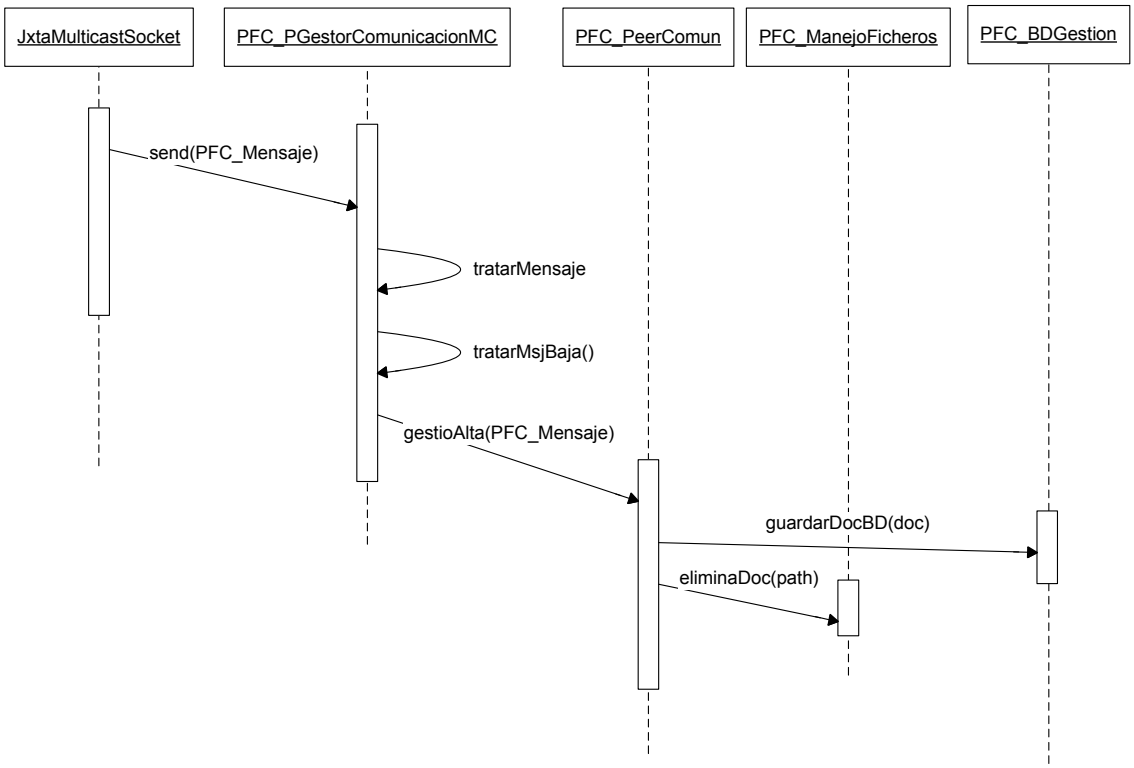
Los pasos a seguir son:

- El *peer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica si el *peer* que ha enviado el mensaje no es él mismo; en tal caso descarta el mensaje.
- Verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_NUEVO
- Crea en disco el fichero con los datos enviados en el mensaje
- Guarda en base de datos la referencia del documento.

5.7.6 RECEPCIÓN BAJA DOCUMENTO – NODO PEER

Operación de negocio vinculada: Baja documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte de un nodo *peer*.



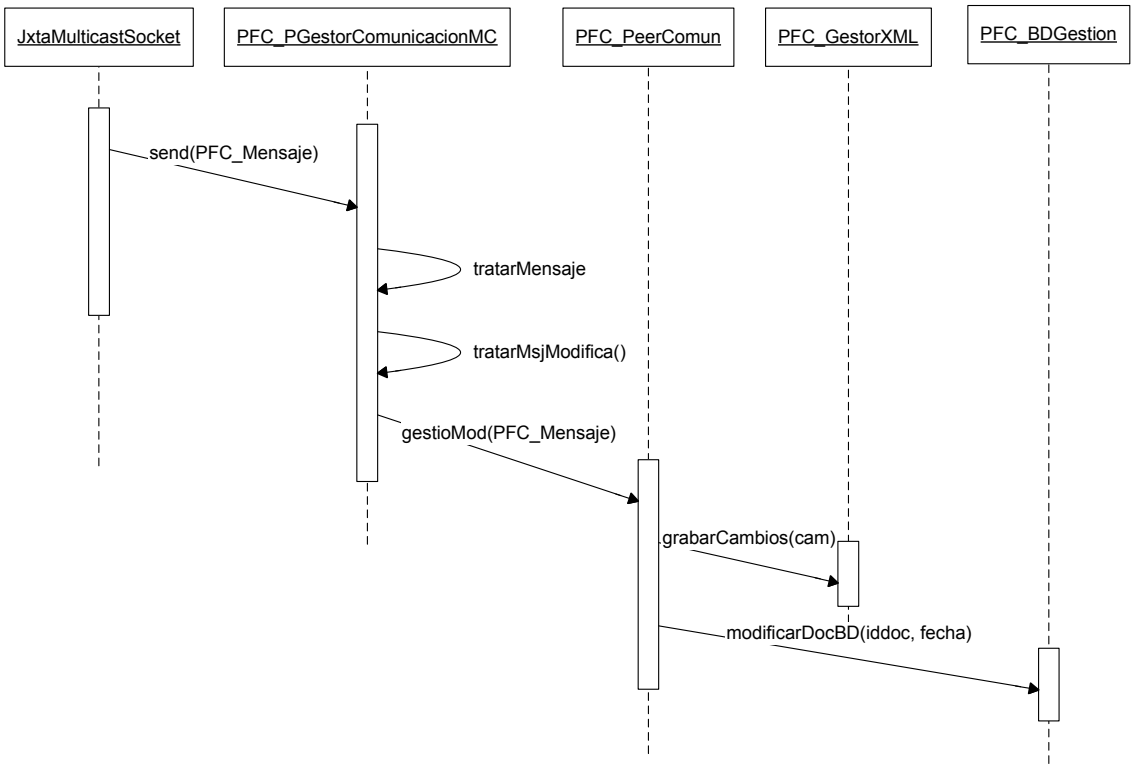
Los pasos a seguir son:

- El *peer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica si el *peer* que ha enviado el mensaje no es él mismo; en tal caso descarta el mensaje.
- Verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_BAJA
- Elimina el fichero en disco.
- Elimina el registro de base de datos del documento.

5.7.7 RECEPCIÓN MODIFICAR DOCUMENTO – NODO PEER

Operación de negocio vinculada: Modificar documento XML

Tipo de mensaje enviado Multicast. Recepción del mensaje por parte de un nodo *peer*.



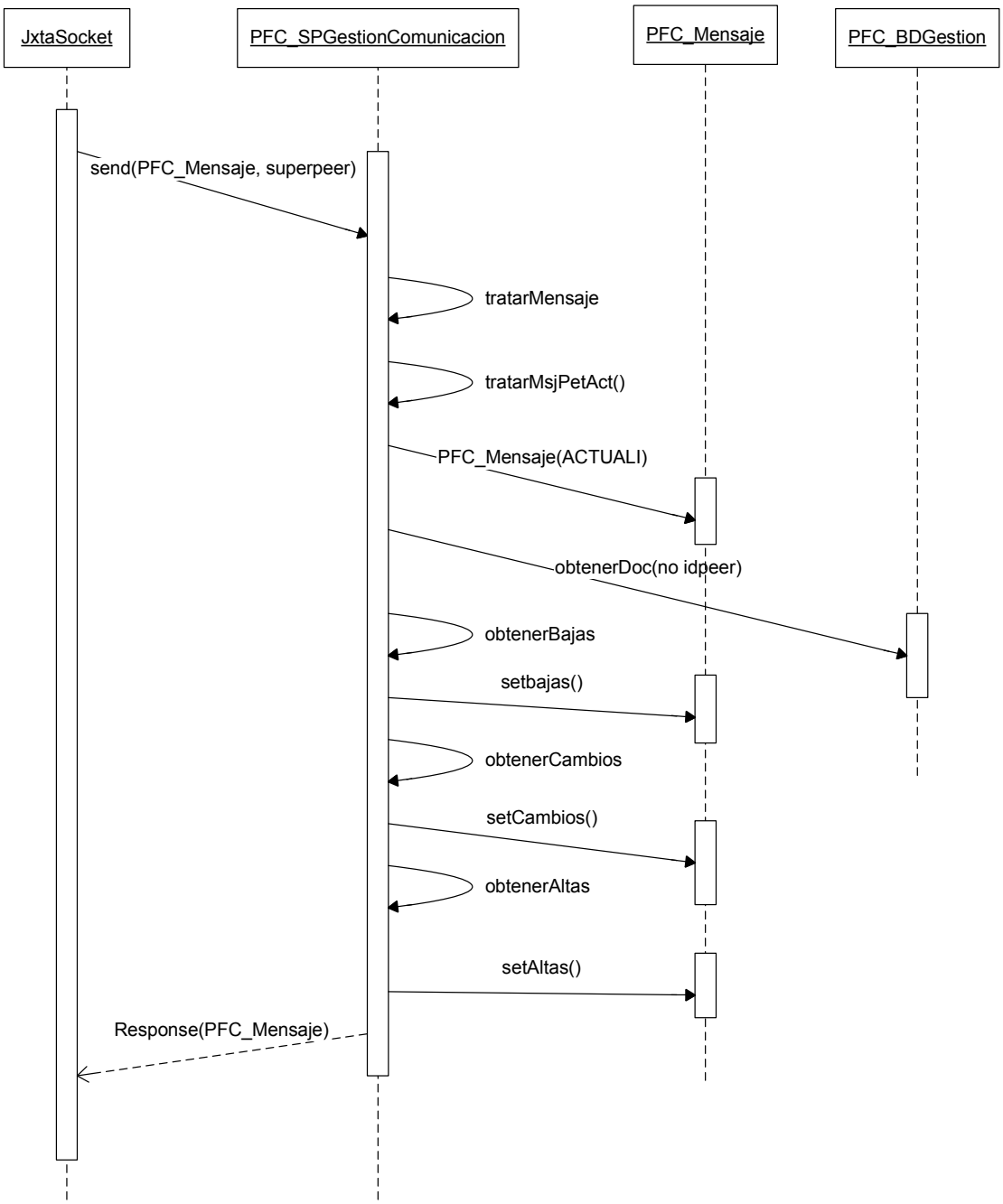
Los pasos a seguir son:

- El *peer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje, verifica si el *peer* que ha enviado el mensaje no es él mismo; en tal caso descarta el mensaje.
- Verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_CAMBIO
- Realiza todos los cambios indicados en el fichero. Estos cambios pueden ser: Alta(A) de un nodo, Baja(B) de un nodo, Modificación(M) de un nodo. En el caso de la Alta y la modificación, a parte del path del nodo, se envía el contenido del mismo.
- Actualiza el registro en base de datos del documento para indicar la nueva fecha de actualización.

5.7.8 PETICIÓN ACTUALIZACIONES – NODO SUPERPEER

Operación de negocio vinculada: Conectar

Tipo de mensaje enviado Unicast por parte de un nodo que se acaba de conectar al grupo y le pide las posibles actualizaciones que haya podido haber. Recepción del mensaje por parte del nodo *superpeer*.



Los pasos a seguir son:

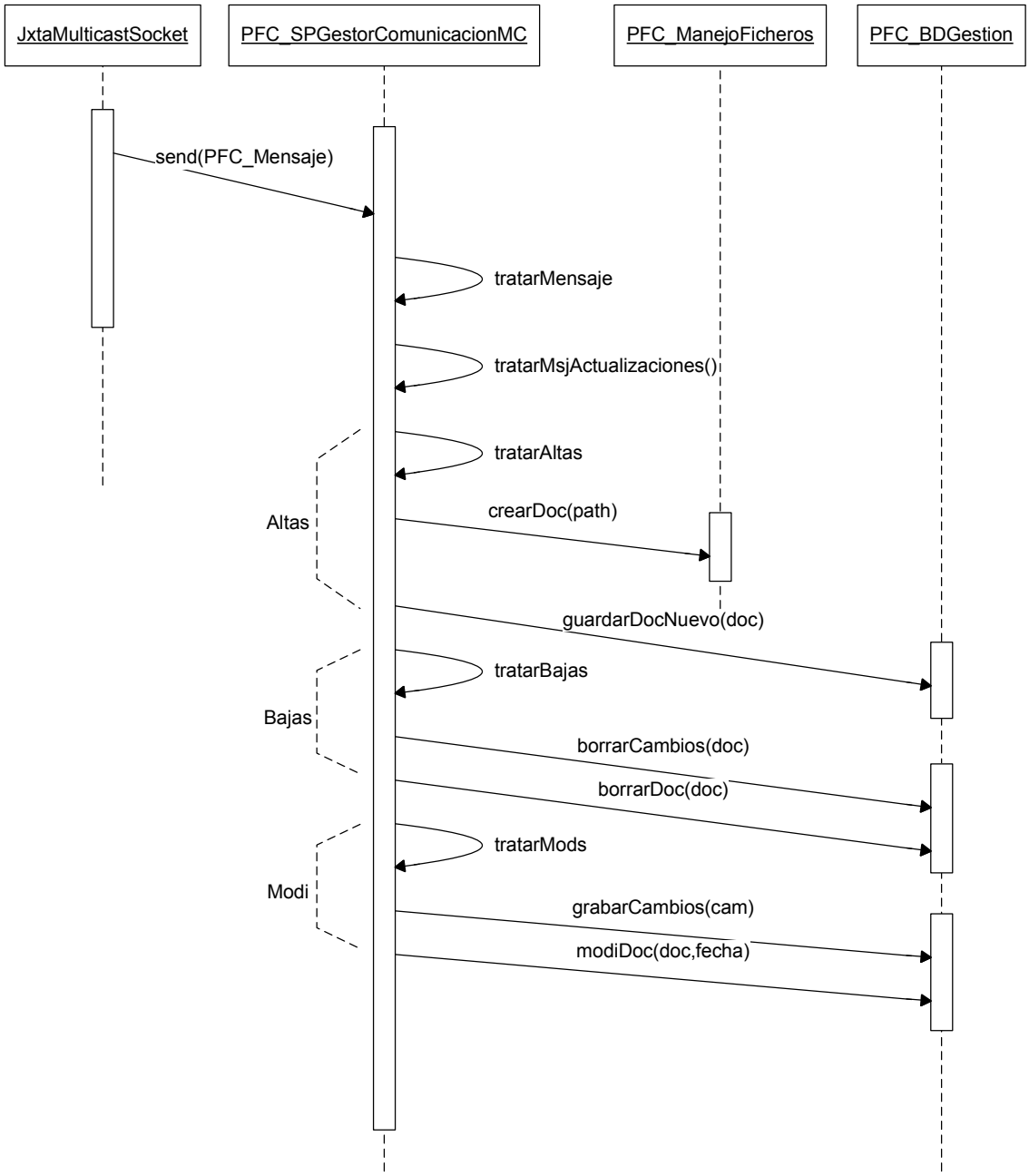
- El *superpeer* está a la espera de recibir mensajes por el canal Unicast.

- Cuando recibe un mensaje verifica la tipología del mismo. En este caso se trata de un PFC_Mensaje TIPO_PETICION_ACTUALIZACION.
- Obtiene todos los documentos que tiene censados que no son propiedad del *peer* peticionario
- Compara ese censo con la lista de documentos que el *peer* le ha informado que tiene (y que no son de su propiedad).
- De esa comparación saldrán Altas, Bajas y modificaciones. Las modificaciones las detecta por la fecha de los documentos.
- Por cada cambio detectado crea un PFC_Mensaje TIPO_CAMBIO como si de un cambio normal de un documento se tratase.
- Por cada baja crea un PFC_Mensaje TIPO_BAJA.
- Y por cada alta crea un PFC_Mensaje TIPO_AVISO_ALTA. Lo que significa que el *superpeer* únicamente informa que existe un documento nuevo y quién es su propietario.
- Si detecta que ha habido una alta y baja del mismo documento en el periodo de tiempo, no envía nada.
- Con un PFC_Mensaje creado con *n* PFC_Mensaje contesta la petición llegada desde un *peer*.

5.7.9 RECEPCIÓN ACTUALIZACIONES – NODO SUPERPEER

Operación de negocio vinculada: Conectar

Tipo de mensaje enviado Multicast por parte de un nodo que se acaba de conectar al grupo informando de todos los cambios realizados en sus documentos mientras ha estado desconectado. Recepción del mensaje por parte del nodo *superpeer*.



Los pasos a seguir son:

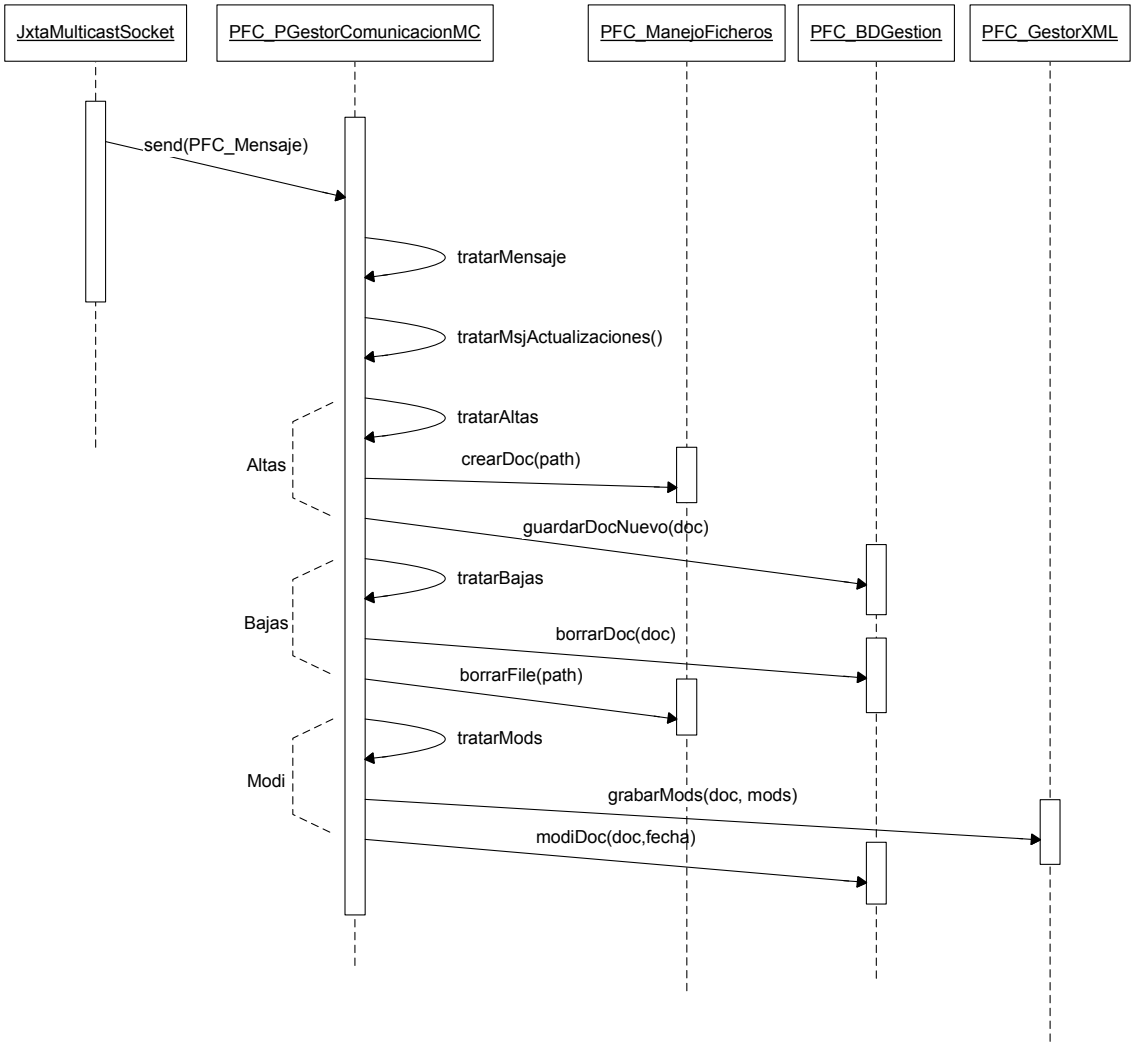
- El *superpeer* está a la espera de recibir mensajes por el canal Multicast.
- Cuando recibe un mensaje verifica la tipología del mismo. En este caso se trata de un PFC_Mensaje TIPO_ENVIO_ACTUALIZACIONES.
- Este mensaje contiene todas las actualizaciones que un *peer* ha realizado mientras estaba desconectado.

- Se tratan las modificaciones por bloques; Altas, Bajas y Modificaciones y por cada una de ellas se sigue el mismo proceso como si se tratase de forma individual.

5.7.10 RECEPCIÓN ACTUALIZACIONES – NODO PEER

Operación de negocio vinculada: Conectar

Tipo de mensaje enviado Multicast por parte de un nodo que se acaba de conectar al grupo informando de todos los cambios realizados en sus documentos mientras ha estado desconectado. Recepción del mensaje por parte de un nodo *peer*.



Los pasos a seguir son:

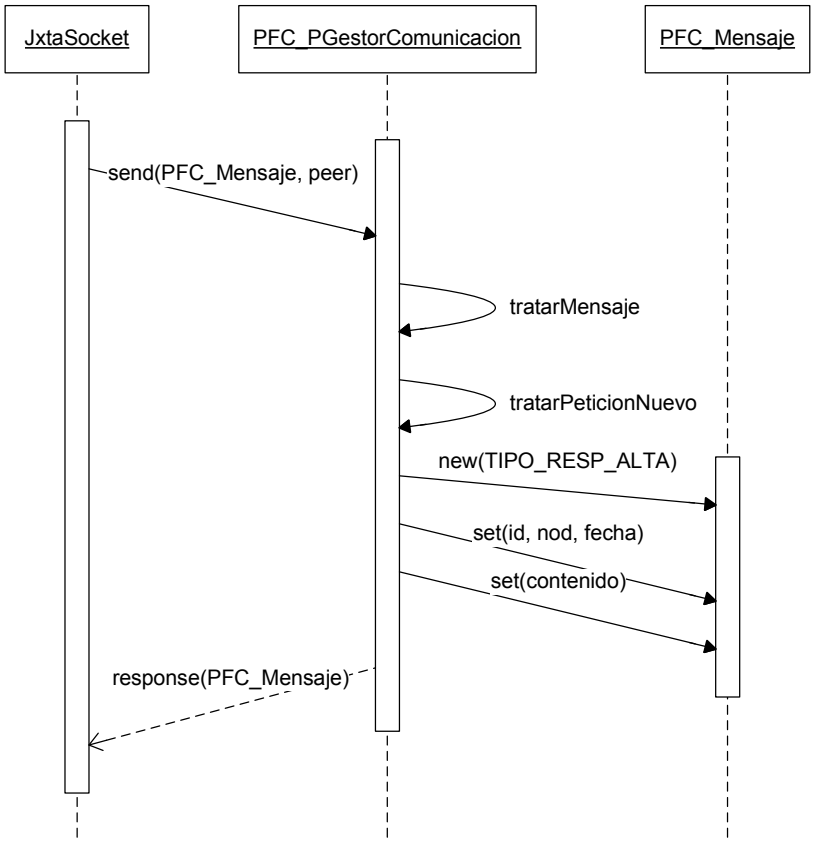
- El *peer* está a la espera de recibir mensajes por el canal Multicast.

