Universidad Zaragoza

1542

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

MC-SPY

UN SISTEMA PARA LA EXPLOTACIÓN DE LA INFORMACIÓN HISTÓRICA DE OPERACIONES REALIZADAS EN BANCA ELECTRÓNICA

Santiago Abadía Zárate

Director: Javier Campos Laclaustra
Departamento de Informática en Ingeniería de Sistemas

Escuela de Ingeniería y Arquitectura
2014

Resumen

MC-Spy básicamente es un módulo de una aplicación web, que por un lado: maneja, almacena y gestiona una serie de datos estadísticos obtenidos por él mismo de dicha aplicación web; y por otro lado utiliza todos esos datos almacenados para generar estadísticas y previsiones en formato de texto y gráfico.

MC-Spy surgió inicialmente, y de hecho se desarrolló, bajo el contexto de su integración en una banca electrónica, pero puede ser totalmente integrado en cualquier otra aplicación web (Java), como de hecho se hizo posteriormente por parte de la empresa en la que se realizó el proyecto.

Gracias a MC-Spy, los administradores de la banca electrónica (o aplicación web en general), podrán detectar al instante posibles errores de funcionamiento de la banca, podrán obtener estadísticas con la que los directores bancarios poder hacer sus decisiones, e incluso mostrar previsiones de entrada de dinero, operaciones que se prevé que se realizarán en la banca en los próximos meses (útil por ejemplo para anticipar un aumento de prestaciones de los servidores) ,etc… gracias a unos algoritmos de previsiones en base a las series de datos almacenados que el mismo MC-Spy ha estado contabilizando.

Agradecimientos

Por supuesto, agradecer su preocupación por el futuro de este proyecto, a Javier Campos, verdadero forjador de su (buen) fin.

Recordar igualmente a todos mis compañeros del CPS.

Reconocer también la preocupación de todos los amigos que de vez en cuando me