- Cuando recibe un mensaje, verifica si el *peer* que ha enviado el mensaje no es él mismo; en tal caso descarta el mensaje.
- Verifica la tipología de mensaje. En este caso el mensaje recibido es TIPO_ENVIO_ACTUALIZACIONES
- Gestiona los cambios por bloques y de igual forma que si lo hiciera de forma individual.

5.7.11 PETICIÓN DAME NUEVO – NODO PEER

Operación de negocio vinculada: Conectar

Tipo de mensaje enviado Unicast por parte de un nodo que al conectarse el *superpeer* le ha informado que hay un documento nuevo cuyo propietario es el *peer* que recibe el mensaje. El objetivo de este mensaje es que el *peer creador* le envíe el nuevo documento. Recepción del mensaje por parte de un nodo *peer*.



Los pasos a seguir son:

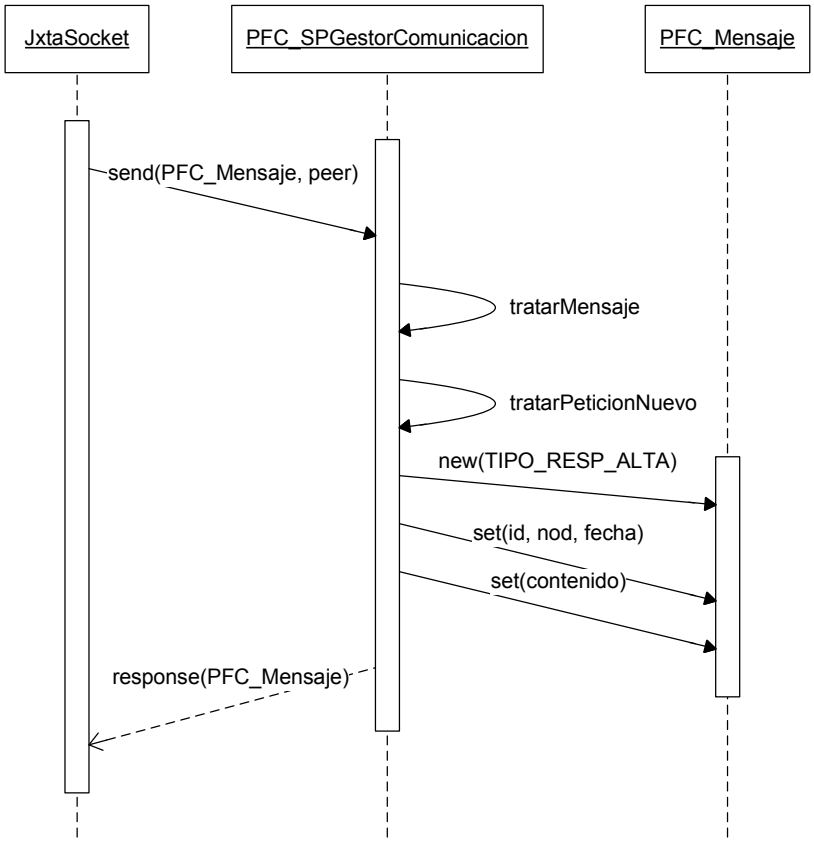
- El *peer* está a la espera de recibir mensajes por el canal Unicast.

- Recibe un PFC_Mensaje del TIPO_PETICION_ALTA.
- Crea un PFC_Mensaje del TIPO_RESPUESTA_ALTA donde incluye el contenido del documento y los datos como id, nombre, fecha.
- Responde utilizando el mismo socket el PFC_Mensaje creado.

5.7.12 PETICIÓN DAME NUEVO – NODO SUPERPEER (DATA CENTER)

Operación de negocio vinculada: Conectar

Tipo de mensaje enviado Unicast por parte de un nodo que al conectarse el *superpeer* le ha informado que hay un documento nuevo cuyo propietario es un *peer* determinado. Cuando el *peer* que necesita el documento ha intentado pedir al *peer creador* el documento no ha podido obtenerlo por no estar disponible dicho *peer*. En ese instante, envía un mensaje al *data center*, que en nuestro caso es el *superpeer* para que le envíe el documento en cuestión. Recepción del mensaje por parte del nodo *superpeer* en sus funciones de *data center*.



Los pasos a seguir son:

- El *superpeer* (en sus funciones de *data center*) está a la espera de recibir mensajes por el canal Unicast.
- Recibe un PFC_Mensaje del TIPO_PETICION_ALTA.
- Crea un PFC_Mensaje del TIPO_RESPUESTA_ALTA donde incluye el contenido del documento y los datos como id, nombre, fecha.
- Responde utilizando el mismo socket el PFC_Mensaje creado.

6 ESTUDIO DEL RENDIMIENTO

6.1 INTRODUCCIÓN

Uno de los aspectos importantes una vez realizado un desarrollo, es poder medir el rendimiento ante una serie de circunstancias donde poder sacar conclusiones del funcionamiento del mismo y poder detectar puntos débiles donde hacer hincapié para lograr tener un sistema robusto y completo.

En el siguiente estudio se han tenido en cuenta una serie de pruebas donde los parámetros con los que se jugaba era el número de nodos activos, el número de documentos a tratar pero sobre todo, un aspecto muy importante, qué pasa cuando un nodo está inactivo y tiene que actualizar toda su información.

En base a estos parámetros, se ha medido el vector tiempo; ya que como veremos, la aplicación tiene un consumo de CPU y memoria mínima.

A continuación se detallarán una serie de escenarios y los resultados obtenidos. Se podrá observar que en los primeros escenarios se ha medido el parámetro CPU y tiempo de respuesta. En los siguientes sólo tiempo de respuesta ya que la tónica de consumo de CPU siempre es la misma. El vector memoria no se ha medido ya que el consumo era mínimo (cuando se habla de memoria se entiende memoria de proceso y no de disco duro. Como se ha explicado en la parte técnica, si que se requiere de espacio de disco teniendo en cuanta que es una aplicación que trata con ficheros físicos).

Por último, y en base a las pruebas realizadas, se sacarán unas conclusiones de los resultados.

6.2 ESCENARIOS DE RENDIMIENTO

6.2.1 CONEXIÓN AL GRUPO – INICIAL

Medición:

Se realiza la conexión al grupo de los distintos nodos por vez primera. Esto significa que no tienen configuración local del grupo creada y tendrán que adherirse al mismo.

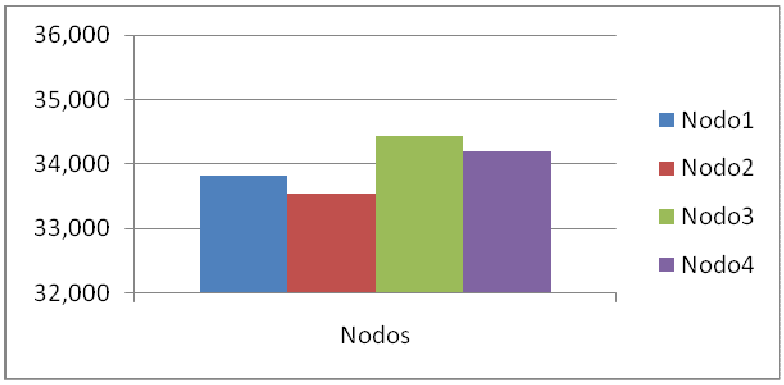
La conexión de los nodos será secuencial. Se inicia con el Nodo1 estando el resto de nodos desconectados. Una vez conectado Nodo1, se conecta Nodo2 estando Nodo3 y Nodo4 desconectados y así

sucesivamente hasta estar los 4 nodos conectado al grupo y al *superpeer*.

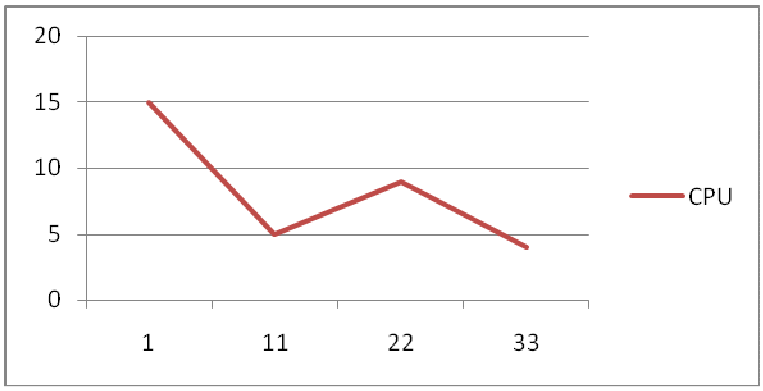
No hay ningún documento dado de alta ni por el nodo a conectarse, ni por los nodos conectados; por lo que no habrá actualizaciones a realizar en la conexión.

Se mide por un lado el tiempo que tarda cada nodo en realizar todo el proceso de conexión al grupo (hasta conseguir el estado *Conectado*). Y por otro lado, el consumo de CPU en todo este proceso.

tiempo/seg	**1 Nodo**	**2 Nodos**	**3 Nodos**	**4 Nodos**
Conectar	33,809	33,544	34,440	34,199



Se puede observar que el tiempo de conexión de cada uno de los nodos es muy similar y no hay afectaciones en que haya o no haya nodos conectados al grupo de forma previa.



A nivel de CPU se han marcado 4 tiempos. El tiempo inicial, 1, es dónde se inicia todo el proceso de conexión y donde la aplicación consumo el máximo de CPU. Se trata de un pico de incremento de un 10% de CPU con una duración de unos pocos milisegundos.

Luego cae la CPU ya que la aplicación queda a la espera de tener respuesta del *superpeer*. El siguiente punto de incremento, 22, es el punto donde ya se ha tenido respuesta con el *superpeer* y se inicia el

proceso de adhesión al grupo. Con estos pasos realizados la aplicación queda a la espera y el consumo de CPU baja al mínimo.

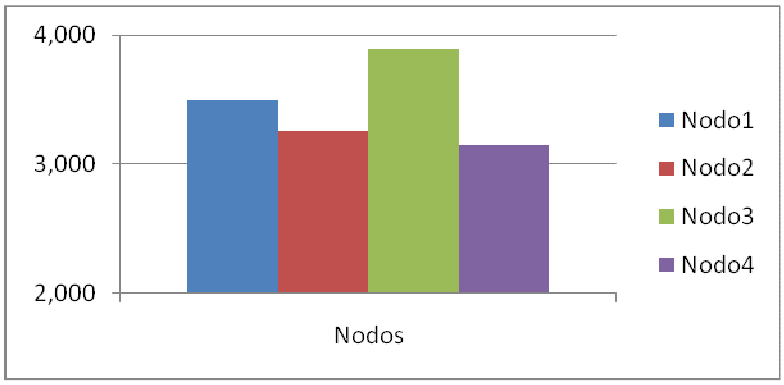
6.2.2 CONEXIÓN AL GRUPO – NO INICIAL

Medición:

A diferencia del escenario anterior, aquí previamente ya se ha hecho una conexión al grupo; por lo que la configuración JXTA ya está creada en local.

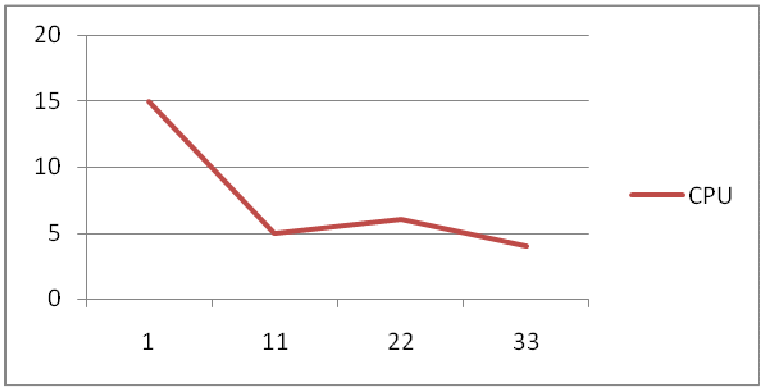
Por lo demás es lo mismo. Conexión secuencial de los Nodos.

tiempo/seg	**1 Nodo**	**2 Nodos**	**3 Nodos**	**4 Nodos**
Conectar	3,495	3,260	3,890	3,151



Se observa que los tiempos del proceso de conexión caen empicados y son muy parecidos entre nodos observando que no afecta el hecho de que haya o no nodos conectados al grupo.

Este descenso de los tiempos es debido a que la configuración de las comunicaciones JXTA ya existen en local y el tiempo de adhesión al grupo desaparece.



La gráfica de CPU es muy similar al caso anterior observando que el segundo pico de CPU ahora es mínimo y está relacionado a que el proceso de adhesión al grupo ya no se tiene que realizar.

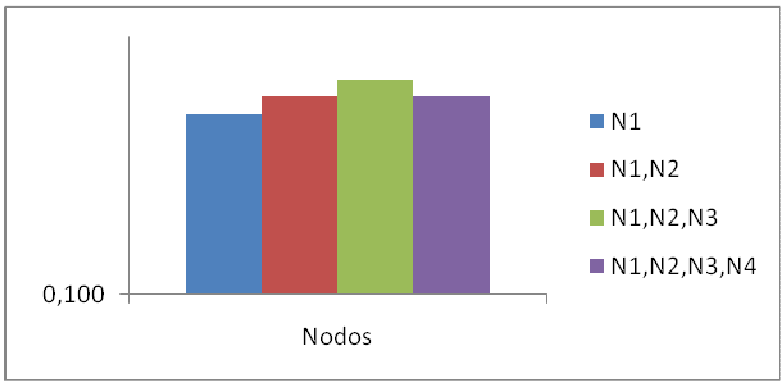
6.2.3 ALTA DE DOCUMENTO

Medición:

Se mide el proceso del alta de un documento en función de los nodos conectados. Se iniciará una primera alta de documento estando sólo conectado el nodo origen. Una segunda medición será otra alta estando el nodo origen y un segundo nodo conectado. Se realizarán posteriores mediciones siguiendo este patrón hasta tener los 4 nodos conectados.

El documento utilizado tiene un peso aproximado de 5k

tiempo/seg	**1 Nodo**	**2 Nodos**	**3 Nodos**	**4 Nodos**
1 doc (5k)	0,310	0,330	0,350	0,330



En la gráfica se puede ver la evolución de Nodos conectados con el tiempo de proceso del Nodo1 en dar de alta un documento.

Se observa que el tiempo de proceso de la alta de un documento no está vinculado al número de nodos conectados. Esto es debido a que la comunicación se realiza utilizando Multicast.

6.2.4 ACTUALIZARSE TRAS INACTIVO – ESCENARIO 1

Medición:

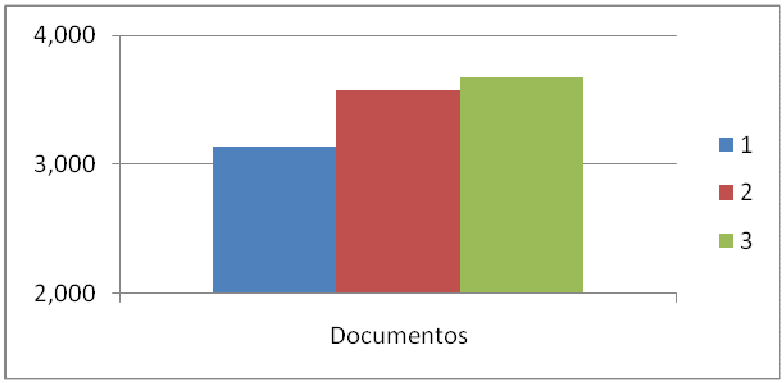
Se trata de tener un nodo inactivo, en este caso del Nodo 2, durante un periodo de tiempo donde lo importante no es el tiempo sino la actividad de los otros nodos.

Tras esa actividad, activar el nodo y medir el tiempo que tarda en realizar todo el proceso de conexión (tener en cuenta que en este proceso es donde actualiza el estado de los documentos).

En estas pruebas, todos los nodos estarán activos cuando el Nodo 2 se active. De esta forma los nuevos documentos que haya podrán ser servidos por los nodos origen.

La única actividad que ha habido es el alta de documento (se mide esta actividad porque es la más costosa al tener que viajar todo el documento y haber dobles vías de comunicación: primera con *superpeer* y segunda con el nodo origen).

tiempo/seg	**Nodo 1**	**Nodo 2**	**Nodo 3**	**Nodo 4**
conectar	1doc A, activo	3,136	0doc	0doc
conectar	2doc A, activo	3,572	0doc	0doc
conectar	4doc A, activo	3,671	0doc	0doc



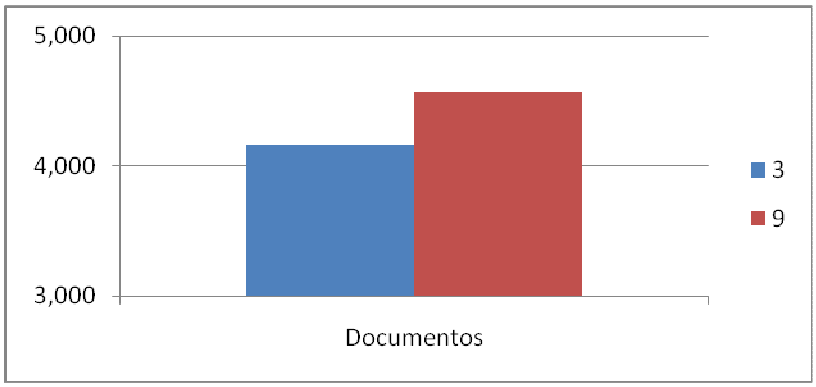
En esta primera medición se tiene el Nodo 2 inactivo y la única actividad es realizada por el Nodo 1. Se observa que en la primera prueba únicamente se ha dado de alta un documento (por el Nodo1) y el resto de nodos no han tenido actividad.

La segunda prueba es lo mismo pero el Nodo 1 ha dado de alta 2 documentos. Y en la última, ha dado de alta 4 documentos.

Se observa que aunque el tiempo se incrementa algo en función de los documentos dados de alta, el incremento es mínimo.

En segundo lugar se ha realizado la misma prueba pero ahora los tres nodos activos han tenido actividad.

tiempo/seg	**Nodo 1**	**Nodo 2**	**Nodo 3**	**Nodo 4**
conectar	1doc A, activo	4,165	1doc A, activo	1doc A, activo
conectar	3doc A, activo	4,567	3doc A, activo	3doc A, activo



Los tiempos de proceso se incrementan ya que el nodo al activarse tiene que comunicarse de forma individual con cada uno de los nodos para pedir el documento(s) dado(s) de alta.

Recalcar que todos los nodos están activos en el momento de levantar el Nodo 2(el que estaba inactivo).

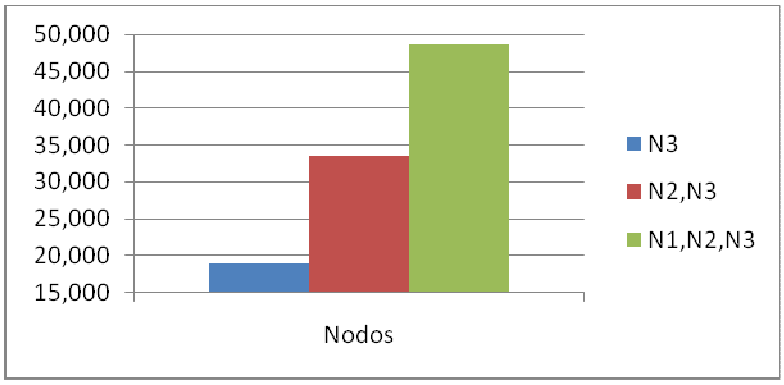
6.2.5 ACTUALIZARSE TRAS INACTIVO – ESCENARIO 2

Medición:

Es muy similar al anterior. Se tiene un nodo inactivo, en este caso el Nodo 4, y mientras se realiza actividad con los otros nodos. La actividad que se realiza es la alta de documentos pero en este caso, cuando se active el Nodo 4 (inactivo mientras había actividad), habrá en cada prueba un nodo origen inactivo. De tal forma, que llegará al escenario que no habrá ningún nodo origen activo.

La actividad es la alta de un documento por cada nodo activo (Nodo1, Nodo2 y Nodo3).

tiempo/seg	**Nodo 1**	**Nodo 2**	**Nodo 3**	**Nodo 4**
conectar	1doc A, activo	1doc A, activo	1doc A, inactivo	18,845
conectar	1doc A, activo	1doc A, inactivo	1doc A, inactivo	33,509
conectar	1doc A, inactivo	1doc A, inactivo	1doc A, inactivo	48,719



Se observa en este caso que el incremento del tiempo se casi duplica en función de los nodos inactivos existentes.

Este es uno de los puntos críticos y donde se puede crear un cuello de botella. Hay que tener en cuenta que al pedir las actualizaciones al *superpeer* le indica que hay tres documentos nuevos. Cada documento es de un nodo distinto. Cuando inicia al proceso de conexión con un nodo origen para obtener el documento, sino consigue conectarse por estar inactivo, tiene que realizar otra conexión con el *data center* para pedirle el documento y de esta forma mantener la consistencia del sistema.

Por ello el tiempo se va duplicando en función de los nodos origen inactivos que haya en el momento de actualizar la información.

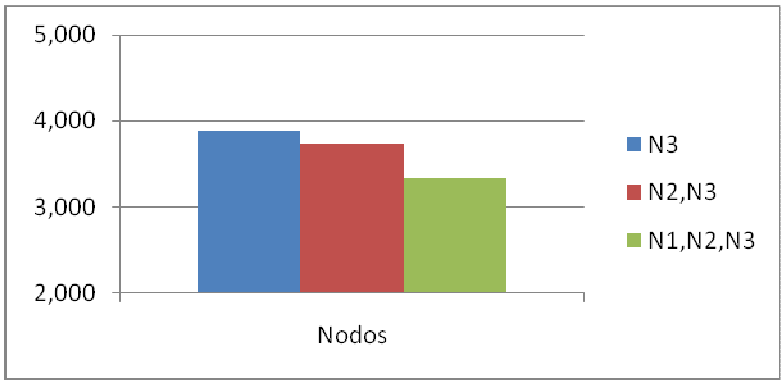
6.2.6 ACTUALIZARSE TRAS INACTIVO – ESCENARIO 3

Medición:

Estamos ante un escenario similar al anterior pero ahora la actividad es la baja de documentos. El Nodo 4 está inactivo mientras cada uno de los otros nodos da de baja un documento.

Las pruebas se realizarán teniendo en cada momento un nodo origen inactivo hasta estar los tres inactivos.

tiempo/seg	**Nodo 1**	**Nodo 2**	**Nodo 3**	**Nodo 4**
conectar	1doc B, activo	1doc B, activo	1doc B, inactivo	3,880
conectar	1doc B, activo	1doc B, inactivo	1doc B, inactivo	3,730
conectar	1doc B, inactivo	1doc B, inactivo	1doc B, inactivo	3,329



Sorprendentemente en este caso no hay afectación alguna. Los tiempos no se disparan ni son el doble por nodo/documento inactivo.

Esto es debido a que el proceso de baja es informado por el *superpeer* y no requiere de comunicación directa con los nodos origen. Por ello, no hay afectación alguna que haya o no nodos activos en el momento de la actualización (conexión).

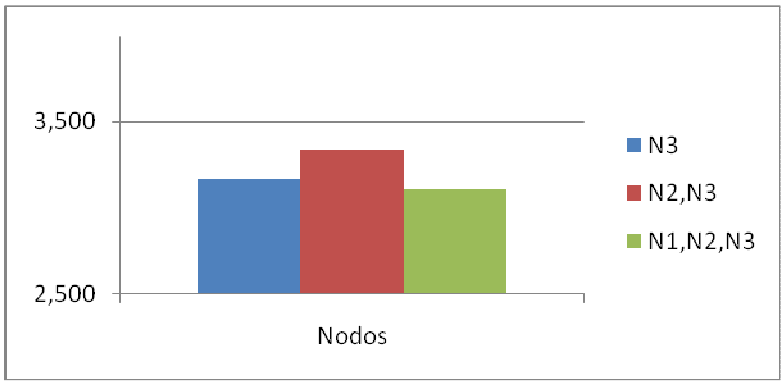
6.2.7 ACTIVARSE TRAS INACTIVO – ESCENARIO 4

Medición:

Estamos ante un escenario similar al anterior pero ahora la actividad es la modificación de documentos. El Nodo 4 está inactivo mientras cada uno de los otros nodos modifica un documento.

Las pruebas se realizarán teniendo en cada momento un nodo origen inactivo hasta estar los tres inactivos.

tiempo/seg	**Nodo 1**	**Nodo 2**	**Nodo 3**	**Nodo 4**
conectar	1doc M, activo	1doc M, activo	1doc M, inactivo	3,167
conectar	1doc M, activo	1doc M, inactivo	1doc M, inactivo	3,338
conectar	1doc M, inactivo	1doc M, inactivo	1doc M, inactivo	3,104



Al igual que en el caso anterior no hay afectación alguna. Los tiempos no se disparan ni son el doble por nodo/documento inactivo.

Esto es debido a que el proceso de modificación es informado por el *superpeer* y no requiere de comunicación directa con los nodos origen. El nodo superpeer informa de los cambios a realizar en el documento y por ello, no hay afectación alguna que haya o no nodos activos en el momento de la actualización (conexión).

6.3 CONCLUSIONES

Tras realizar pruebas de rendimiento del sistema creado, se constatan muchos de los aspectos relevantes de la comunicación tipo **peer to peer**.

En primer lugar se observa que hay aspectos como consumos de CPU y memoria de proceso que no son importantes para el sistema. Es cierto que al inicio de algunos procesos se pueden obtener picos del 10% de CPU pero que no pasan a ser preocupantes por el poco

tiempo de su duración (milisegundos) y porque son en puntos concretos de la solución.

De igual forma el consumo de memoria de proceso es mínimo y el único aspecto a tener en cuenta es el espacio en disco que dependerá del número y peso de ficheros xml que se manejen.

A nivel de procesos de negocio, destacar la diferencia en tiempo cuando un nodo se conecta al grupo por primera vez de cuando ya lo ha realizado. Esto es debido a la creación de la configuración local *jxta* donde se encarga de definir el grupo al que pertenece y todos los servicios a los que se suscriben.

Sin embargo, en el proceso de negocio no influye para nada el número de nodos que puedan o estén conectados. Se observa que el tiempo de proceso no se ve alterado por este parámetro y esto es debido a las bases de la comunicación **peer to peer**.

De igual forma, cuando la aplicación trabaja con los documentos xml, bien sea por altas, bajas o modificaciones, no hay afectación en el tiempo de proceso en función de los nodos que haya en ese instantes conectados al grupo. Esto es debido a que la comunicación en estos tipos de proceso es *Multicast*.

Podemos apreciar una bajada del rendimiento considerable (el tiempo de proceso se incrementa), en el proceso de conexión de un nodo que ha pasado tiempo inactivo y que tiene que actualizar los documentos conforme los cambios realizados por otros nodos. Concretamente cuando se han realizado alta de documentos y el nodo origen (el que ha creado el documento) no está activo en ese instante.

En este caso, tiene que entrar en juego el *data center*; de tal forma que cuando el nodo que se está actualizando pide al nodo origen el nuevo documento, pasa un tiempo hasta que detecta que el nodo origen no está activo por lo que tiene que realizar una nueva petición al *data center* para que sea éste el que le suministre el documento.

Si lo explicado lo hacemos extensible a varios documentos dados de alta mientras inactivo y a diferentes nodos que en ese momento no están levantados, el tiempo de proceso se eleva de forma considerable.

Hay que tener en cuenta que esta problemática sólo se da en el caso de el alta de nuevos documentos; en la modificación y baja no importa que el nodo origen no esté activo ya que es el propio *superpeer* el encargado de informar de forma directa este tipo de cambios.

Con todo esto vemos que el sistema funciona de forma óptima, que como era de esperar no importa el número de nodos que haya conectados en el grupo y que el vector de mejora es tras tiempos de inactividad en el proceso de altas de documentos.

7 CONCLUSIONES

Uno de los aspectos más importantes y relevantes de cara al presente y futuro es el trabajo en grupo sin importar el lugar físico desde donde trabajes. Hoy en día, y quizá debido a la crisis que estamos sufriendo donde los costes se intentan reducir de cualquier forma, el trabajar en casa está siendo una realidad y esto, como debe ser, no tiene que influir en el trabajo colaborativo con otros miembros del grupo.

El poder profundizar con el tipo de comunicación **peer to peer** me ha hecho ver las ventajas que aporta pero sobre todo, me ha abierto la mente y me ha permitido ver más allá del extendido software de compartición de películas, música, etc.

Peer to peer en mucho más que todo eso, es una forma de comunicación que rompe la idea de único servidor con múltiples clientes (con la problemática que esto aporta) para extender el concepto de que todos podemos ser clientes y servidores.

A la hora de crear entornos colaborativos facilita el poder disponer de un dispositivo capaz de suministrar en tiempo real la actualización de todos los documentos gestionados por el grupo y dar un rendimiento muy elevado sin tener en cuenta el número de nodos (clientes) conectados. Con lo que el grupo se puede hacer tan grande como se quiera sin notar ni un ápice la caída del rendimiento.

No hay que olvidar que el poder disponer de las librerías *jxta* te simplifica mucho el hecho de poder crear tu entorno colaborativo. Estas librerías te dan una interficie con la que poder implementar y crear tu propia aplicación de forma ágil sin tener que *pelearte* con aspectos de comunicación de bajo nivel.

Con todo esto, se ha podido crear un entorno colaborativo donde cada miembro del grupo puede publicar documentos xml y puede disponer de los xmls creados por los otros miembros del grupo. A parte, te permite disponer en todo momento de los cambios realizados en los documentos con la flexibilidad y ventajas que esto aporta a la hora de trabajar en grupo.

No por eso, aunque el entorno de trabajo creado es un entorno que se reduce a la gestión de documentos xml, con un número reducido de nodos y un solo nodo del tipo *superpeer*, se han establecido las bases para poder crear un entorno de trabajo donde el número de usuarios sea mucho más elevado, donde los documentos a compartir puedan ser cualquier tipo de documento que forme parte del proyecto en el que estemos trabajando y pudiendo elevar los nodos del tipo *superpeer* para ofrecer una mayor fiabilidad y elevar los vectores de rendimiento.

También destacar que aspectos de movilidad, tan importantes en la problemática actual, están cubiertos con este tipo de comunicación. La posibilidad de poder tener en un dispositivo móvil un nodo nos facilita el poder trabar desde cualquier lugar.

Por último me gustaría destacar que este tipo de proyecto me ha permitido dar un paso para conocer de forma más profunda una nueva forma de comunicación (que aunque realmente no sea nueva, para mí a nivel técnico sí que lo es) que me permitirá cambiar el planteamiento de alguno de los proyectos que uno puede llevar a cabo pero que sobre todo me permite tener un visión mucho más clara de las posibilidades que este tipo de comunicación te da.

8 TRABAJO FUTURO

Aunque muchos de los aspectos que se comentan ya se han ido viendo a lo largo de la memoria, se destacan aspectos de trabajo de futuro teniendo como base el proyecto desarrollado:

- Poder trabajar con más de un nodo tipo *superpeer*. Todas las pruebas y desarrollo realizado sólo se ha tenido en cuenta trabajar con un nodo tipo *superpeer*.
- Trabajar con cualquier tipo de documento. En la actualidad sólo se manejan ficheros tipo xml. Se podría hacer extensible a cualquier tipo de documento y habilitar editores de aquellos que lo pudiesen permitir.
- Mejorar actualización de los ficheros. Se ha visto que unos de los puntos donde el rendimiento del sistema decrecía, era tras un periodo de inactividad de un nodo, al tratar de obtener las altas de los documentos y no estar activos los nodos origen. Tratar de mejorar este aspecto para aumentar el tiempo de respuesta.
- Dispositivos móviles. Todo lo probado ha sido en ordenadores. Ampliar estas pruebas a dispositivos móviles.

9 ANEXOS

9.1 ESTUDIO DE LA BASE DE DATOS

9.1.1 INTRODUCCIÓN

A la hora de llevar a cabo la implementación de todo el sistema, uno de los elementos principales es el Sistema Gestor de Base de Datos (SGBD) a utilizar para dar cabida a la consistencia de información que la solución tiene que manejar.

Como el mercado ofrece diversas alternativas, se estudiarán tres sistemas con el fin de seleccionar aquel que más se ajuste a las necesidades del producto final teniendo en cuenta los requisitos establecidos inicialmente.

En primer lugar se hará un estudio de las características de cada una de ellas; los tres SGBD seleccionados son:

- PostGreSQL
- JavaDB
- Oracle

Con esta información, se seleccionar la que más se ajuste a nuestras necesidades.

9.1.2 SGBD POSTGRESQL

Por su historia, producto de más de 30 años, se puede decir que es un sistema estable y con una gran comunidad de soporte.

A continuación se detallan algunas de sus principales características:

- Sistema de gestión de base de datos relacional orientada a objetos.
- Es libre, se publica bajo Licencia BSD; licencia de software libre como OpenSSL pero más permisiva, permite el uso del código fuente en software no libre.
- Se define como Full ACID (Automicidad, Consistencia, Aislamiento, Durabilidad)
- Funciona en casi todos los principales sistemas operativos: Linux, Unix, Mac OC, Windowsm..

- Amplia documentación bien organizada, pública y libre.
- Comunidades muy activas
- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Java/JDBC,Perl, Python, etc.
- Drivers: Odbc, Jdbc, .NET, etc
- Disponibilidad de: Triggers, store-procedures, funciones, secuencias, relaciones, reglas, tipos de datos definidos por usuario, vistas, vistas materializadas, etc.
- Soporte de tipos de datos SQL92 y SQL99
- Soporte de protocolo de comunicación encriptado por SSL
- Extensión para alta disponibilidad
- Utilidades para: Limpieza de base de datos (Vacuum), análisis y optimización de querys
- Almacenaje para tipo de datos grandes (TOAST)
- Varios tipos de índices
- Clusterización de datos en base a índices

Si se habla de los límites de este gestor de base de datos se puede observar que:

- Máximo de base de datos: Ilimitado
- Máximo de tamaño de tabla: 32TB
- Máximo de tamaño de registros: 1,6TB
- Máximo de tamaño de campo: 1GB
- Máximo de registros por tabla: Ilimitado
- Máximo de campos por tabla: 250 a 1600
- Máximo de índices por tabla: Ilimitado
- Lenguajes en los que se puede programar funciones: pl/pgsql, pl/java, pl/perl, pl/python, tcl, pl/php, C, C++, Ruby, etc..
- Métodos de almacenamiento de índices: 4 (B-tree, R-tree, Hash, Gist)

Ventajas

Se pueden destacar las siguientes ventajas

- Está considerado como la base de datos de código abierto más avanzada del mundo; proporciona un gran número de características que normalmente sólo se encuentran en base de datos comerciales como Oracle.
- Es libre (gratuita)
- Amplio soporte con comunidades muy activas varias de ellas en castellano
- Amplia red de documentación libre
- Muy extendida.
- Amplia gama de utilidades de tunning y de mantenimiento del propio gestor de base de datos
- Es altamente confiable a cuanto estabilidad se refiere
- Posee manejo y control de transacciones para asegurar la consistencia de los datos
- Soporte SQL92 y SQL99
- Pude operar sobre varias plataformas
- Soporte replicación de base de datos asíncrona
- Buen sistema de seguridad mediante gestión de usuarios, grupos de usuarios, permisos y contraseñas.
- Gran capacidad de almacenamiento

Desventajas

Se pueden destacar las siguientes desventajas

- Consume muchos recursos; necesidad de mayores características de hardware
- En función de su uso, su arquitectura cliente/servidor
- Algo más lenta que otro sistemas gestores
- Primer contacto complejo; sintaxis de algunos comandos poco intuitivo

9.1.3 SGBD JAVADB

JavaDB o también conocido como Apache Derby en un sistema gestor de base de datos relacional escrito en Java.

A continuación se detallan algunas de sus principales características:

- Tamaño en disco de 2MB
- Proyecto opensource licenciado bajo la Apache 2.0 License (en la actualidad se distribuye como Sun Java DB)
- Soporta cifrado, roles y permisos.
- Posee SQL SCHEMAS
- Soporta internamente procedures
- Soporte multilenguaje
- Soporte SQL92 y SQL99
- Full ACID
- Contiene copias de respaldo asíncronas
- Posee herramientas de rendimiento
- Puede funcionar en modo empotrado; o como cliente /servidor
- Soportado por los principales sistemas operativos

Ventajas

Se pueden destacar las siguientes ventajas

- Libre (gratuito)
- Soporte SQL92 y SQL99
- Puede trabajar en modo empotrado en la propia aplicación.

Desventajas

Se pueden destacar las siguientes desventajas

- No tiene *bindings* para otros lenguajes (que no sea Java)
- Hace un uso de mayor cantidad de memoria
- Rendimiento y compatibilidad SQL puede considerarse inferior que otras alternativas
- En el modo empotrado, sólo soporte un único proceso que tenga abierta la base de datos.

9.1.4 SGBD ORACLE

Es un sistema gestor de base de datos relacional y en la actualidad se podría decir que es uno de los mayores y más usados Sistema manejadores de base de datos relacionales del mundo.

A continuación se detallan algunas de sus principales características:

- Distribuido comercialmente bajo licencia propietaria por la empresa Oracle Corporation.
- Puede ejecutarse en PCs, microcomputadores, mainframes..
- Dispone de versiones para lo principales sistemas operativos
- Soporta 17 idiomas
- Disponibilidad sinfín de productos relacionados con la BD
- Soporta todo tipo de funciones: Triggers, procedimientos almacenados, lenguaje de programación PL/SQL, etc..
- Permite el uso de particiones para mejorar el eficiencia, de replicación
- Permite la administración de base de datos distribuidas
- Arquitectura cliente/servidor
- Gestión de grandes bases de datos
- Usuarios concurrentes
- Alto rendimiento en transacciones
- Sistemas de alta disponibilidad
- SQL-92, SQL-99
- Buena gestión de la seguridad
- Autogestión de la integridad de datos
- Full ACID
- Portabilidad, Compatibilidad y conectividad.
- Amplia gama de tipología de datos

<u>Ventajas</u>

Se pueden destacar las siguientes ventajas

- Una de los SGBD más extendidos y utilizados del mundo
- SGBD de alto nivel; robusto y con la capacidad de gestión de elevados volúmenes de datos

- Amplia documentación y soporte por su extensión de utilización en el mercado
- Disponible versión reducida gratuita para pequeños desarrollos y formación
- Disponible para las principales plataformas
- Disponible es 17 lenguajes
- Permite trabajar en modo distribuido
- Gran gestión de la seguridad
- Etc..

Se podría decir que Oracle es uno de los mejores SGBD (sino el mejor) del mercado.

Desventajas

Se pueden destacar las siguientes desventajas

- Producto distribuido bajo licencia propietaria (no gratuito). Aunque se dispone de una versión express gratuita
- Muy pesado
- Arquitectura cliente/servidor. No se puede trabajar en modo empotrado

9.1.5 CONCLUSIONES

Una vez estudiado los tres sistemas de gestión de base de datos, se podrían sacar varias conclusiones. En primer lugar se muestra una simple comparativa de los tres para tener presente algunas de las características:

SGBD	Licencia	MultiPlataf.	ACID	Int.Referencial
PostGreSQL	Libre	Si	Si	Si
JavaDB	Libre	Si	Si	Si
Oracle	Pago	Si	Si	Si

SGBD	Transaccional	SQL92-99	Grandes vol datos	Empotrado Java

PostGreSQL	Si	Si	Si	No
JavDB	Si	Si	No	Si
Oracle	Si	Si	Si	No

Visto el estudio y refrescando algunas de las principales características, se observa que *Oracle* sería la mejor alternativa. Se trata de una de los SGBD más extendidos en todo el mundo y los servicios y particularidades hacen de ella la base de datos número uno.

Pero está claro que para poder seleccionar un SGBD a utilizar en un nuevo proyecto, nos tenemos que situar en el marco, características y objetivos de la propia solución a desarrollar.

En el caso que nos ocupa, podemos definir que nuestros requisitos del SGBD no son de un gran volumen de información y no se tiene que disponer de un sistema muy grande ni complejo.

Nuestra solución necesita disponer de un SGBD que sea transparente para el usuario final; que no necesite de una administración ni instalación previa y que su coste sea reducido o nulo.

Una de las principales premisas fijadas en la ejecución del proyecto es desarrollar *un sistema de bajo coste*.

Por otro lado, y siempre pensado en un ámbito de trabajo a distancia, se piensa en un sistema fácil de instalar y que no requiera de una administración compleja ni de instalación de elementos adicionales que no sean la propia aplicación.

Por último, resaltar que otra de nuestras principales características es desarrollar un producto J2EE que corra sobre el sistema operativo Windows. Aunque como cualquier visionario y por la tipología del aplicativo a desarrollar, no cabe olvidar que la solución final pueda ser ejecutada en cualquier plataforma y sin requisitos previos de instalación (o como poco, los mínimos posibles).

Por todo esto, se utilizará **JavaDB** como el SGBD a utilizar.

Se descarta Oracle porque *no hace falta un cañon para matar moscas* y porque la idea es trabajar en un sistema NO cliente/servidor. Tanto *Oracle* como *PostGreSQL* trabajan bajo esta arquitectura y esto obligaría a instalar un servidor de base de datos en cada uno de los puestos de trabajo (máquinas) donde se quiere ejecutar la solución.

Sin embargo, **JavaDB** nos ofrece todo lo que necesitamos pudiendo empaquetar todo la solución evitando instalaciones de productos adicionales.

9.2 INTERFICIES JAVA

9.2.1 PANEL PRINCIPAL

```
package pfc.prototipo;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;

import javax.swing.*;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import pfc.prototipo.bbdd.PFC_BDGestion;
import pfc.prototipo.jxta.peer.PFC_Peer;


/**
 * <p>Panel principal punto de partida de un Nodo tipo Peer.</p>
 * <p>Genera la pantalla inicial donde se define marco de trabajo.</p>
 *
 * @author Angel
 *
 */
public class PFC_PanelPrincipal extends JFrame
{
	//Trazas
	private final static Logger moLog = Logger.getLogger(PFC_PanelPrincipal.class);


	//Gestion de la base de datos
	private PFC_BDGestion moGestionBD = null;

	//Ventana principal de la aplicacion
	//JFrame moMarco = null;

	//Definición de paneles
	private PFC_PanelConectarConexion	moPanelGestion;
	private PFC_PanelListarFicheros	moPanelListar;
	private PFC_PanelListarFicheros	moPanelNuevo;
	private PFC_PanelConfigurar	moPanelConfigurar;
	private JPanel loPanelEstado;
	//Indica que opcion de menu tenemos seleccionado.
	private String msOpcionMenu = "";

   // Gestion de la conexion al grupo de Peer
   private PFC_Peer moPeer = null;


	/**
	 * <p>Inicializa la aplicacion creando/cargando configuracion de la misma.</p>
	 */
	public PFC_PanelPrincipal()
	{
		super(PFC_Constantes.NOMBRE_APLICACION + " : " +
PFC_Constantes.NODO_NOMBRE);

		//Indica si existe configuracion
		boolean lbExisteConfig = false;

		try
```

```
		{
			//Se crea conexion con base de datos.
			moGestionBD = new PFC_BDGestion();

			//Se verifica si existe una configuracion creada. En caso de no existir, se
genera de nuevo.
			//Si no existe es porque la aplicacion se utiliza por primera vez y es cuando se
tiene que generar los identificadores unicos
			//del peer y de los canales de comunicacion necesarios
			lbExisteConfig =
moGestionBD.existeTabla(PFC_Constantes.TABLA_CONFIGURACION);

			if (lbExisteConfig)
			{
				moLog.info("Se carga configuracion.");
				PFC_Constantes.cargarConfiguracion(moGestionBD);
			}
			else
			{
				moLog.info("Aplicacion en proceso de inicializacion. Se carga
configuracion por primera vez");
				//La primera vez que se inicia, se establece que el directorio es el
path actual mas el establecido por defecto
				PFC_Constantes.DIRECTORIO = new File(".").getCanonicalPath() +
PFC_Constantes.DIRECTORIO;
				moLog.info("Se establece el directorio de trabajo
inicial<"+PFC_Constantes.DIRECTORIO+">");
				PFC_Constantes.crearConfiguracion(moGestionBD, false);
			}

		}catch(Exception e)
		{
			moLog.error("No se ha podido iniciar la solucion:"+e.getLocalizedMessage());
			System.out.println("No se ha podido iniciar la solucion");
			if (moGestionBD != null)
			{
				moGestionBD.cerrarConexion();
			}
			System.exit(-1);
		}

	}//PFC_PanelPrincipal



	/**
	 * <p>Se inician los siguiente puntos:</p>
	 * <p>  . Sistema de trazas</p>
	 * <p>  . Marco principal de la aplicación</p>
	 * <p></p>
	 * <p>Únicamente hay un parámetro de entrada que indica el número de nodo que se está
generando. Es un numérico.</p>
	 * @param args
	 */
	public static void main(String[] args)
	{
		int liNumeroNodo = -1;

		//Lo primero que se realiza es inicialiar el sistema de trazas. Se carga fichero de
propiedades donde se indica toda la gestión del sistema de trazas
		try
		{
			//Se carga configuracion de trazas
			PropertyConfigurator.configure("log4j.properties");

		}catch(Exception e)
		{
			System.out.println("ERROR: Problemas al inicializar el sistema de trazas.
Verifique su instalación para poder utilizar la solución.");
```

```
                        System.exit(-1);
                }


                //Se verifica que se tiene el pametro de entrada que indica el numero de nodo a crear
                if ( (args == null) || (args.length == 0) )
                {
                        moLog.error("Parámetro que indica número de nodo no existe!. Introducir
número de nodo.");
                        System.out.println("Parámetro que indica número de nodo no existe!.
Introducir número de nodo.");
                        System.exit(-1);
                }

                //Se verifica que el parametro introducido sea un numerico
                try
                {
                        liNumeroNodo =  new Integer(args[0]).intValue();
                }catch(Exception e)
                {
                        //No se trata de un numero. Parametro introducido pero formato erroneo
                        moLog.error("Formato del parámetro erróneo. Debe ser un numérico
entero.");
                        System.out.println("Formato del parámetro erróneo. Debe ser un numérico
entero.");
                        System.exit(-1);
                }

                //Se construye el nombre del nodo. Se compone de una parte fija y comun a cualquier
nodo mas el número de nodo pasado por parámetro
                PFC_Constantes.NODO_NOMBRE = PFC_Constantes.NODO_NOMBRE_PREFIJO +
liNumeroNodo;

                moLog.debug("Se inicia el proceso de creacion del Panel Principal para
Nodo<"+PFC_Constantes.NODO_NOMBRE+">");

                //Se inicializa panel principal
                final PFC_PanelPrincipal loVentana = new PFC_PanelPrincipal();

                java.awt.EventQueue.invokeLater(new Runnable()
                {
                        public void run()
                        {
                                loVentana.generarPantallaPrincipal();
                        }
                });
        }//main

        /**
         * <p>Genera el marco principal de la aplicacion.</p>
         */
        private void generarPantallaPrincipal()
        {
    //Se inicializan parámetros de la pantalla principal
                JFrame.setDefaultLookAndFeelDecorated(true);
                this.setName(PFC_Constantes.NOMBRE_APLICACION);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setSize(PFC_Constantes.PANTALLA_ANCHO, PFC_Constantes.PANTALLA_ALTO);

                //Generacion del menu
                generarMenu();

                // Incluir un JPanel abajo como barra de estado donde se indicara si esta o no
conectado al grupo
                loPanelEstado = new JPanel();
                loPanelEstado.setBorder(BorderFactory.createEtchedBorder());
                loPanelEstado.setName(PFC_Constantes.PANEL_ESTADO_NOMBRE);

                JLabel loTexto = new JLabel(PFC_Constantes.ESTOY_CONECTADO ?
PFC_Constantes.ESTADO_CONECTADO : PFC_Constantes.ESTADO_DESCONECTADO);
```

```
		loTexto.setName(PFC_Constantes.LABEL_ESTADO_NOMBRE);
		loPanelEstado.add(loTexto);

		JProgressBar loBarra = new JProgressBar(0, 100);
		loBarra.setName(PFC_Constantes.PROGRESO_BARRA);
		loBarra.setStringPainted(true);
		loBarra.setVisible(false);
		loPanelEstado.add(loBarra);

		//Se añade el menu a la pantalla principal
	this.getContentPane().add(loPanelEstado, BorderLayout.SOUTH);

		// Pantalla visible
	this.setVisible(true);
		// Pantalla centrada
	this.setLocationRelativeTo(null);

		//Se añade un listener para capturar el evento de cierre de pantalla y poder cerrar la
conexion
	this.addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e)
			{
				moLog.info("Se cierra la aplicacion!");
				if (moGestionBD != null)
				{
					moGestionBD.cerrarConexion();
				}

				// En caso de estar conectado a la red, para comunicaciones
				pararComunicaciones();

				System.exit(0);
			}
		});

	}//generarPantallaPrincipal

	/**
	 * <p>Crea el menu de la aplicacion.</p>
	 */
	private void generarMenu()
	{
		//Menu
		JMenuBar loMenubar = null;
		//Opcion de menu Conexion
		JMenu loOpConexion = null;
		//Opcion de menu Fichero
		JMenu loOpFichero = null;
		//Opcion de menu Sistema
		JMenu loOpSistema = null;

		//Opcion Conectar del menu Conexion
		JMenuItem loOpGestionar;
		//Opcion Listar del menu Fichero
		JMenuItem loOpListar;
		//Opcion Nuevo del menu Fichero
		JMenuItem loOpNuevo;
		//Opcion Configuracion
		JMenuItem loOpConfig;
		//Opcion Salir
		JMenuItem loOpSalir;

		//Se crea opcion Listar del menu Fichero
		loOpListar = new JMenuItem(PFC_Constantes.MENU_FICHEROS_LISTAR);
		loOpListar.setMnemonic('L');
		loOpListar.setAccelerator(KeyStroke.getKeyStroke("control L"));
		//Se crea opcion Nuevo del menu Fichero
		loOpNuevo = new JMenuItem(PFC_Constantes.MENU_FICHEROS_NUEVO);
		loOpNuevo.setMnemonic('N');
		loOpNuevo.setAccelerator(KeyStroke.getKeyStroke("control N"));
```

```
                //Se crea opcion Conectar del menu Conexion
                loOpGestionar = new JMenuItem(PFC_Constantes.MENU_CONEXION_GESTION);
                loOpGestionar.setMnemonic('M');
                loOpGestionar.setAccelerator(KeyStroke.getKeyStroke("control m"));
                //Se crea opcion Configuracion del menu Sistema
                loOpConfig = new JMenuItem(PFC_Constantes.MENU_SISTEMA_CONFIGURAR);
                loOpConfig.setMnemonic('C');
                loOpConfig.setAccelerator(KeyStroke.getKeyStroke("control O"));
                //Se crea opcion Salir del menu Sistema
                loOpSalir = new JMenuItem(PFC_Constantes.MENU_SISTEMA_SALIR);
                loOpSalir.setMnemonic('E');
                loOpSalir.setAccelerator(KeyStroke.getKeyStroke("control e"));

    //Se crea y genera menu de la aplicacion
    loMenubar = new JMenuBar();
    //Opcion Conexion
    loOpConexion = new JMenu(PFC_Constantes.MENU_CONEXION);
    loOpConexion.setMnemonic('C');
    loMenubar.add(loOpConexion);
    loOpConexion.add(loOpGestionar);

    //Opcion Ficheros
    loOpFichero = new JMenu(PFC_Constantes.MENU_FICHEROS);
    loOpFichero.setMnemonic('F');
            loMenubar.add(loOpFichero);
            loOpFichero.add(loOpListar);
            loOpFichero.add(loOpNuevo);

    //Opcion Sistema
            loOpSistema = new JMenu(PFC_Constantes.MENU_SISTEMA);
            loOpSistema.setMnemonic('S');
            loMenubar.add(loOpSistema);
            loOpSistema.add(loOpConfig);
            loOpSistema.add(loOpSalir);

            //Se añade los eventos de las opciones de menu
            loOpGestionar.addActionListener(new gestionarAction());
            loOpListar.addActionListener(new listarFicherosAction());
            loOpNuevo.addActionListener(new nuevoFicheroAction());
            loOpConfig.addActionListener(new configurarAction());
            loOpSalir.addActionListener(new salirAction());

            //Se añade el menu a la pantalla principal
            this.getContentPane().add(loMenubar, BorderLayout.NORTH);
        }//generarMenu

/**
 * <p>Carga panel de conexion</p>
 * @author angel
 *
 */
class gestionarAction implements ActionListener
{
    public void actionPerformed(ActionEvent e)
    {
                //Se crea panel Listar con los elementos necesarios
        limpiarPanel();
        moPanelGestion = null;
        moPanelGestion = new PFC_PanelConectarConexion(moGestionBD);

                PFC_Constantes.getPrincialMenu(e).getContentPane().add(moPanelGestion);
                moPanelGestion.updateUI();
                msOpcionMenu = "CONEXIONES";
    }
}//class conectarAction


/**
 * <p>Carga panel con Listado de documento y edicion de ficheros</p>
 * @author angel
```

```
 *
 */
class listarFicherosAction implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
                //Se crea panel Listar con los elementos necesarios
      limpiarPanel();
      moPanelListar = null;
      moPanelListar = new PFC_PanelListarFicheros(moGestionBD, false, moPeer);

                PFC_Constantes.getPrincialMenu(e).getContentPane().add(moPanelListar);
                moPanelListar.updateUI();
                msOpcionMenu = "LISTAR";
   }
}//cargarFicheroAction

/**
 * <p>Carga panel en modo nuevo documento.</p>
 * @author angel
 *
 */
class nuevoFicheroAction implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
                //Se crea panel Nuevo con los elementos necesarios
      limpiarPanel();
      moPanelNuevo = null;
      moPanelNuevo = new PFC_PanelListarFicheros(moGestionBD, true, moPeer);
                PFC_Constantes.getPrincialMenu(e).getContentPane().add(moPanelNuevo);
                moPanelNuevo.updateUI();
                msOpcionMenu = "NUEVO";
   }
}//cargarFicheroAction

/**
 * <p>Carga panel de configuracion de la aplicacion</p>
 * @author angel
 *
 */
class configurarAction implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
                //Se crea panel Nuevo con los elementos necesarios
      limpiarPanel();
      moPanelConfigurar = null;
      moPanelConfigurar = new PFC_PanelConfigurar(moGestionBD);
                PFC_Constantes.getPrincialMenu(e).getContentPane().add(moPanelConfigurar);
                moPanelConfigurar.updateUI();
                msOpcionMenu = "CONFIGURAR";
   }
}//configurarAction

/**
 * <p>Se sale de la aplicacion</p>
 * @author angel
 *
 */
class salirAction implements ActionListener
{
   public void actionPerformed(ActionEvent e)
   {
      System.exit(-1);
   }
}//salirAction

/**
 * <p>Se encarga de quitar el Panel activado de la ventana</p>
```

```
 */
private void limpiarPanel()
{
	if (msOpcionMenu.equalsIgnoreCase("NUEVO"))
	{
		moLog.debug("Limpiado de Panel - Se quita Panel NUEVO");
		this.remove(moPanelNuevo);
		this.repaint();
	}
	if (msOpcionMenu.equalsIgnoreCase("LISTAR"))
	{
		moLog.debug("Limpiado de Panel - Se quita Panel LISTAR");
		this.remove(moPanelListar);
		this.repaint();
	}
	if (msOpcionMenu.equalsIgnoreCase("CONEXIONES"))
	{
		moLog.debug("Limpiado de Panel - Se quita Panel CONEXIONES");
		this.remove(moPanelGestion);
	}
	if (msOpcionMenu.equalsIgnoreCase("CONFIGURAR"))
	{
		moLog.debug("Limpiado de Panel - Se quita Panel CONFIGURAR");
		this.remove(moPanelConfigurar);
	}

}//limpiarPanel()


public PFC_Peer getComunicaciones()
{
	return moPeer;
}
public void setComunicaciones(PFC_Peer poPeer)
{
	moPeer = poPeer;
}

public void pararComunicaciones()
{
	if (moPeer != null)
	{
		moPeer.pararServicio();
		moPeer = null;
	}
}

	/**
	 * <p>Cambia el literal del estado de la aplicacion. Conectado/Desconectado al Grupo</p>
	 * @param psEstado Literal con el estado
	 */
	public void cambiarNotificacionEstado(String psEstado)
	{
		System.out.println("HOLAA");
		((JLabel)loPanelEstado.getComponent(0)).setText(psEstado);
	}//cambiarNotificacionEstado

}
```

9.2.2 GESTIÓN DE LA BBDD

```
package pfc.prototipo.bbdd;
```

```
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Timestamp;
import java.util.Hashtable;
import java.util.Vector;

import org.apache.log4j.Logger;

import pfc.prototipo.PFC_Constantes;
import pfc.prototipo.PFC_Documentos;


/**
 * <p>Gestiona la conexion y el manejo con la base de datos</p>
 * @author Angel
 *
 */
public class PFC_BDGestion
{
        //Trazas
        private final static Logger moLog = Logger.getLogger(PFC_BDGestion.class);

        // Objeto conexion con la base de datos de la aplicacion
        private Connection moConexion = null;

        /**
         * <p>Crea una conexion con la base de datos de la aplicacion</p>
         * @throws Exception
         */
   public PFC_BDGestion() throws Exception
   {
      try
      {
         //Se instancia driver de la base de datos
         Class.forName(PFC_Constantes.BD_DRIVER);
        moLog.debug("Driver<"+PFC_Constantes.BD_DRIVER+"> de base de datos instanciado!");

      } catch(java.lang.ClassNotFoundException e)
      {
         moLog.error("Problemas al instanciar el Driver de Base de datos.
Driver<"+PFC_Constantes.BD_DRIVER+">");
         throw new Exception("Problemas al instanciar el Driver de Base de datos.
Driver<"+PFC_Constantes.BD_DRIVER+">:"+e.getMessage());
      }

      try
      {
        // Se crea conexion con la base de datos de la aplicacion
         moLog.debug("Se inicia proceso para crear conexion con la base de datos de la aplicación. URL
de la base de datos<"+PFC_Constantes.BD_URL+">");
         moConexion = DriverManager.getConnection(PFC_Constantes.BD_URL);
        moLog.debug("Conexion con la base de datos de la aplicación realizado de forma correcta!");
      }  catch (Throwable e)
      {
         moLog.error("Problemas al crear la conexion con base de datos. URL
BBDD<"+PFC_Constantes.BD_URL+">");
         throw new Exception("Problemas al crear la conexion con base de datos. URL
BBDD<"+PFC_Constantes.BD_URL+">:"+e.getMessage());
      }
   }//PFC_BDGestion

   /**
    * <p>Genera la configuracion inicial del sistema cuando se accede por primera vez</p>
    * @param psNomPeer Nombre del peer
```

```
     * @param psIdPeer Identificador del peer
     * @param psNomGrupo Nombre del grupo
     * @param psIdGrupo Identificador del grupo
     * @param psDirectorio Directorio donde se guardaran los documentos
     * @throws Exception
     */
    public void inicializaSistema(String psNomPeer, String psIdPeer, String psNomGrupo, String
psIdPipeMulticast, String psDirectorio,
                    String psIpSP, String psPuertoSP, String psIdPipeUnicast, boolean pbSoySuperPeer)
throws Exception
    {
            //Query para crear la tabla
            String lsTabla = null;
            Statement loCreate = null;
            PreparedStatement loInsert = null;
            String lsInsert = null;

            //Se crea la tabla de configuracion del sistema
            try
            {
                    loCreate = moConexion.createStatement();
                            //Sentencia para crear la tabla de Configuracion
                    lsTabla = "CREATE TABLE CONFIGURACION  "
            +  "(CONFIG_PARAMETRO VARCHAR(20) NOT NULL"
              +  "   CONSTRAINT PARAMETRO_PK PRIMARY KEY, "
              +  " CONFIG_VALOR VARCHAR(300) NOT NULL) " ;

           //Se crea la tabla
           loCreate.execute(lsTabla);

                            moLog.debug("Tabla CONFIGURACION creada!");

                    } catch (SQLException e)
                    {
                            moLog.error("Problemas al crear la tabla de Configuracion:"+e.getMessage());
                            try
                            {
                                    loCreate.close();
                                    loCreate = null;
                            } catch (SQLException e1)
                            {
                                    moLog.error("Problemas al cerrar el Statement:"+e1.getMessage());

                            }
                            throw new Exception("No se he podido generar la configuracion!");
                    }

                    try
                    {
                            //Una vez creada la tabla, se insertan los valores de configuracion
                            lsInsert = "INSERT INTO CONFIGURACION (CONFIG_PARAMETRO,
CONFIG_VALOR) VALUES (?, ?)";

                            loInsert = moConexion.prepareStatement(lsInsert);

                            //Se inserta nombre del peer
                            loInsert.setString(1, PFC_Constantes.CONFIGURACION_NOMBRE_PEER);
                            loInsert.setString(2, psNomPeer);

            moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_NOMBRE_PEER+"> creado
<"+psNomPeer+">");
                            loInsert.executeUpdate();
                            loInsert.clearParameters();

                            //Se inserta identificador del peer
                            loInsert.setString(1, PFC_Constantes.CONFIGURACION_ID_PEER);
                            loInsert.setString(2, psIdPeer);
                            moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_ID_PEER+">
creado <"+psIdPeer+">");
                            loInsert.executeUpdate();
```

```
                loInsert.clearParameters();

                //Se inserta nombre del grupo
                loInsert.setString(1, PFC_Constantes.CONFIGURACION_NOMBRE_GRUPO);
                loInsert.setString(2, psNomGrupo);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_NOMBRE_GRUPO+"> creado
<"+psNomGrupo+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta identificador del pipe Multicast
                loInsert.setString(1,
PFC_Constantes.CONFIGURACION_ID_PIPE_MULTICAST);
                loInsert.setString(2, psIdPipeMulticast);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_ID_PIPE_MULTICAST+">
creado <"+psIdPipeMulticast+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta identificador del pipe Unicast
                loInsert.setString(1, PFC_Constantes.CONFIGURACION_ID_PIPE_UNICAST);
                loInsert.setString(2, psIdPipeUnicast);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_ID_PIPE_UNICAST+"> creado
<"+psIdPipeUnicast+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta la IP del SuperPeer
                loInsert.setString(1, PFC_Constantes.CONFIGURACION_SUPERPEER_IP);
                loInsert.setString(2, psIpSP);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_SUPERPEER_IP+"> creado
<"+psIpSP+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta el Puerto del SuperPeer
                loInsert.setString(1,
PFC_Constantes.CONFIGURACION_SUPERPEER_PUERTO);
                loInsert.setString(2, psPuertoSP);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_SUPERPEER_PUERTO+">
creado <"+psPuertoSP+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta directorio
                loInsert.setString(1, PFC_Constantes.CONFIGURACION_DIRECTORIO);
                loInsert.setString(2, psDirectorio);

        moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_DIRECTORIO+"> creado
<"+psDirectorio+">");
                loInsert.executeUpdate();
                loInsert.clearParameters();

                //Se inserta parametro que indica que no hay cambios aplicar
                loInsert.setString(1, PFC_Constantes.CONFIGURACION_CAMBIOS);
                loInsert.setString(2, "N");
                moLog.debug("Parametro<"+PFC_Constantes.CONFIGURACION_CAMBIOS+">
creado <N>");
                loInsert.executeUpdate();
                loInsert.clearParameters();

            }catch (Exception e)
            {
                moLog.error("Problemas al insertar la configuracion:"+e.getMessage());
                throw new Exception("No se he podido generar la configuracion!");
```

```
                }
                finally
                {
                        loInsert.close();
                        loInsert = null;
                }

        //Se crea la tabla documentos
        try
        {
                loCreate = moConexion.createStatement();

                        moLog.info("Peticion de crear tabla Documentos para Peer");

                lsTabla = "CREATE TABLE DOCUMENTOS  "
        + "(DOC_IDDOCUMENTO VARCHAR(111) NOT NULL,"
          + " DOC_PROPIETARIO VARCHAR(80) NOT NULL,"
          + " DOC_NOMBRE VARCHAR(35) NOT NULL,"
          + " DOC_RUTA VARCHAR(300) NOT NULL,"
          + " DOC_FECHA TIMESTAMP NOT NULL,"
          + " PRIMARY KEY(DOC_IDDOCUMENTO)) " ;

                        //Se crea la tabla
        loCreate.execute(lsTabla);

                        moLog.debug("Tabla DOCUMENTOS creada!");

                        //Sentencia para crear la tabla de DOC_CAMBIOS para el SuperPeer
                        moLog.info("Peticion de crear tabla DOC_CAMBIOS");

                lsTabla = "CREATE TABLE DOC_CAMBIOS  "
        + "(CAM_ID VARCHAR(63) NOT NULL,"
          + " CAM_IDDOC VARCHAR(111) NOT NULL,"
          + " CAM_TIPO VARCHAR(1) NOT NULL,"
          + " CAM_FECHA TIMESTAMP NOT NULL,"
          + " CAM_NODO VARCHAR(50),"
          + " CAM_SUBTIPO VARCHAR(1),"
          + " CAM_CONTENIDO VARCHAR(1000),"
          + " CAM_ORDEN INTEGER,"
          + " PRIMARY KEY(CAM_ID))";

                //Se crea la tabla
        loCreate.execute(lsTabla);

                        moLog.debug("Tabla DOC_CAMBIOS creada!");

                } catch (SQLException e)
                {
                        moLog.error("Problemas al crear las tablas de documentos y
cambios:"+e.getMessage());
                        try
                        {
                                loCreate.close();
                                loCreate = null;
                        } catch (SQLException e1)
                        {
                                moLog.error("Problemas al cerrar el Statement:"+e1.getMessage());

                        }
                        throw new Exception("No se he podido generar la configuracion!");
                }


    }//inicializaSistema

    /**
     * <p>Obtiene la configuracion y la cargas en las variables de trabajo</p>
     * @throws Exception
     */
    public void cargaConfiguracion() throws Exception
```

```
{
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;

                try
                {
                        //Una vez creada la tabla, se insertan los valores de configuracion
                        lsConsulta = "SELECT CONFIG_VALOR FROM CONFIGURACION WHERE
CONFIG_PARAMETRO = ?";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        //Se consulta nombre del peer
                        loConsulta.setString(1, PFC_Constantes.CONFIGURACION_NOMBRE_PEER);
                        loResultado = loConsulta.executeQuery();

                        try
                        {
                                loResultado.next();
                                PFC_Constantes.NODO_NOMBRE =
loResultado.getString("CONFIG_VALOR");
                                moLog.debug("Carga de configuracion. Nombre del nodo
cargado<"+PFC_Constantes.NODO_NOMBRE+">");
                        }catch(Exception e)
                        {
                                moLog.error("Problemas al obtener de configuracion el nombre del
Nodo:"+e.getMessage());
                                throw new Exception("Problemas al obtener de configuracion el
nombre del Nodo");
                        }

                        //Se consulta identificador del peer
                        loConsulta.setString(1, PFC_Constantes.CONFIGURACION_ID_PEER);
                        loResultado = loConsulta.executeQuery();

                        try
                        {
                                loResultado.next();
                                PFC_Constantes.NODO_ID =
loResultado.getString("CONFIG_VALOR");
                                moLog.debug("Carga de configuracion. ID del nodo
cargado<"+PFC_Constantes.NODO_ID+">");
                        }catch(Exception e)
                        {
                                moLog.error("Problemas al obtener de configuracion el identificador
del Nodo:"+e.getMessage());
                                throw new Exception("Problemas al obtener de configuracion el
identificador del Nodo");
                        }

                        //Se consulta nombre del grupo
                        loConsulta.setString(1, PFC_Constantes.CONFIGURACION_NOMBRE_GRUPO);
                        loResultado = loConsulta.executeQuery();

                        try
                        {
                                loResultado.next();
                                PFC_Constantes.GRUPO_TRABAJO_NOMBRE =
loResultado.getString("CONFIG_VALOR");
                                moLog.debug("Carga de configuracion. Nombre del grupo
cargado<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        }catch(Exception e)
                        {
                                moLog.error("Problemas al obtener de configuracion el nombre del
grupo:"+e.getMessage());
                                throw new Exception("Problemas al obtener de configuracion el
nombre del Grupo");
                        }
```

```
//Se consulta identificador del Pipe Multicast
loConsulta.setString(1,
PFC_Constantes.CONFIGURACION_ID_PIPE_MULTICAST);
loResultado = loConsulta.executeQuery();

try
{
	loResultado.next();
	PFC_Constantes.PIPE_MULTICAST_ID =
loResultado.getString("CONFIG_VALOR");
	moLog.debug("Carga de configuracion. ID del Pipe Multicast
cargado<"+PFC_Constantes.PIPE_MULTICAST_ID+">");
}catch(Exception e)
{
	moLog.error("Problemas al obtener de configuracion el ID del Pipe
Multicast:"+e.getMessage());
	throw new Exception("Problemas al obtener de configuracion el ID
del Pipe Multicast");
}

//Se consulta identificador del Pipe Unicast
loConsulta.setString(1,
PFC_Constantes.CONFIGURACION_ID_PIPE_UNICAST);
loResultado = loConsulta.executeQuery();

try
{
	loResultado.next();
	PFC_Constantes.PIPE_UNICAST_ID =
loResultado.getString("CONFIG_VALOR");
	moLog.debug("Carga de configuracion. ID del Pipe Unicast
cargado<"+PFC_Constantes.PIPE_UNICAST_ID+">");
}catch(Exception e)
{
	moLog.error("Problemas al obtener de configuracion el ID del Pipe
Unicast:"+e.getMessage());
	throw new Exception("Problemas al obtener de configuracion el ID
del Pipe Unicast");
}

//Se consulta IP del SuperPeer
loConsulta.setString(1, PFC_Constantes.CONFIGURACION_SUPERPEER_IP);
loResultado = loConsulta.executeQuery();

try
{
	loResultado.next();
	PFC_Constantes.SUPERPEER_IP =
loResultado.getString("CONFIG_VALOR");
	moLog.debug("Carga de configuracion. IP del SuperPeer
cargado<"+PFC_Constantes.SUPERPEER_IP+">");
}catch(Exception e)
{
	moLog.error("Problemas al obtener de configuracion la IP del
SuperPeer:"+e.getMessage());
	throw new Exception("Problemas al obtener de configuracion la IP
del SuperPeer");
}

//Se consulta Puerto del SuperPeer
loConsulta.setString(1,
PFC_Constantes.CONFIGURACION_SUPERPEER_PUERTO);
loResultado = loConsulta.executeQuery();

try
{
	loResultado.next();
	PFC_Constantes.SUPERPEER_PUERTO =
loResultado.getString("CONFIG_VALOR");
```

```
                                        moLog.debug("Carga de configuracion. Puerto del SuperPeer
cargado<"+PFC_Constantes.SUPERPEER_PUERTO+">");
                                }catch(Exception e)
                                {
                                        moLog.error("Problemas al obtener de configuracion la Puerto del
SuperPeer:"+e.getMessage());
                                        throw new Exception("Problemas al obtener de configuracion la
Puerto del SuperPeer");
                                }

                                //Se consulta directorio
                                loConsulta.setString(1, PFC_Constantes.CONFIGURACION_DIRECTORIO);
                                loResultado = loConsulta.executeQuery();

                                try
                                {
                                        loResultado.next();
                                        PFC_Constantes.DIRECTORIO =
loResultado.getString("CONFIG_VALOR");
                                        moLog.debug("Carga de configuracion. Directorio
cargado<"+PFC_Constantes.DIRECTORIO+">");
                                }catch(Exception e)
                                {
                                        moLog.error("Problemas al obtener de configuracion el
Directorio:"+e.getMessage());
                                        throw new Exception("Problemas al obtener de configuracion el
Directorio");
                                }

                                //Se consulta si hay cambios aplicar
                                loConsulta.setString(1, PFC_Constantes.CONFIGURACION_CAMBIOS);
                                loResultado = loConsulta.executeQuery();

                                try
                                {
                                        loResultado.next();
                                        String lsResultado = loResultado.getString("CONFIG_VALOR");
                                        PFC_Constantes.CAMBIO_CONFIGURACION = (lsResultado == null ||
lsResultado.equals("N") ) ? false : true;
                                        moLog.debug("Carga de configuracion.
cambios<"+PFC_Constantes.CAMBIO_CONFIGURACION+">");
                                }catch(Exception e)
                                {
                                        moLog.error("Problemas al obtener de configuracion el parametro
cambios:"+e.getMessage());
                                        throw new Exception("Problemas al obtener de configuracion el
parametro cambios");
                                }
                        }catch (Exception e)
                        {
                                moLog.error("Problemas al cargar la configuracion:"+e.getMessage());
                                throw new Exception("No se he podido cargar la configuracion!");
                        }
                        finally
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        }

    }//cargaConfiguracion


    /**
     * <p>Verifica si una determinada tabla ya existe en la base de datos</p>
     * @param psNomTabla
     * @return
     */
```

```
	public boolean existeTabla(String psNomTabla)
	{
		DatabaseMetaData loMetadata = null;
		String[] lvsTipo = { "TABLE" };
		ResultSet loResultado = null;
		boolean lbSalir = false;
		String lsTabla = null;

		try
		{
				loMetadata = moConexion.getMetaData();

				loResultado = loMetadata.getTables(null, null, null, lvsTipo);

				//Recorremos tablas existentes en busca del objeto de la busqueda
				while ( (!lbSalir) && (loResultado.next()) )
				{
					lsTabla = loResultado.getString("TABLE_NAME");
					if ( (lsTabla != null) && (lsTabla.equalsIgnoreCase(psNomTabla)) )
					{
						lbSalir = true;
						moLog.debug("Tabla<"+psNomTabla+"> SI que existe en
base de datos.");
					}
				}

				if (!lbSalir)
				{
					moLog.debug("Tabla<"+psNomTabla+"> NO existe en base de
datos.");
				}


			} catch (SQLException e)
			{
				moLog.error("Problemas al verificar si existe tabla.", e);
			}
		finally
		{
			try {
					loResultado.close();
					loResultado = null;
				} catch (SQLException e)
				{
					moLog.error("Problemas al cerrar el resultset:", e);

				}
		}

		return lbSalir;

	}//existeTabla

	/**
	 * <p>Obtiene todos los documentos existentes en la base de datos</p>
	 * <p>Se crea un objeto tipo PFC_Documentos que contiene todo el listado de documentos
existentes.</p>
	 * @return
	 */
	public PFC_Documentos obtenerDocumentos()
	{
		//Sentencia a ejecutar
		PreparedStatement loConsulta = null;
		ResultSet loResultado = null;
		String lsConsulta = null;
		PFC_Documentos moDocumentos = null;

			try
			{
				//Una vez creada la tabla, se insertan los valores de configuracion
```

```
                        lsConsulta = "SELECT DOC_IDDOCUMENTO, DOC_PROPIETARIO,
DOC_NOMBRE, DOC_RUTA, DOC_FECHA FROM DOCUMENTOS ORDER BY DOC_FECHA DESC";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        //Se consultaN los documentos
                        loResultado = loConsulta.executeQuery();

                        moDocumentos = new PFC_Documentos();
                        while (loResultado.next())
                        {
                                moDocumentos.addFila();

                                //Se obtiene el identificador

        moDocumentos.addElemento(loResultado.getString("DOC_IDDOCUMENTO"));

                                //Se obtiene el propietario

        moDocumentos.addElemento(loResultado.getString("DOC_PROPIETARIO"));

                                //Se obtiene el nombre

        moDocumentos.addElemento(loResultado.getString("DOC_NOMBRE"));

                                //Se obtiene la ruta
                                moDocumentos.addElemento(loResultado.getString("DOC_RUTA"));

                                //Se obtiene la fecha actualizado

        moDocumentos.addElemento(loResultado.getTimestamp("DOC_FECHA"));


                                moDocumentos.addRegistro();
                        }//while

                }catch (Exception e)
                {
                        moLog.error("Problemas al obtener los documentos:"+e.getMessage());
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

                return moDocumentos;

    }//obtenerDocumentos

    /**
     * <p>Da de alta un documento en la tabla documentos.</p>
     * <p>Previamente verifica que no exista dicho nombre/ruta de documento. En tal caso propaga
error</p>
     * @param psIdDoc Identificador del documento
     * @param psPropietario Identificador del propietario
     * @param psNombre Nombre del documento
     * @param psRuta Ruta del fichero
     * @param psFecha Fecha alta
     * @return
     * @throws Exception
     */
```

```
   public void guardarDocumento(String psIdDoc, String psPropietario, String psNombre, String
psRuta,
                    Date poFecha) throws Exception
   {
          //Sentencia a ejecutar
          PreparedStatement loConsulta = null;
          PreparedStatement loInsert = null;
          ResultSet loResultado = null;
          String lsConsulta = null;
          String lsInsert = null;

                    try
                    {
                              moLog.debug("Guardar Documento - Peticion nuevo documento.
IdDoc<"+psIdDoc+"> Propietario<"+psPropietario+">"+
                                        " Nombre<"+psNombre+"> Ruta<"+psRuta+">
Fecha<"+poFecha+">");

                              // Se verifica si existe un documento con el mismo nombre y
                              lsConsulta = "SELECT DOC_IDDOCUMENTO FROM DOCUMENTOS WHERE
DOC_NOMBRE = ? AND DOC_PROPIETARIO = ?";

                              loConsulta = moConexion.prepareStatement(lsConsulta);

                              // Se consulta existencia del documento
                              loConsulta.setString(1, psNombre);
                              loConsulta.setString(2, psPropietario);
                              loResultado = loConsulta.executeQuery();


                              if (loResultado.next())
                              {
                                        // En caso de existir, indicamos el error
                                        moLog.debug("Guardar Documento - Documento con
nombre<"+psNombre+"> YA existe en BBDD!");
                                        throw new Exception("Documento<"+psNombre+"> YA existe en
base de datos.");
                              }

                              //Se da de alta un nuevo documento en la base de datos
                              lsInsert = "insert into documentos
(doc_iddocumento,doc_propietario,doc_nombre,doc_ruta,doc_fecha)"+
                                        " values(?, ?, ?, ?, ?)";

                              loInsert = moConexion.prepareStatement(lsInsert);

                              //Se inserta nombre del peer
                              loInsert.setString(1, psIdDoc);
                              loInsert.setString(2, psPropietario);
                              loInsert.setString(3, psNombre);
                              loInsert.setString(4, psRuta);
                              //loInsert.setDate(5, poFecha);
                              loInsert.setTimestamp(5, new Timestamp(poFecha.getTime()));

                              loInsert.executeUpdate();


                    }catch (Exception e)
                    {
                              moLog.error("Problemas al dar de alta el documento:"+e.getMessage());
                              throw e;
                    }
                    finally
                    {
                              try
                              {
                                        loResultado.close();
                                        loResultado = null;
                                        loConsulta.close();
                                        loConsulta = null;
```

```
                                loInsert.close();
                                loInsert = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

    }//guardarDocumento

    /**
     * <p>Da de alta un documento en la tabla documentos para SuperPeer</p>
     * <p>SuperPeer lo que guardara sera censo de documentos con su propietario. No guarda copia
local del documentos</p>
     * @param psIdDoc Identificador del documento
     * @param psPropietario Identificador del propietario
     * @param psNombre Nombre del documento
     * @param psFecha Fecha alta
     * @param psRuta Ruta
     * @return
     * @throws Exception
     */
    public void guardarDocumentoSP(String psIdDoc, String psPropietario, String psNombre, Date
poFecha, String psRuta) throws Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loInsert = null;
        String lsInsert = null;

                try
                {
                        moLog.debug("Guardar Documento - Peticion nuevo documento en SuperPeer.
IdDoc<"+psIdDoc+"> Propietario<"+psPropietario+">"+
                                        " Nombre<"+psNombre+"> Fecha<"+poFecha+">
Ruta<"+psRuta+">");

                        //Se da de alta un nuevo documento en la base de datos
                        lsInsert = "insert into documentos
(doc_iddocumento,doc_propietario,doc_nombre,doc_fecha, doc_ruta) values(?, ?, ?, ?, ?)";

                        loInsert = moConexion.prepareStatement(lsInsert);

                        //Se inserta nombre del peer
                        loInsert.setString(1, psIdDoc);
                        loInsert.setString(2, psPropietario);
                        loInsert.setString(3, psNombre);
                        loInsert.setTimestamp(4, new Timestamp(poFecha.getTime()));
                        loInsert.setString(5, psRuta);

                        loInsert.executeUpdate();


                }catch (Exception e)
                {
                        moLog.error("Problemas al dar de alta el documento para
SuperPeer:"+e.getMessage());
                        throw e;
                }
                finally
                {
                        try
                        {
                                loInsert.close();
                                loInsert = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }
```

```
    }//guardarDocumentoSP
    /**
     * <p>Modificacion informacion de un documento.</p>
     * <p>Los datos del documento que se pueden modificar son la Fecha y si se notificado el
cambio</p>
     * @param psIdDoc Identificador del documento
     * @param psPropietario Identificador del propietario
     * @param psFecha Fecha modificacion
     * @return
     * @throws Exception
     */
    public void modificarDocumento(String psIdDoc, String psPropietario, Date poFecha) throws
Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loUpdate = null;
        String lsUpdate = null;


            try
            {
                moLog.debug("Modificar Documento - Peticion modificar documento.
IdDoc<"+psIdDoc+"> Propietario<"+psPropietario+">"+
                        " Fecha<"+poFecha+">");


                //Se da de alta un nuevo documento en la base de datos
                lsUpdate = "update documentos set doc_fecha = ? where doc_iddocumento =
?";


                loUpdate = moConexion.prepareStatement(lsUpdate);

                loUpdate.setTimestamp(1, new Timestamp(poFecha.getTime()));
                loUpdate.setString(2, psIdDoc);

                loUpdate.executeUpdate();


            }catch (Exception e)
            {
                moLog.error("Problemas al modificar el documento:"+e.getMessage());
                throw e;
            }
            finally
            {
                try
                {
                    loUpdate.close();
                    loUpdate = null;
                } catch (SQLException e)
                {
                    moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                }
            }

    }//modificarDocumento

    /**
     * <p>Da de alta un cambio en la tabla DOC_CAMBIOS</p>
     * @param psIdDoc
     * @param psTipo
     * @param poFecha
     * @param psNodo
     * @param psSubtipo
     * @param psContenido
     * @param psOrden
     * @throws Exception
     */
```

```
	public void altaNotificacion(String psIdDoc, String psTipo, Date poFecha, String psNodo, String psSubtipo,
		String psContenido, String psOrden) throws Exception
	{
		//Sentencia a ejecutar
		PreparedStatement loInsert = null;
		String lsInsert = null;
		String lsNotId = "";

			try
			{
				// En el caso de realizarse una modificacion, el identificador tendra concatenado el nodo
				if (psTipo.equals("M"))
					lsNotId = psNodo;

				lsNotId += System.currentTimeMillis();
				moLog.debug("Alta nuevo cambio. IdDoc<"+psIdDoc+"> Tipo<"+psTipo+"> NotId<"+lsNotId+"> Fecha<"+poFecha+">"+
						"Nodo<"+psNodo+"> Subtipo<"+psSubtipo+"> Orden<"+psOrden+">");

				//Se da de alta un nuevo documento en la base de datos
				lsInsert = "insert into doc_cambios (cam_id, cam_iddoc, cam_tipo, cam_fecha, cam_nodo, cam_subtipo,"+
					"cam_contenido, cam_orden) values(?, ?, ?, ?, ?, ?, ?, ?)";

				loInsert = moConexion.prepareStatement(lsInsert);

				//Se inserta nombre del peer
				loInsert.setString(1, lsNotId);
				loInsert.setString(2, psIdDoc);
				loInsert.setString(3, psTipo);
				loInsert.setTimestamp(4, new Timestamp(poFecha.getTime()));
				loInsert.setString(5, psNodo);
				loInsert.setString(6, psSubtipo);
				loInsert.setString(7, psContenido);
				loInsert.setInt(8, (psOrden != null ? new Integer(psOrden) : 0));

				loInsert.executeUpdate();


			}catch (Exception e)
			{
				moLog.error("Problemas al dar de alta una notificacion:"+e);
				throw e;
			}
			finally
			{
				try
				{
					loInsert.close();
					loInsert = null;
				} catch (SQLException e)
				{
					moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
				}
			}

	}//altaNotificacion


	/**
	 * <p>Elimina el documento.</p>
	 * <p>Elimina el regsitro definitivamente</p>
	 * @param psIdDoc Identificador del documento
	 * @return
	 * @throws Exception
	 */
	public void eliminarDocumento(String psIdDoc) throws Exception
```

```
{
        //Sentencia a ejecutar
        PreparedStatement loDelete = null;
        String lsDelete = null;


                try
                {
                        moLog.debug("Eliminar Documento - Peticion de eliminar documento.
IdDoc<"+psIdDoc+">");


                        //Se da de alta un nuevo documento en la base de datos
                        lsDelete = "delete from documentos where doc_iddocumento = ? ";

                        loDelete = moConexion.prepareStatement(lsDelete);

                        loDelete.setString(1, psIdDoc);

                        loDelete.executeUpdate();


                }catch (Exception e)
                {
                        moLog.error("Problemas al eliminar el documento:"+e.getMessage());
                        throw e;
                }
                finally
                {
                        try
                        {
                                loDelete.close();
                                loDelete = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

}//eliminarDocumento

/**
 * <p>Elimina cualquier cambio asociado a un documento.</p>
 * <p>En caso de psIdDoc==null, se eliminan todos los cambios</p>
 * @param psIdDoc Identificador del documento
 * @return
 * @throws Exception
 */
public void eliminarCambios(String psIdDoc) throws Exception
{
        //Sentencia a ejecutar
        PreparedStatement loDelete = null;
        String lsDelete = null;


                try
                {
                        moLog.debug("Eliminar Documento - Peticion de eliminar cambios.
IdDoc<"+psIdDoc+">");


                        //Se da de alta un nuevo documento en la base de datos
                        if (psIdDoc != null)
                                lsDelete = "delete from doc_cambios where cam_iddoc = ? ";
                        else
                                lsDelete = "delete from doc_cambios";
                        loDelete = moConexion.prepareStatement(lsDelete);

                        if (psIdDoc != null)
                                loDelete.setString(1, psIdDoc);
```

```
                loDelete.executeUpdate();

            }catch (Exception e)
            {
                moLog.error("Problemas al eliminar cambios:"+e.getMessage());
                throw e;
            }
            finally
            {
                try
                {
                    loDelete.close();
                    loDelete = null;
                } catch (SQLException e)
                {
                    moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                }
            }

    }//eliminarCambios


    /**
     * <p>Modificacion informacion de configuracion. Cambio del directorio</p>
     * @param psDirectorio Nuevo directorio
     * @return
     * @throws Exception
     */
    public void modificarDirectorio(String psDirectorio) throws Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loUpdate = null;
        String lsUpdate = null;


            try
            {
                moLog.debug("Modificar Configuracion - Nuevo directorio.
Directorio<"+psDirectorio+">");


                //Se modifica el parametro
                lsUpdate = "update configuracion set CONFIG_VALOR = ? where
CONFIG_PARAMETRO = ?";

                loUpdate = moConexion.prepareStatement(lsUpdate);

                loUpdate.setString(1, psDirectorio);
                loUpdate.setString(2, PFC_Constantes.CONFIGURACION_DIRECTORIO);

                loUpdate.executeUpdate();


            }catch (Exception e)
            {
                moLog.error("Problemas al modificar el directorio:"+e.getMessage());
                throw e;
            }
            finally
            {
                try
                {
                    loUpdate.close();
                    loUpdate = null;
                } catch (SQLException e)
                {
                    moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                }
```

```
                }

    }//modificarDirectorio

    /**
     * <p>Modificacion informacion de configuracion. Cambio de la IP del SuperPeer.</p>
     * @param psSuperPeerIP Nueva IP del SuperPeer
     * @return
     * @throws Exception
     */
    public void modificarSuperPeerIP(String psSuperPeerIP) throws Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loUpdate = null;
        String lsUpdate = null;


                try
                {
                        moLog.debug("Modificar Configuracion - Nueva IP del SuperPeer.
IP<"+psSuperPeerIP+">");


                        //Se modifica el parametro
                        lsUpdate = "update configuracion set CONFIG_VALOR = ? where
CONFIG_PARAMETRO = ?";

                        loUpdate = moConexion.prepareStatement(lsUpdate);

                        loUpdate.setString(1, psSuperPeerIP);
                        loUpdate.setString(2, PFC_Constantes.CONFIGURACION_SUPERPEER_IP);

                        loUpdate.executeUpdate();


                }catch (Exception e)
                {
                        moLog.error("Problemas al modificar la IP del SuperPeer:"+e.getMessage());
                        throw e;
                }
                finally
                {
                        try
                        {
                                loUpdate.close();
                                loUpdate = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

    }//modificarSuperPeerIP

    /**
     * <p>Modificacion informacion de configuracion. Cambio del Puerto del SuperPeer</p>
     * @param psSuperPeerPuerto Nuevo Puerto del SuperPeer
     * @return
     * @throws Exception
     */
    public void modificarSuperPeerPuerto(String psSuperPeerPuerto) throws Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loUpdate = null;
        String lsUpdate = null;


                try
                {
```

```
                        moLog.debug("Modificar Configuracion - Nuevo Puerto del SuperPeer.
Puerto<"+psSuperPeerPuerto+">");


                        //Se modifica el parametro
                        lsUpdate = "update configuracion set CONFIG_VALOR = ? where
CONFIG_PARAMETRO = ?";

                        loUpdate = moConexion.prepareStatement(lsUpdate);

                        loUpdate.setString(1, psSuperPeerPuerto);
                        loUpdate.setString(2,
PFC_Constantes.CONFIGURACION_SUPERPEER_PUERTO);

                        loUpdate.executeUpdate();


                }catch (Exception e)
                {
                        moLog.error("Problemas al modificar el Puerto del
SuperPeer:"+e.getMessage());
                        throw e;
                }
                finally
                {
                        try
                        {
                                loUpdate.close();
                                loUpdate = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

    }//modificarSuperPeerPuerto

    /**
     * <p>Modificacion informacion de configuracion. Cambio del parametro que indica que hay cambios
en la configuracion</p>
     * @param psCambio S/N indicando si hay o no que aplicar cambios
     * @return
     * @throws Exception
     */
    public void modificarCambio(String psCambio) throws Exception
    {
        //Sentencia a ejecutar
        PreparedStatement loUpdate = null;
        String lsUpdate = null;


                try
                {
                        moLog.debug("Modificar Configuracion - Cambio en la
configuracion<"+psCambio+">");


                        //Se modifica el parametro
                        lsUpdate = "update configuracion set CONFIG_VALOR = ? where
CONFIG_PARAMETRO = ?";

                        loUpdate = moConexion.prepareStatement(lsUpdate);

                        loUpdate.setString(1, psCambio);
                        loUpdate.setString(2, PFC_Constantes.CONFIGURACION_CAMBIOS);

                        loUpdate.executeUpdate();


                }catch (Exception e)
```

```
                {
                        moLog.error("Problemas al modificar el parametro cambio:"+e.getMessage());
                        throw e;
                }
                finally
                {
                        try
                        {
                                loUpdate.close();
                                loUpdate = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

    }//modificarCambio


    /**
     * <p>Obtiene todos los documentos que no son de la propiedad del nodo indicado por
parametro</p>
     * <p>Se crea un objeto tipo PFC_Mensaje listo para poder ser enviado al SP</p>
     * @param psPropietario id unico del Peer que no queremos obtener los documentos
     * @return
     */
    public Hashtable obtenerMisNoDocumentos(String psPropietario)
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        Hashtable loDocumentos = null;
        String[] lvsDatos = null;
        String lsKey = null;

                try
                {
                        moLog.info("Peticion para obtene documentos que no sean del
<"+psPropietario+">");
                        //Una vez creada la tabla, se insertan los valores de configuracion
                        lsConsulta = "SELECT DOC_IDDOCUMENTO, DOC_PROPIETARIO,
DOC_NOMBRE, DOC_FECHA FROM DOCUMENTOS WHERE DOC_PROPIETARIO <> ?";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        // Buscamos todos los documentos que no sean propiedad del nodo
peticionario
                        loConsulta.setString(1, psPropietario);

                        //Se consultaN los documentos
                        loResultado = loConsulta.executeQuery();

                        loDocumentos = new Hashtable();
                        while (loResultado.next())
                        {
                                lvsDatos = new String[3];

                                //Se obtiene el identificador
                                lsKey = loResultado.getString("DOC_IDDOCUMENTO");

                                //Se obtiene el propietario
                                lvsDatos[0] = loResultado.getString("DOC_PROPIETARIO");

                                //Se obtiene el nombre
                                lvsDatos[1] = loResultado.getString("DOC_NOMBRE");

                                //Se obtiene la fecha actualizado
                                lvsDatos[2] = ""+loResultado.getTimestamp("DOC_FECHA");
```

```
                                        loDocumentos.put(lsKey, lvsDatos);
                                }//while

                                moLog.info("Documentos que no sean del
<"+psPropietario+">=<"+(loDocumentos != null ? loDocumentos.size() : 0)+">");

                        }catch (Exception e)
                        {
                                moLog.error("Problemas al obtener los documentos NO propiedad del
nodo:"+e.getMessage());
                        }
                        finally
                        {
                                try
                                {
                                        loResultado.close();
                                        loResultado = null;
                                        loConsulta.close();
                                        loConsulta = null;
                                } catch (SQLException e)
                                {
                                        moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                                }
                        }

                        return loDocumentos;

    }//obtenerMisNoDocumentos

    /**
     * <p>Obtiene los cambios que se han realizado en un determinado documento a partir de una
fecha</p>
     * @param psIdDoc
     * @param psTipo
     * @param poFecha
     * @return
     */
    public Vector obtenerCambiosDocumento(String psIdDoc, String psTipo, Date poFecha)
    {
            //Sentencia a ejecutar
            PreparedStatement loConsulta = null;
            ResultSet loResultado = null;
            String lsConsulta = null;
            Vector loCambios = null;
            String[] lvsDatos = null;


                        try
                        {
                                moLog.info("Peticion para obtener cambios sobre el
documento<"+psIdDoc+"> a partir de fecha<"+new Timestamp(poFecha.getTime())+"> y del
tipo<"+psTipo+">");
                                //Una vez creada la tabla, se insertan los valores de configuracion
                                lsConsulta = "SELECT CAM_NODO, CAM_SUBTIPO, CAM_CONTENIDO FROM
DOC_CAMBIOS WHERE CAM_IDDOC = ? AND CAM_TIPO = ? AND CAM_FECHA > ? ORDER BY
CAM_FECHA, CAM_ORDEN";

                                loConsulta = moConexion.prepareStatement(lsConsulta);

                                // Buscamos todos los documentos que no sean propiedad del nodo
peticionario
                                loConsulta.setString(1, psIdDoc);
                                loConsulta.setString(2, psTipo);
                                loConsulta.setTimestamp(3, new Timestamp(poFecha.getTime()));

                                //Se consultaN los documentos
                                loResultado = loConsulta.executeQuery();

                                loCambios = new Vector();
                                while (loResultado.next())
```

```
                        {
                                lvsDatos = new String[3];

                                //Se obtiene el identificador de nodo
                                lvsDatos[0] = loResultado.getString("CAM_NODO");

                                //Se obtiene el tipo del cambio
                                lvsDatos[1] = loResultado.getString("CAM_SUBTIPO");

                                //Se obtiene el contenido
                                lvsDatos[2] = loResultado.getString("CAM_CONTENIDO");


                                loCambios.add(lvsDatos);
                        }//while

                        moLog.info("Numero de cambios <"+(loCambios != null ? loCambios.size() :
0)+"> del Documento<"+psIdDoc+">");

                }catch (Exception e)
                {
                        moLog.error("Problemas al obtener los cambios del documento:"+e);
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

                return loCambios;

    }//obtenerCambiosDocumento

    /**
     * <p>Obtiene los cambios que se han realizado en cualquier documento</p>
     * @param psIdDoc
     * @param psTipo
     * @param poFecha
     * @return
     */
    public Hashtable obtenerCambiosDocs()
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        Vector loCambios = null;
        Hashtable loDocs = null;
        String[] lvsDatos = null;
        String lsIdDoc = null;
        String lsIdTemp = null;
        boolean lbPrimer = true;

                try
                {
                        moLog.info("Peticion para obtener cambios realizados en cualquier
documento");
                        //Cambios en los documentos
                        lsConsulta = "SELECT CAM_IDDOC, CAM_NODO, CAM_SUBTIPO,
CAM_CONTENIDO, CAM_FECHA, CAM_ORDEN FROM DOC_CAMBIOS WHERE CAM_TIPO = 'M' ORDER BY
CAM_IDDOC, CAM_FECHA, CAM_ORDEN";
```

```
loConsulta = moConexion.prepareStatement(lsConsulta);

//Se consultaN los documentos
loResultado = loConsulta.executeQuery();

loDocs = new Hashtable();
loCambios = new Vector();
while (loResultado.next())
{
	lvsDatos = new String[5];

	// Se obtiene id del documento
	lsIdDoc = loResultado.getString("CAM_IDDOC");

	if (lbPrimer)
	{
		lsIdTemp = lsIdDoc;
		lbPrimer = false;
	}

	//Se obtiene el identificador de nodo
	lvsDatos[0] = loResultado.getString("CAM_NODO");

	//Se obtiene el tipo del cambio
	lvsDatos[1] = loResultado.getString("CAM_SUBTIPO");

	//Se obtiene el contenido
	lvsDatos[2] = loResultado.getString("CAM_CONTENIDO");

	//Se obtiene la fecha
	lvsDatos[3] = ""+loResultado.getTimestamp("CAM_FECHA");

	//Se obtiene el orden del cambio
	lvsDatos[4] = ""+loResultado.getInt("CAM_ORDEN");

	if (!lsIdTemp.equals(lsIdDoc))
	{
		loDocs.put(lsIdTemp, loCambios);
		lsIdTemp = lsIdDoc;
		loCambios = new Vector();

	}

	loCambios.add(lvsDatos);
}//while

//Se añade el ultimo doc en caso de que se haya obtenido algo
if (!lbPrimer)
	loDocs.put(lsIdDoc, loCambios);

moLog.info("Numero de cambios realizados en documentos <"+(loCambios !=
null ? loCambios.size() : 0)+">");

}catch (Exception e)
{
	moLog.error("Problemas al obtener los cambios de los
documentos:"+e.getMessage());
}
finally
{
	try
	{
		loResultado.close();
		loResultado = null;
		loConsulta.close();
		loConsulta = null;
	} catch (SQLException e)
	{
		moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
	}
```

```
                }

                return loDocs;

    }//obtenerCambiosDocs

    /**
     * <p>Obtiene los cambios tipo Alta (A) realizados sobre los documentos mientras un peer ha estado
desconectado</p>
     * @return
     */
    public Vector obtenerCambiosAlta()
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        Vector loCambios = null;
        String[] lvsDatos = null;

                try
                {
                        moLog.info("Peticion para obtener cambios tipo Alta realizados sobre los
documentos.");
                        //Consulta
                        lsConsulta = "SELECT DOC.DOC_IDDOCUMENTO, DOC.DOC_NOMBRE,
DOC.DOC_RUTA, DOC.DOC_FECHA FROM DOCUMENTOS DOC, DOC_CAMBIOS CAM WHERE
CAM.CAM_IDDOC = DOC.DOC_IDDOCUMENTO AND CAM.CAM_TIPO='A'";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        //Se consultaN los documentos
                        loResultado = loConsulta.executeQuery();

                        loCambios = new Vector();
                        while (loResultado.next())
                        {
                                lvsDatos = new String[4];

                                //Se obtiene el identificador del documento
                                lvsDatos[0] = loResultado.getString("DOC_IDDOCUMENTO");

                                //Se obtiene el nombre del documento
                                lvsDatos[1] = loResultado.getString("DOC_NOMBRE");

                                //Se obtiene la ruta
                                lvsDatos[2] = loResultado.getString("DOC_RUTA");

                                // Se obtiene la fecha
                                lvsDatos[3] = ""+loResultado.getTimestamp("DOC_FECHA");

                                loCambios.add(lvsDatos);
                        }//while

                        moLog.info("Numero de altas <"+(loCambios != null ? loCambios.size() :
0)+">");

                }catch (Exception e)
                {
                        moLog.error("Problemas al obtener cambios tipo Alta en los
documentos:"+e.getMessage());
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
```

```
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

                return loCambios;

    }//obtenerCambiosAlta

    /**
     * <p>Obtiene los cambios tipo Baja (B) realizados sobre los documentos mientras un peer ha estado
desconectado</p>
     * @return
     */
    public Vector obtenerCambiosBaja()
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        Vector loCambios = null;
        String[] lvsDatos = null;

                try
                {
                        moLog.info("Peticion para obtener cambios tipo Baja realizados sobre los
documentos.");
                        //Consulta
                        lsConsulta = "SELECT CAM_IDDOC FROM DOC_CAMBIOS WHERE
CAM_TIPO='B'";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        //Se consultaN los documentos
                        loResultado = loConsulta.executeQuery();

                        loCambios = new Vector();
                        while (loResultado.next())
                        {
                                lvsDatos = new String[1];

                                //Se obtiene el identificador del documento
                                lvsDatos[0] = loResultado.getString("CAM_IDDOC");


                                loCambios.add(lvsDatos);
                        }//while

                        moLog.info("Numero de bajas <"+(loCambios != null ? loCambios.size() :
0)+">");

                }catch (Exception e)
                {
                        moLog.error("Problemas al obtener cambios tipo Baja en los
documentos:"+e.getMessage());
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }
```

```
                return loCambios;

    }//obtenerCambiosBaja

    /**
     * <p>Obtiene el path y ruta de un documento y fecha</p>
     * @param psIdDoc Identificador del documento
     * @return String[]  donde String[0]= path y nombre del documento, String[1]=fecha
     */
    public Object[] obtenerDocumento(String psIdDoc)
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        Object[] lvsDocumento = null;

                try
                {
                        moLog.info("Peticion para obtene el path completo y nombre de un documento
con id <"+psIdDoc+">");
                        //Una vez creada la tabla, se insertan los valores de configuracion
                        lsConsulta = "SELECT DOC_RUTA, DOC_NOMBRE, DOC_FECHA,
DOC_PROPIETARIO FROM DOCUMENTOS WHERE DOC_IDDOCUMENTO = ?";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        // Buscamos el path completo del documento
                        loConsulta.setString(1, psIdDoc);

                        //Se consultaN los documentos
                        loResultado = loConsulta.executeQuery();

                        if (loResultado.next())
                        {
                                lvsDocumento = new Object[4];
                                //Se obtiene la ruta del documento
                                lvsDocumento[0] = loResultado.getString("DOC_RUTA");
                                // Se obtiene el nombre del documento
                                lvsDocumento[1] = loResultado.getString("DOC_NOMBRE");
                                //Se obtiene la fecha
                                lvsDocumento[2] =
loResultado.getTimestamp("DOC_FECHA").getTime();
                                // Se obtiene el propietario
                                lvsDocumento[3] = loResultado.getString("DOC_PROPIETARIO");

                        }//if

                        moLog.info("Nombre y ruta del documento con id<"+psIdDoc+"> es
<"+lvsDocumento[0]+lvsDocumento[1]+"> con fecha<"+lvsDocumento[2]+">");

                }catch (Exception e)
                {
                        moLog.error("Problemas al obtener el path completo de un
documento:"+e.getMessage());
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }
```

```
                return lvsDocumento;

    }//obtenerDocumento


    /**
     * <p>Verifica si existe un cambio pendiente de enviar al grupo que indique el alta de un
determinado documento</p>
     * @param psIdDoc Identificador del documento que se quiere verificar si hay o no alta
     * @return boolean true si existe alta pendiente de envio
     */
    public boolean existeAltaPendiente(String psIdDoc)
    {
        //Sentencia a ejecutar
        PreparedStatement loConsulta = null;
        ResultSet loResultado = null;
        String lsConsulta = null;
        boolean lbExiste = false;

                try
                {
                        moLog.info("Peticion para verificar si existe alta pendiente de envio del
documento <"+psIdDoc+">");
                        //consulta el doc_cambios
                        lsConsulta = "SELECT CAM_ID FROM DOC_CAMBIOS WHERE CAM_IDDOC = ?
AND CAM_TIPO = 'A'";

                        loConsulta = moConexion.prepareStatement(lsConsulta);

                        // Indicamos el iddocumento
                        loConsulta.setString(1, psIdDoc);

                        //Se verifica si hay o no resultado
                        loResultado = loConsulta.executeQuery();

                        if (loResultado.next())
                        {
                                lbExiste = true;
                        }
                        else
                        {
                                lbExiste = false;
                        }

                        moLog.info("Resultado de verificar si existe alta pendiente de envio del
doc<"+psIdDoc+"> es <"+lbExiste+">");

                }catch (Exception e)
                {
                        moLog.error("Problemas al verificar si existe alta
pendiente:"+e.getMessage());
                }
                finally
                {
                        try
                        {
                                loResultado.close();
                                loResultado = null;
                                loConsulta.close();
                                loConsulta = null;
                        } catch (SQLException e)
                        {
                                moLog.error("Problemas al cerrar el Prepared:"+e.getMessage());
                        }
                }

                return lbExiste;

    }//existeAltaPendiente
```

```
    /**
     * <p>Cierra conexion de base de datos</p>
     */
    public void cerrarConexion()
    {
        try
        {
                        if (moConexion != null)
                        {
                        moConexion.close();
                                moConexion = null;
                        }
                } catch (SQLException e)
                {
                        moLog.error("Problemas al cerrar la conexion con la base de datos. URL de la
base de datos<"+PFC_Constantes.BD_URL+">");
                }
    }//cerrarConexion
}
```

9.2.3 PEER

```
package pfc.prototipo.jxta.peer;

import java.io.DataInput;
import java.io.DataInputStream;
import java.io.DataOutput;
import java.io.DataOutputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.net.DatagramPacket;
import java.net.URI;
import java.util.Enumeration;
import java.util.Vector;

import org.apache.log4j.Logger;

import pfc.prototipo.PFC_Constantes;
import pfc.prototipo.PFC_PanelConectarConexion;
import pfc.prototipo.PFC_PanelConectarConexion.PFC_ConectarAccion;

import pfc.prototipo.bbdd.PFC_BDGestion;
import pfc.prototipo.jxta.PFC_JxtaUtilidades;
import pfc.prototipo.jxta.PFC_Mensaje;
import pfc.prototipo.rendimiento.PFC_Rendimiento;
import net.jxta.credential.AuthenticationCredential;
import net.jxta.credential.Credential;
import net.jxta.discovery.DiscoveryService;
import net.jxta.document.Advertisement;
import net.jxta.document.StructuredDocument;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
import net.jxta.membership.Authenticator;
import net.jxta.membership.MembershipService;
import net.jxta.peer.PeerID;
import net.jxta.peergroup.PeerGroup;
```

```
import net.jxta.peergroup.PeerGroupID;
import net.jxta.pipe.PipeService;
import net.jxta.platform.NetworkConfigurator;
import net.jxta.platform.NetworkManager;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.socket.JxtaMulticastSocket;
import net.jxta.socket.JxtaServerSocket;
import net.jxta.socket.JxtaSocket;

public class PFC_Peer
{
	//Trazas
	private final static Logger moLog = Logger.getLogger(PFC_Peer.class);

	//Red
	private NetworkManager moRed = null;
	//Servicio del Peer
	private DiscoveryService moSrvPeer = null;
	//Servicio de Grupo
	private DiscoveryService moSrvGrupoPeer = null;

	//Grupo de Peers
	private PeerGroup moGrupo = null;

	// Canal de comunicacion Multicast
	private JxtaMulticastSocket moSocketMC = null;

	//Gestion los mensajes de entrada por el canal Multicast
	private PFC_PGestorComunicacionMC  moGestioComunicacionMC = null;

	//Gestion los mensajes de entrada por el canal Unicast
	private PFC_PGestorComunicacion  moGestioComunicacion = null;

	// Canal de comunicaion Unicast
	private JxtaServerSocket moSocket = null;
	private PipeAdvertisement moAdvPipe = null;

   //Conexion con la base de datos
   private PFC_BDGestion moGestionBD = null;


	public PFC_Peer(PFC_BDGestion poGestionBD,
pfc.prototipo.PFC_PanelConectarConexion.PFC_ConectarAccion poPanel) throws Exception
	{
		NetworkConfigurator loRedConfig;
		File loConfigLocal = null;
		PeerGroup loRedGrupo = null;
		PeerGroup loTrabajoGrupo = null;
		Enumeration<Advertisement> loAnuncios = null;
		PeerGroupAdvertisement loAdGrupo = null;


		// Se inicializa objeto de conexion con base de datps
		moGestionBD = poGestionBD;

		// Indicador avance del proceso
		poPanel.cambiarProgreso(5);

		try
		{
			moLog.info("Se inicia conexion a la red con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
			//Iniciamos conexion a la red que se generara. En primer lugar se indica el
tipo de Peer que se quiere. En segundo el nombre
			//de la red que se crea, en tercer lugar se genera un fichero de configuracion
con el nombre del Peer
			moRed = new NetworkManager(NetworkManager.ConfigMode.EDGE,
```

```
                                        PFC_Constantes.GRUPO_TRABAJO_NOMBRE, new File(new
File(".cache"), PFC_Constantes.NODO_NOMBRE).toURI());

                } catch (Exception e)
                {
                        moLog.error("No se puede crear la conexion con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede crear la conexion con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        throw new Exception(PFC_Constantes.ERROR_CONEXION_PEER);
                }

                // Indicador avance del proceso
                poPanel.cambiarProgreso(15);

                //Hacemos persistente el ID de Peer
                moRed.setConfigPersistent(true);


                //Gestión de la configuración
                try
                {
                        //Miro si existe configuración local. En caso de existir, obtengo la
configuración del disco. En caso de no tenerla,
                        //se crea y se guarda
                        loRedConfig = moRed.getConfigurator();

                        if (loRedConfig.exists() && !PFC_Constantes.CAMBIO_CONFIGURACION)
                        {
                                moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> con
configuración local. Se inicia proceso de carga");
                                // Se obtiene configuración local y se carga
                                loConfigLocal = new File(loRedConfig.getHome(), "PlatformConfig");

                                loRedConfig.load(loConfigLocal.toURI());
                                moLog.info("Peer<"+ PFC_Constantes.NODO_NOMBRE+"> con
configuración local. Carga realizada!");

                        }else
                        {
                                //No se dispone de configuracion local o se tiene que volver a crear
por algun cambio en configuracion
                                moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> sin
configuración local. Se crea nueva configuracion local");

                                //Se establece nombre del peer
                                loRedConfig.setName(PFC_Constantes.NODO_NOMBRE);
                                loRedConfig.setPeerId(PFC_Constantes.NODO_ID);

                                //Se indica cual sera el SuperPeer
                                loRedConfig.clearRendezvousSeeds();
                                loRedConfig.addSeedRendezvous(URI.create("tcp://" +
PFC_Constantes.SUPERPEER_IP + ":" + PFC_Constantes.SUPERPEER_PUERTO));

                                //Se graba configuracion
                                loRedConfig.save();

                                // Se establece cualquier cambio aplicado
                                moGestionBD.modificarCambio("N");
                                PFC_Constantes.CAMBIO_CONFIGURACION = false;

                                moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> sin
configuración local. Configuracion local CREADA!");
                        }

                } catch (Exception e)
                {
```

```
                        moLog.error("No se puede crear cargar la configuracion para realizar la
conexion con el Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede crear cargar la configuracion para realizar la
conexion con el Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        throw new
Exception(PFC_Constantes.ERROR_CARGA_CONFIGURACION_PEER);
                }

                // Indicador avance del proceso
                poPanel.cambiarProgreso(25);

                try
                {
                        //Se inicia red
                        loRedGrupo = moRed.startNetwork();
                        moLog.info("Conexion a la red con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADA!");

                        //Se obtiene servicio de grupo
                        moSrvPeer = loRedGrupo.getDiscoveryService();

                } catch (Exception e)
                {
                        moLog.error("No se puede realizar Start en la conexion con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede realizar Start en la conexion con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        throw new Exception(PFC_Constantes.ERROR_INI_RED_PEER);

                }

                // Indicador avance del proceso
                poPanel.cambiarProgreso(35);

                //Una vez conectados y definido configuraciones propias, esperamos conexion con el
mismo.
                //Se configura la espera a 0 para indicar que esperaremos hasta obtener conexion
                System.out.println("*************** A la espera de conectarno con
SuperPeer.........");
                moLog.info("Conectando con SuperPeer.......................");
                boolean lbConectado =
moRed.waitForRendezvousConnection(PFC_Constantes.TIMEOUT_ESPERA_SUPERPEER);

                if (lbConectado)
                {
                        System.out.println("*************** Conexion con SuperPeer
realizada!!!!");
                        moLog.info("Conexión con SuperPeer REALIZADA!");
                }
                else
                {
                        System.out.println("*************** NO se ha podido establecer conexion
con SuperPeer!!!!");
                        moLog.error("NO se ha podido establecer conexion con SuperPeer!!!!");
                        this.pararServicio();
                        throw new Exception(PFC_Constantes.ERROR_NO_CONEXION_SUPERPEER);
                }

                // Indicador avance del proceso
                poPanel.cambiarProgreso(45);

                //Gestión del grupo
                try
                {
```

```
                        moLog.info("Inicio proceso para adherirse al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");

                        //En primer lugar se verifica si existe o no grupo creado. Para eso, vemos la
cache(recursos locales)
                        loAnuncios = moSrvPeer.getLocalAdvertisements(DiscoveryService.GROUP,
"Name", PFC_Constantes.GRUPO_TRABAJO_NOMBRE);

                        if ((loAnuncios != null) && loAnuncios.hasMoreElements())
                        {
                                moLog.debug("Grupo
<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> existe en Local!");
                                //Como el grupo ya existe, se obtiene del anuncio el grupo para luego
poder unirse al grupo
                                loAdGrupo = (PeerGroupAdvertisement) loAnuncios.nextElement();
                                loTrabajoGrupo = loRedGrupo.newGroup(loAdGrupo);
                                //Se une al grupo
                                unirseAlGrupo(loTrabajoGrupo);
                                setGrupo(loTrabajoGrupo);
                                //Una vez se ha unido al grupo, publicamos esa informacion
                                moSrvGrupoPeer = loTrabajoGrupo.getDiscoveryService();
                                moSrvGrupoPeer.publish(loTrabajoGrupo.getPeerAdvertisement());

        moSrvGrupoPeer.remotePublish(loTrabajoGrupo.getPeerAdvertisement());

                        }
                        else
                        {
                                moLog.debug("Grupo
<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> NO existe en Local!");
                                //Se busca grupo de forma remota
                                moSrvPeer.getRemoteAdvertisements(null, DiscoveryService.GROUP,
"Name", PFC_Constantes.GRUPO_TRABAJO_NOMBRE, 1);

                                //Verificamos que la busqueda de Grupo se ha realizado de forma
correcta.
                                esperaMientrasGrupo(loRedGrupo);
                        }

                        moLog.info("Proceso para adherirse al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADO!");

                } catch (Exception e)
                {
                        moLog.error("No se puede adherir al Grupo con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede adherir al Grupo con el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        throw new Exception(PFC_Constantes.ERROR_NO_CONEXION_GRUPO);

                }

                // Indicador avance del proceso
                poPanel.cambiarProgreso(55);

        }//PFC_Peer

        /**
         * <p>Proceso que permite unirse al grupo.</p>
         * @param poGrupo
         */
        private void esperaMientrasGrupo(PeerGroup poGrupo) throws Exception
        {
                int liItera = 2;
                long llTiempoEspera = 30 * 1 * 1000L;
                PeerGroup loTrabajoGrupo = null;
                Enumeration<Advertisement> loAnuncios = null;
                PeerGroupAdvertisement loAdGrupo = null;
```

```
		boolean lbAdhesion = false;

		moLog.info("Se inicia proceso de Espera para Unirse al Grupo");

		//Se itera esperando a la adhesion al grupo
		for (int i=0; i < liItera; i++)
		{
			moLog.debug("Intento<"+i+"> de <"+liItera+"> intentos para unirse al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");

			moLog.debug("Duermo por :" + (llTiempoEspera/60000)+" minutos a la
espera de la adhesion al grupo");
			Thread.sleep(llTiempoEspera);
			liItera++;

			loAnuncios = moSrvPeer.getLocalAdvertisements(DiscoveryService.GROUP,
"Name", PFC_Constantes.GRUPO_TRABAJO_NOMBRE);

			if ((loAnuncios != null) && loAnuncios.hasMoreElements())
			{
				moLog.debug("Grupo
<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> existe en Local!");
				//Como el grupo ya existe, se obtiene del anuncio el grupo para luego
poder unirse al grupo
				loAdGrupo = (PeerGroupAdvertisement) loAnuncios.nextElement();
				loTrabajoGrupo = poGrupo.newGroup(loAdGrupo);
				//Se une al grupo
				unirseAlGrupo(loTrabajoGrupo);
				setGrupo(loTrabajoGrupo);
				//Una vez se ha unido al grupo, publicamos esa informacion
				moSrvGrupoPeer = loTrabajoGrupo.getDiscoveryService();
				moSrvGrupoPeer.publish(loTrabajoGrupo.getPeerAdvertisement());

	moSrvGrupoPeer.remotePublish(loTrabajoGrupo.getPeerAdvertisement());
				lbAdhesion = true;
				break;
			}//if ((loAnuncios != null) && loAnuncios.hasMoreElements())


		}//for

		if (lbAdhesion)
		{
			moLog.info("Union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> correcta!");
		}
		else
		{
			moLog.info("Union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> incorrecta!");
			throw new Exception("No se ha podido realizar la union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
		}

	}//esperaMientrasGrupo


	/**
	 * <p>Permite realizar la union a un grupo de nodos</p>
	 * @param grp
	 */
	private void unirseAlGrupo(PeerGroup grp) throws Exception
	{
		StructuredDocument creds = null;

		moLog.info("Se inicia proceso de union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
		// credenciales para el Grupo
		AuthenticationCredential authCred =	new AuthenticationCredential( grp, null,
creds );
```

```
		// MembershipService para el grupo
		MembershipService membership = grp.getMembershipService();

		// Se obtiene Authenticator para el Authentication creds
		Authenticator auth = membership.apply(authCred);

		// Si todo esta bien, se une al grupo
		if (auth.isReadyForJoin())
		{
			//Se une este peer al grupo
			Credential myCred = membership.join(auth);

			moLog.debug("Union al grupo correcta. Nombre del Grupo<" +
grp.getPeerGroupName()+"> ID del Grupo<"+grp.getPeerGroupID()+">");

			// A partir de aqui se muestran los credenciales, es solo para mostrar info por
log, no hace falta
			/*System.out.println("\nCredential: ");
			StructuredTextDocument doc = (StructuredTextDocument)
			myCred.getDocument(new MimeMediaType("text/plain"));

			StringWriter out = new StringWriter();
			doc.sendToWriter(out);
			System.out.println(out.toString());
			out.close();*/
		}
		else
		{
			moLog.info("No nos hemos podido unir al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
			throw new Exception("No nos hemos podido unir al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
		}

		moLog.info("Proceso de union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADO!");
	}//unirseAlGrupo


	/**
	 * <p>Inicia los servicios del SuperPeer</p>
	 */
	public void ejecutaServicio()
	{
		PipeAdvertisement loAdvPipeMC = null;


		//Se crea canal de comunicacion Multicast y se inicia servicio de escucha
		try
		{
			moLog.debug("Se inicia proceso de creación de canal de comunicación
Multicast");
			loAdvPipeMC =
PFC_JxtaUtilidades.generarAdvertisement(PFC_Constantes.PIPE_MULTICAST_NOMBRE,
PFC_Constantes.PIPE_MULTICAST_ID, getGrupo().getPeerGroupID(), PipeService.PropagateType);
			moSocketMC = new JxtaMulticastSocket(getGrupo(), loAdvPipeMC);
			moSocketMC.setSoTimeout(0);
			moLog.debug("Proceso de creación de canal de comunicación Multicast
REALIZADO!");

			moGestioComunicacionMC = new PFC_PGestorComunicacionMC(moSocketMC,
moGestionBD);
			Thread loThread = new Thread(moGestioComunicacionMC);
			loThread.start();

		}catch(Exception e)
		{
			moLog.error("No se puede iniciar canal de comunicacion
Multicast:"+e.getMessage());
```

```
                }


                //Se crea canal de comunicacion Unicast y se inicia servicio de escucha
                try
                {
                        moLog.debug("Se inicia proceso de creación del Pipe del canal de
comunicación Unicast");
                        moAdvPipe =
PFC_JxtaUtilidades.generarAdvertisement(PFC_Constantes.PIPE_UNICAST_NOMBRE,
PFC_Constantes.PIPE_UNICAST_ID, getGrupo().getPeerGroupID(), PipeService.UnicastType);
                        moSocket = new JxtaServerSocket(getGrupo(), moAdvPipe);

                        moSocket.setSoTimeout(0);
                        moLog.debug("Proceso de creación de canal de comunicación Unicast
REALIZADO!");

                        moGestioComunicacion = new PFC_PGestorComunicacion(moSocket,
moGestionBD);
                        Thread loThread = new Thread(moGestioComunicacion);
                        loThread.start();

                }catch(Exception e)
                {
                        moLog.error("No se puede iniciar canal de comunicacion
Unicast:"+e.getMessage());
                }


        }//ejecutaServicio

        /**
         * <p>Envia un mensaje al Grupo de Trabajo.</p>
         */
        public void enviarMensajeGrupo(PFC_Mensaje poMensaje) throws Exception
        {
                DatagramPacket loDatos = null;
                int liTam = -1;

                moLog.info("Petición de envío de mensaje al grupo desde el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> y el objeto mensaje<"+poMensaje+">");

                //Se obtiene el mensaje en el formato correcto para poder ser enviado
                byte[] byteVal = PFC_JxtaUtilidades.convertirObjectByteArray(poMensaje);

                liTam = byteVal.length;
            loDatos = new DatagramPacket(byteVal, liTam);
                moLog.debug("Se envia mensaje al grupo desde el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> con un tamaño de<"+liTam+">");
                System.out.println("*************Se envia mensaje al grupo desde el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> con un tamaño de<"+liTam+">");
                moSocketMC.send(loDatos);
                System.out.println("************ Envío de mensaje al grupo desde el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> REALIZADO");
                moLog.info("Envío de mensaje al grupo desde el
Peer<"+PFC_Constantes.NODO_NOMBRE+"> REALIZADO");

        }//enviarMensajeGrupo

        public void enviarMensaje(PFC_Mensaje poMensaje, PeerID poDestino) throws Exception
        {
                PeerID loDestino = null;
                OutputStream loSalida = null;
                InputStream loInput = null;
                DataOutput loDatosSalida = null;
                DataInput loDatosIn = null;
                byte[] loDatos = null;
                int liTamano;
                JxtaSocket loSocket = null;
                PFC_Mensaje loMensaje = null;
```

```
		try
		{
			moLog.info("Peticion de envio de mensaje Unicast al
destinatario<"+poDestino+">");
			// Se verifica el parametro poDestino ya que si su valor es null, significa que el
destinatario es el SuperPeer
			loDestino = poDestino;
			if (poDestino == null)
			{
				moLog.info("Destinatario es SuperPeer");
				loDestino =  (PeerID)getURISuperPeer(1);
			}


			// Se abre socket de conexion
			moLog.debug("Se inicia apertura de Socket para enviar mensaje a
destino<"+loDestino+">");

			loSocket = new JxtaSocket(getGrupo(),
				//Indica el Peer destinatario, null si no especifico
				loDestino,
				moAdvPipe,
				//Timeout de conexion
				PFC_Constantes.TIMEOUT_SOCKET_UNICAST,
				// Conexion segura (confianza)
				true);
			moLog.debug("Socket ABIERTO!");

			// Flujo de salida de datos
			loSalida = loSocket.getOutputStream();
			loDatosSalida = new DataOutputStream(loSalida);

			loDatos = PFC_JxtaUtilidades.convertirObjectByteArray(poMensaje);

			liTamano = loDatos.length;

			moLog.info("Se va a enviar mensaje con Tamnño<"+liTamano+">");
			System.out.println("Se va a enviar mensaje con Tamnño<"+liTamano+">");

			loDatosSalida.writeInt(liTamano);
			loSalida.write(loDatos);
			loSalida.flush();
			loSalida.close();
			moLog.info("Mensaje con Tamaño<"+liTamano+"> ENVIADO");
			System.out.println("Mensaje con Tamaño<"+liTamano+"> ENVIADO");

			// Se obtiene mensaje
			loInput = loSocket.getInputStream();
			loDatosIn = new DataInputStream(loInput);
			moLog.debug("Canales de entrada abiertos..");
			//Obtenemos tamanyo del mensaje
			liTamano = loDatosIn.readInt();
			moLog.debug("Mensaje recibido de Tamanyo<"+liTamano+">");
			//Se obtiene el objeto mensaje enviado
			loDatos = new byte[liTamano];
			loDatosIn.readFully(loDatos);
			loMensaje = (PFC_Mensaje)
PFC_JxtaUtilidades.convertirByteArrayObject(loDatos);
			moLog.info("Mensaje recibido de<"+loMensaje.getPeerOrigen()+"> y
Tipo<"+loMensaje.getTipoMensaje()+">");
			System.out.println("**********Mensaje recibido
de<"+loMensaje.getPeerOrigen()+"> y Tipo<"+loMensaje.getTipoMensaje()+">");
			loInput.close();
			loSocket.close();

			// Se inicia tratamiento mensaje respuesta
			tratarRespuesta(loMensaje);

		} catch (Exception e)
```

```
		{
			moLog.error("Problemas al enviar mensaje Unicast:"+e);
			throw new Exception("Problemas al enviar mensaje Unicast.");
		}

	}//enviaMensaje

	private void tratarRespuesta(PFC_Mensaje poMensaje) throws Exception
	{
		// Informacion de Altas recibidas.
		Vector loAltas = null;


		moLog.info("Peticion para tratar mensaje tipo<"+poMensaje.getTipoMensaje()+">");
		// Se inicializa medidor del rendimiento
		PFC_Rendimiento loRendimiento = new
PFC_Rendimiento("(Unicast)"+poMensaje.getTipoMensaje());

		// En funcion de la tipologia del mensaje se hace una u otra cosa
		if (poMensaje.getTipoMensaje().equals(PFC_Mensaje.TIPO_ACTUALIZACION))
		{
			moLog.info("Se inicia proceso de actualizacion de documentos. Numero de
actualizaciones a realizar<"+(poMensaje.getActualizaciones() == null ? 0 :
poMensaje.getActualizaciones().size())+">");
			System.out.println("***********Se inicia proceso de actualizacion de
documentos. Numero de actualizaciones a realizar<"+(poMensaje.getActualizaciones() == null ? 0 :
poMensaje.getActualizaciones().size())+">");

			// Se recorren todas las actualizaciones y se van haciendo las adaptaciones
indicadas
			Vector loActualizaciones = poMensaje.getActualizaciones();
			if ( (loActualizaciones != null) && (loActualizaciones.size() > 0) )
			{

				for (int i=0; i < loActualizaciones.size(); i++)
				{
					if (
((PFC_Mensaje)loActualizaciones.elementAt(i)).getTipoMensaje().equals(PFC_Mensaje.TIPO_BAJA))
					{

	PFC_PeerComun.gestionarBajaDocumento((PFC_Mensaje)loActualizaciones.elementAt(i) ,
moGestionBD);
					}
					if (
((PFC_Mensaje)loActualizaciones.elementAt(i)).getTipoMensaje().equals(PFC_Mensaje.TIPO_AVISO_ALT
A))
					{
						// Se obtiene la informacion del nuevo documento.
Id documento y propietario
						String[] lvsDatos = new String[2];
						lvsDatos[0] =
((PFC_Mensaje)loActualizaciones.elementAt(i)).getIdDocumento();
						lvsDatos[1] =
((PFC_Mensaje)loActualizaciones.elementAt(i)).getPeerOrigen();
						moLog.info("Detectado nueva alta del documento
con id<"+lvsDatos[0]+"> del propietario<"+lvsDatos[1]+">");
						//Se añade al vector de altas
						if (loAltas == null)
							loAltas = new Vector();

						loAltas.add(lvsDatos);
					}
					if (
((PFC_Mensaje)loActualizaciones.elementAt(i)).getTipoMensaje().equals(PFC_Mensaje.TIPO_CAMBIO))
					{

	PFC_PeerComun.gestionarModificacionDocumento((PFC_Mensaje)loActualizaciones.elementAt(i)
, moGestionBD);
					}
				}//for (int i=0; i < loActualizaciones.size(); i++)
```

```
                                        // Una vez se han procesado todas las actualizaciones que el
Superpeer ha informado, se ejecutan aquellas que hayan podido ser de alta
                                        moLog.info("Numero de altas a procesar<"+(loAltas != null ?
loAltas.size() : 0)+">");

                                        // Se verifica y realizan las altas que hayan podido ser informadas
                                        realizarAltas(loAltas);

                                }//if (loActualizaciones != null)
                        }//if (poMensaje.getTipoMensaje().equals(PFC_Mensaje.TIPO_ACTUALIZACION))

                        if (poMensaje.getTipoMensaje().equals(PFC_Mensaje.TIPO_RESPUESTA_ALTA))
                        {
                                moLog.info("Se inicia proceso de alta de un nuevo documento. ");

                                PFC_PeerComun.gestionarAltaDocumento(poMensaje, moGestionBD);
                                moLog.info("Proceso de alta de un nuevo documento realizado!");
                        }//if (poMensaje.getTipoMensaje().equals(PFC_Mensaje.TIPO_RESPUESTA_ALTA))

                        //Calculo del tiempo utilizado en dar de alta un documento.
                        loRendimiento.tiempo();


                        moLog.info("Peticion para tratar mensaje tipo<"+poMensaje.getTipoMensaje()+">
FINALIZADA!");

                }//tratarRespuesta

                /**
                 * <p>Trata todas las altas identificadas por el SuperPeer. Se envia mensaje a Peer creador del
documento para que nos lo de.
                 * En caso de no estar disponible el peer, se pide el documento al Data Center
(SuperPeer)</p>
                 * @param poAltas
                 * @throws Exception
                 */
                private void realizarAltas(Vector poAltas) throws Exception
                {

                        moLog.info("Peticion para tratar altas de documentos. Numero de altas
informadas<"+(poAltas != null ? poAltas.size() : 0)+">");

                        if (poAltas != null)
                        {
                                // Para cada una de las altas indicadas se tiene que realizar una peticion al
propietario del documento y este nos
                                // tiene que enviar el documento
                                for (int i = 0; i < poAltas.size(); i++)
                                {
                                        // Se obntiene la informacion del documento a pedir
                                        String[] lvsDoc = (String[])poAltas.elementAt(i);
                                        // Se construye un mensaje del tipo DAME_ALTA donde le
indicaremos el iddocumento y sabremos el propietario al que enviarlo
                                        PFC_Mensaje loMensaje = new
PFC_Mensaje(PFC_Mensaje.TIPO_PETICION_ALTA);
                                        // Se establece el iddocumento
                                        loMensaje.setIdDocumento(lvsDoc[0]);
                                        // Establecemos el id del peer que envia el mensaje
                                        loMensaje.setPeerOrigen(PFC_Constantes.NODO_ID);

                                        // Se hace la peticion de envio de mensaje
                                        try
                                        {
                                                this.enviarMensaje(loMensaje,
(PeerID)IDFactory.fromURI(new URI(lvsDoc[1])));
                                        }catch(Exception e)
                                        {
```

```
                                        // Al dar un error al realizar la peticion del documento al
Peer creador, se realiza la peticion
                                        // del documento al Data Center (SuperPeer)
                                        this.enviarMensaje(loMensaje, null);
                                }

                        }//for (int i = 0; i < poAltas.size(); i++)
                }

                moLog.info("Peticion para tratar altas de documentos REALIZADA");
        }//realizarAltas

        /**
         * <p>Obtiene el identificador unico del SuperPeer indicado</p>
         * <p>Esta preparado para existir mas de un SuperPeer por lo que la variable de entrada indica
la posicion del SuperPeer que queremos</p>
         * @param piPosicionSuperPeer Posicion del SuperPeer que queremos. Si p.ej. = 1, de la
posible lista de SuperPeer, obtiene la primera
         * @return
         */
        private ID getURISuperPeer(int piPosicionSuperPeer) throws Exception
        {
                //Identificador unico del SuperPeer seleccionado
                String lsURI = "";
                //Listado de SuperPeers disponibles
                Enumeration<ID> loSuperPeers = null;
                //Indice que nos indica porque posicion de la lista vamos
                int liIndice = 1;
                ID loSP = null;

                //Obtenemos los SuperPeer disponibles.
                loSuperPeers =
moRed.getNetPeerGroup().getRendezVousService().getConnectedRendezVous();

                //En caso de no haber SuperPeer, error y abortamos. Sin SuperPeer no esta preparado
para trabajar
                if (loSuperPeers == null)
                {
                        System.out.println("Error - No disponemos de ningun SuperPeer.");
                        System.exit(-1);
                }

                //Recorremos SuperPeers disponibles
                while (loSuperPeers.hasMoreElements())
                {
                        //Obtenemos SuperPeer
                        loSP = loSuperPeers.nextElement();

                        if (liIndice == piPosicionSuperPeer)
                        {
                                lsURI =  loSP.toURI().toString();

                                break;
                        }
                }

                if (lsURI.equals(""))
                {
                        throw new Exception("No se puede obtener el identificador de SuperPeer");
                }

                return loSP;

        }//getURISuperPeer

        /**
         * <p>Establece Grupo de Trabajo</p>
         * @param poGrupo
         */
        private void setGrupo(PeerGroup poGrupo)
```

```
	{
		moGrupo = poGrupo;
	}//setGrupo

	/**
	 * <p>Obtiene grupo de trabajo</p>
	 * @return
	 */
	public PeerGroup getGrupo()
	{
		return moGrupo;
	}//getGrupo

	/**
	 * <p>Para el servicio de red</p>
	 * @throws Throwable
	 */
	public void pararServicio()
	{
		moLog.info("Peticion para parar conexion del
Peer<"+PFC_Constantes.NODO_NOMBRE+"> del
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");


		if (moGrupo != null)
		{
			moLog.info("Se para Grupo.");
			moGrupo.stopApp();
			moGrupo.unref();
			moGrupo = null;
			moLog.info("Grupo PARADO!");
		}

		if (moGestioComunicacionMC != null)
		{
			moLog.info("Se para Thread comunicacion Multicast");
			moGestioComunicacionMC.terminarProceso();
			moGestioComunicacionMC = null;
			moLog.info("Thread comunicacion Multicast PARADO!");
		}
		if (moGestioComunicacion != null)
		{
			moLog.info("Se para Thread comunicacion Uniicast");
			moGestioComunicacion.terminarProceso();
			moGestioComunicacion = null;
			moLog.info("Thread comunicacion Unicast PARADO!");
		}
		if (moSocket != null)
		{
			try
			{
				moLog.info("Se cierra Socket Unicast.");
				moSocket.close();
				moLog.info("Socket Unicast CERRADO.");
			} catch (IOException e)
			{
				moLog.error("Problemas al cerrar el Socket Unicast.", e);
			}
			moSocket = null;
		}
		if (moSocketMC != null)
		{
			moLog.info("Se cierra Socket Multicast.");
			moSocketMC.close();
			moLog.info("Socket Multicast CERRADO.");
			moSocketMC = null;
		}
		if (moRed != null)
		{
			moLog.info("Se cierra NetworkManager");
```

```
                    moRed.stopNetwork();
                    moRed = null;
                    System.gc();
                    System.runFinalization();
                    System.gc();
                    moLog.info("NetworkManager CERRADO!");

            }


            moLog.info("Conexion del Peer<"+PFC_Constantes.NODO_NOMBRE+"> del
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> PARADA!");
        }//stopServicio

}
```

9.2.4 SUPERPEER

```
package pfc.prototipo.jxta.superpeer;

import java.io.DataInput;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.StringWriter;
import java.net.Socket;
import java.net.SocketAddress;
import java.net.URI;
import java.net.URISyntaxException;
import java.text.MessageFormat;
import java.util.Date;
import java.util.Enumeration;
import java.util.StringTokenizer;

import javax.security.cert.CertificateException;

import org.apache.log4j.Logger;
import org.apache.log4j.PropertyConfigurator;

import pfc.prototipo.PFC_Constantes;
import pfc.prototipo.bbdd.PFC_BDGestion;
import pfc.prototipo.jxta.PFC_JxtaUtilidades;
import pfc.prototipo.jxta.peer.PFC_PGestorComunicacion;
import pfc.prototipo.jxta.peer.PFC_PGestorComunicacionMC;

import net.jxta.credential.AuthenticationCredential;
import net.jxta.credential.Credential;
import net.jxta.discovery.DiscoveryEvent;
import net.jxta.discovery.DiscoveryListener;
import net.jxta.discovery.DiscoveryService;
import net.jxta.document.Advertisement;
import net.jxta.document.AdvertisementFactory;
import net.jxta.document.MimeMediaType;
import net.jxta.document.StructuredDocument;
import net.jxta.document.StructuredTextDocument;
import net.jxta.endpoint.Message;
import net.jxta.endpoint.StringMessageElement;
import net.jxta.exception.PeerGroupException;
import net.jxta.id.ID;
import net.jxta.id.IDFactory;
```

```
import net.jxta.membership.Authenticator;
import net.jxta.membership.MembershipService;
import net.jxta.peer.PeerID;
import net.jxta.peergroup.PeerGroup;
import net.jxta.peergroup.PeerGroupID;
import net.jxta.pipe.InputPipe;
import net.jxta.pipe.OutputPipe;
import net.jxta.pipe.OutputPipeEvent;
import net.jxta.pipe.OutputPipeListener;
import net.jxta.pipe.PipeMsgEvent;
import net.jxta.pipe.PipeMsgListener;
import net.jxta.pipe.PipeService;
import net.jxta.platform.NetworkConfigurator;
import net.jxta.platform.NetworkManager;
import net.jxta.protocol.ModuleImplAdvertisement;
import net.jxta.protocol.PeerGroupAdvertisement;
import net.jxta.protocol.PipeAdvertisement;
import net.jxta.rendezvous.RendezVousService;
import net.jxta.rendezvous.RendezvousEvent;
import net.jxta.rendezvous.RendezvousListener;
import net.jxta.socket.JxtaMulticastSocket;
import net.jxta.socket.JxtaServerSocket;

public class PFC_SuperPeer
{
	//Trazas
	private final static Logger moLog = Logger.getLogger(PFC_SuperPeer.class);

	//Red
	private NetworkManager moRed = null;

	//Servicio de SuperPeer
	private DiscoveryService moSrvSuperPeer = null;
	//Servicio de Grupo
	private DiscoveryService moSrvGrupoPeer = null;
	//Grupo de Peers
	private PeerGroup moGrupo = null;

	//Gestion de la base de datos
	private PFC_BDGestion moGestionBD = null;

	// Canal de comunicacion Multicast
	private JxtaMulticastSocket moSocketMC = null;

	// Gestiona mensajes de entrada canal Multicast
	private PFC_SPGestorComunicacionMC moGestioComunicacionMC = null;

	// Gestiona mensajes de entrada canal Multicast
	private PFC_SPGestorComunicacion moGestioComunicacion = null;

	// Canal de comunicaion Unicast
	private JxtaServerSocket moSocket = null;

	/**
	 * <p>Inicializa configuracion y crea conexion al grupo</p>
	 */
	public PFC_SuperPeer()
	{
		NetworkConfigurator loRedConfig;
		File loConfigLocal = null;
		PeerGroup loRedGrupo = null;
		PeerGroup loTrabajoGrupo = null;
		PeerGroupAdvertisement loAdGrupo = null;
		Enumeration<Advertisement> loAnuncios = null;

		//Indica si existe configuracion
		boolean lbExisteConfig = false;

		try
		{
```

```
//Se crea conexion con base de datos.
moGestionBD = new PFC_BDGestion();

//Se verifica si existe una configuracion creada. En caso de no existir, se
genera de nuevo.
//Si no existe es porque la aplicacion se utiliza por primera vez y es cuando se
tiene que generar los identificadores unicos
//del superpeer y de los canales de comunicacion necesarios
lbExisteConfig =
moGestionBD.existeTabla(PFC_Constantes.TABLA_CONFIGURACION);

if (lbExisteConfig)
{
	moLog.info("Se carga configuracion.");
	PFC_Constantes.cargarConfiguracion(moGestionBD);
}
else
{
	moLog.info("Aplicacion en proceso de inicializacion. Se carga
configuracion por primera vez");
	//La primera vez que se inicia, se establece que el directorio es el
path actual mas el establecido por defecto
	PFC_Constantes.DIRECTORIO = new File(".").getCanonicalPath() +
PFC_Constantes.DIRECTORIO;
	moLog.info("Se establece el directorio de trabajo
inicial<"+PFC_Constantes.DIRECTORIO+">");
	PFC_Constantes.crearConfiguracion(moGestionBD, true);
}

}catch(Exception e)
{
	moLog.error("No se ha podido iniciar la solucion:"+e.getLocalizedMessage());
	System.out.println("No se ha podido iniciar la solucion");
	if (moGestionBD != null)
	{
		moGestionBD.cerrarConexion();
	}
	System.exit(-1);
}

// Inicializacion de conexion a la red

try
{
	moLog.info("Se inicia conexion a la red con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
	//Iniciamos conexion a la red que se generara. En primer lugar se indica el
tipo de SuperPeer que se quiere. En segundo el nombre
	//de la red que se crea, en tercer lugar se genera un fichero de configuracion
con el nombre del SuperPeer
	moRed = new
NetworkManager(NetworkManager.ConfigMode.RENDEZVOUS_RELAY,
		PFC_Constantes.GRUPO_TRABAJO_NOMBRE, new File(new
File(".cache"), PFC_Constantes.NODO_NOMBRE).toURI());

} catch (Exception e)
{
	moLog.error("No se puede crear la conexion con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
	System.out.println("No se puede crear la conexion con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
	System.exit(-1);
}

//Hacemos persistente el ID de Peer
moRed.setConfigPersistent(true);
```

```
//Gestión de la configuración
try
{
	//Miro si existe configuración local. En caso de existir, obtengo la
configuración del disco. En caso de no tenerla,
	//se crea y se guarda
	loRedConfig = moRed.getConfigurator();

	if (loRedConfig.exists())
	{
		moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> con
configuración local. Se inicia proceso de carga");
		// Se obtiene configuración local y se carga
		loConfigLocal = new File(loRedConfig.getHome(), "PlatformConfig");

		loRedConfig.load(loConfigLocal.toURI());
		moLog.info("Peer<"+ PFC_Constantes.NODO_NOMBRE+"> con
configuración local. Carga realizada!");

	}else
	{
		//No se dispone de configuracion local, se crea una nueva y se guarda

		moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> sin
configuración local. Se crea nueva configuracion local");
		//Se establece nombre del peer
		loRedConfig.setName(PFC_Constantes.NODO_NOMBRE);
		loRedConfig.setPeerId(PFC_Constantes.NODO_ID);
		System.out.println("*********** Se establece
puerto<"+PFC_Constantes.SUPERPEER_PUERTO+">");
		loRedConfig.setTcpPort(new
Integer(PFC_Constantes.SUPERPEER_PUERTO).intValue());
		loRedConfig.setTcpEnabled(true);
		loRedConfig.setUseMulticast(true);

		//Se graba configuracion
		loRedConfig.save();
		moLog.info("Peer<"+PFC_Constantes.NODO_NOMBRE+"> sin
configuración local. Configuracion local CREADA!");
	}


} catch (Exception e)
{
	moLog.error("No se puede crear cargar la configuracion para realizar la
conexion con el SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
	System.out.println("No se puede crear cargar la configuracion para realizar la
conexion con el SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
	System.exit(-1);
}


//Inicio de la Red (comunicación)
try
{
	//Inicio de las comunicaciones
	loRedGrupo = moRed.startNetwork();
	moLog.info("Conexion a la red con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADA!");

	//Se obtiene servicio de grupo
	moSrvSuperPeer = loRedGrupo.getDiscoveryService();

} catch (Exception e)
{
```

```
                        moLog.error("No se puede realizar Start en la conexion con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede realizar Start en la conexion con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        System.exit(-1);
                }


                //Gestión del grupo
                try
                {
                        moLog.info("Inicio proceso para adherirse al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");

                        //En primer lugar se verifica si existe o no grupo creado. Para eso, vemos la
cache(recursos locales)
                        loAnuncios =
moSrvSuperPeer.getLocalAdvertisements(DiscoveryService.GROUP, "Name",
PFC_Constantes.GRUPO_TRABAJO_NOMBRE);

                        if ((loAnuncios != null) && loAnuncios.hasMoreElements())
                        {
                                moLog.debug("Grupo
<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> existe en Local!");
                                //Como el grupo ya existe, se obtiene del anuncio el grupo para luego
poder unirse al grupo
                                loAdGrupo = (PeerGroupAdvertisement) loAnuncios.nextElement();
                                loTrabajoGrupo = loRedGrupo.newGroup(loAdGrupo);
                                //Se une al grupo
                                unirseAlGrupo(loTrabajoGrupo);
                                setGrupo(loTrabajoGrupo);
                        }
                        else
                        {
                                moLog.debug("Grupo
<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> NO existe en Local!");
                                //Se crea el grupo
                                loTrabajoGrupo = generarGrupo(loRedGrupo);

                                //Se publica el SuperPeer
                                loTrabajoGrupo.getRendezVousService().startRendezVous();
                                //Se une al grupo
                                unirseAlGrupo(loTrabajoGrupo);
                                setGrupo(loTrabajoGrupo);
                        }

                        //Una vez se ha unido al grupo, publicamos esa informacion
                        moSrvGrupoPeer = loTrabajoGrupo.getDiscoveryService();
                        moSrvGrupoPeer.publish(loTrabajoGrupo.getPeerAdvertisement());
                        moSrvGrupoPeer.remotePublish(loTrabajoGrupo.getPeerAdvertisement());

                        moLog.info("Proceso para adherirse al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADO!");

                } catch (Exception e)
                {
                        moLog.error("No se puede adherir al Grupo con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">:"+e.getMessage());
                        System.out.println("No se puede adherir al Grupo con el
SuperPeer<"+PFC_Constantes.NODO_NOMBRE+"> y
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                        System.exit(-1);
                }

        }//PFC_SuperPeer

        /**
```

```
	 * <p>Inicio ejecucion del SuperPeer</p>
	 * @param args
	 */
	public static void main(String args[])
	{
		int liPuerto = -1;

		//Lo primero que se realiza es inicialiar el sistema de trazas. Se carga fichero de
propiedades donde se indica toda la gestión del sistema de trazas
		try
		{
			//Se carga configuracion de trazas
			PropertyConfigurator.configure("log4j.properties");

		}catch(Exception e)
		{
			System.out.println("ERROR: Problemas al inicializar el sistema de trazas.
Verifique su instalación para poder utilizar la solución.");
			System.exit(-1);
		}

		//Si se ha introducido el parametro de entrada, se verifica que sea un entero
		if ( (args != null) && (args.length != 0) )
		{
			//Se verifica que el parametro introducido sea un numerico
			try
			{
				liPuerto =  new Integer(args[0]).intValue();
			}catch(Exception e)
			{
				//No se trata de un numero. Parametro introducido pero formato
erroneo
				moLog.error("Formato del parámetro erróneo. Debe ser un numérico
entero.");
			}
		}


		// Si se ha introducido un numero de puerto numerico se establece como puerto de
trabajo
		if (liPuerto != -1)
		{
			moLog.info("Se establece como puerto de trabajo<"+liPuerto+">");
			System.out.println("************Se establece como puerto de
trabajo<"+liPuerto+">");
			PFC_Constantes.SUPERPEER_PUERTO = ""+liPuerto;
		}

		// Se establece nombre del SuperPeer
		PFC_Constantes.NODO_NOMBRE = PFC_Constantes.SUPERPEER_NOMBRE_PREFIJO;
		moLog.debug("Se inicia el proceso de SuperPeer con nombre
asignado<"+PFC_Constantes.NODO_NOMBRE+">");


		PFC_SuperPeer loSuper = new PFC_SuperPeer();
		loSuper.ejecutaServicio();

	}//main


	/**
	 * <p>Inicia los servicios del SuperPeer</p>
	 */
	private void ejecutaServicio()
	{
		PipeAdvertisement loAdvPipeMC = null;
		PipeAdvertisement loAdvPipe = null;

		moLog.info("Se inicia Servicio del SuperPeer con
nombre<"+PFC_Constantes.NODO_NOMBRE+">");
```

```
                try
                {
                        moLog.debug("Se inicia proceso de creación de canal de comunicación
Multicast");
                        loAdvPipeMC =
PFC_JxtaUtilidades.generarAdvertisement(PFC_Constantes.PIPE_MULTICAST_NOMBRE,
PFC_Constantes.PIPE_MULTICAST_ID, getGrupo().getPeerGroupID(), PipeService.PropagateType);
                        moSocketMC = new JxtaMulticastSocket(getGrupo(), loAdvPipeMC);
                        moSocketMC.setSoTimeout(0);
                        moLog.debug("Proceso de creación de canal de comunicación Multicast
REALIZADO!");

                        moGestioComunicacionMC = new
PFC_SPGestorComunicacionMC(moSocketMC, moGestionBD);
                        Thread loThread = new Thread(moGestioComunicacionMC);
                        loThread.start();

                }catch(Exception e)
                {
                        moLog.error("No se puede iniciar canal de comunicacion
Multicast:"+e.getMessage());
                }


                //Se crea canal de comunicacion Unicast y se inicia servicio de escucha
                try
                {
                        moLog.debug("Se inicia proceso de creación del Pipe del canal de
comunicación Unicast");
                        loAdvPipe =
PFC_JxtaUtilidades.generarAdvertisement(PFC_Constantes.PIPE_UNICAST_NOMBRE,
PFC_Constantes.PIPE_UNICAST_ID, getGrupo().getPeerGroupID(), PipeService.UnicastType);
                        moSocket = new JxtaServerSocket(getGrupo(), loAdvPipe);
                        moSocket.setSoTimeout(0);
                        moLog.debug("Proceso de creación de canal de comunicación Unicast
REALIZADO!");

                        moGestioComunicacion = new PFC_SPGestorComunicacion(moSocket,
moGestionBD);
                        Thread loThread = new Thread(moGestioComunicacion);
                        loThread.start();

                }catch(Exception e)
                {
                        moLog.error("No se puede iniciar canal de comunicacion
Unicast:"+e.getMessage());
                }


        }//ejecutaServicio


        /**
         * <p>Crea Grupo de Trabajo</p>
         * @param poGrupo
         * @return
         * @throws Exception
         */
        private PeerGroup generarGrupo(PeerGroup poGrupo) throws Exception
        {
                ModuleImplAdvertisement loIAGrupo = null;
                PeerGroup loGrupo = null;
                PeerGroupAdvertisement loAnuncioGrupo = null;

                moLog.info("Se inicia proceso de creación del Grupo de
trabajo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
                loIAGrupo = poGrupo.getAllPurposePeerGroupImplAdvertisement();
```

```
		loGrupo = poGrupo.newGroup(null, loIAGrupo,
PFC_Constantes.GRUPO_TRABAJO_NOMBRE,PFC_Constantes.GRUPO_TRABAJO_DESC);
		loAnuncioGrupo = loGrupo.getPeerGroupAdvertisement();
		// Se publica anuncio de grupo
		moSrvSuperPeer.publish(loAnuncioGrupo);
		moSrvSuperPeer.remotePublish(loAnuncioGrupo);

		moLog.info("Creación del Grupo de
trabajo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADA!");

		return loGrupo;
	}//generarGrupo

	/**
	 * <p>Permite realizar la union a un grupo de nodos</p>
	 * @param grp
	 */
	private void unirseAlGrupo(PeerGroup grp) throws Exception
	{
		StructuredDocument creds = null;

		moLog.info("Se inicia proceso de union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
		// credenciales para el Grupo
		AuthenticationCredential authCred =		new AuthenticationCredential( grp, null,
creds );

		// MembershipService para el grupo
		MembershipService membership = grp.getMembershipService();

		// Se obtiene Authenticator para el Authentication creds
		Authenticator auth = membership.apply(authCred);

		// Si todo esta bien, se une al grupo
		if (auth.isReadyForJoin())
		{
			//Se une este peer al grupo
			Credential myCred = membership.join(auth);

			moLog.debug("Union al grupo correcta. Nombre del Grupo<" +
grp.getPeerGroupName()+"> ID del Grupo<"+grp.getPeerGroupID()+">");

			// A partir de aqui se muestran los credenciales, es solo para mostrar info por
log, no hace falta
			/*System.out.println("\nCredential: ");
			StructuredTextDocument doc = (StructuredTextDocument)
			myCred.getDocument(new MimeMediaType("text/plain"));

			StringWriter out = new StringWriter();
			doc.sendToWriter(out);
			System.out.println(out.toString());
			out.close();*/
		}
		else
		{
			moLog.info("No nos hemos podido unir al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
			throw new Exception("No nos hemos podido unir al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+">");
		}

		moLog.info("Proceso de union al
Grupo<"+PFC_Constantes.GRUPO_TRABAJO_NOMBRE+"> REALIZADO!");
	}//unirseAlGrupo

	/**
	 * <p>Establece Grupo de Trabajo</p>
	 * @param poGrupo
	 */
	private void setGrupo(PeerGroup poGrupo)
```

```
        {
                moGrupo = poGrupo;
        }//setGrupo

        /**
         * <p>Obtiene grupo de trabajo</p>
         * @return
         */
        public PeerGroup getGrupo()
        {
                return moGrupo;
        }//getGrupo
}
```

10 BIBLIOGRAFIA

10.1 ESTUDIO DE LA BASE DE DATOS

POSTGRESQL

http://www.dataprix.com/12-prestaciones

http://es.wikipedia.org/wiki/PostgreSQL

http://www.siadsoft.com/index.php?option=com_content&view=article&id=55:postgresql-en-usabilidad-web-postgresql-y-java-postgresql-y-php&catid=34:gossips

http://www.monografias.com/trabajos29/comparacion-sistemas/comparacion-sistemas.shtml

JAVADB

http://es.wikipedia.org/wiki/Apache_Derby

http://axiacore.com/2008/04/java-db-ultima-version-lanzada/

http://www.oracle.com/technetwork/java/javadb/overview/index.html

ORACLE

http://www.rhernando.net/modules/tutorials/doc/bd/oracle.html

http://es.answers.yahoo.com/question/index?qid=20101029073830AAYvbYr

http://www.oracle.es

10.2 ESTUDIO TÉCNICO

Wikipedia

JXSE_ProgGuide_v2.5

http://java.net/projects/jxta

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

http://www.jdom.org/downloads/index.html

http://logging.apache.org/log4j/

10.3 OTROS

Preparación presentación

- Imágenes en Google.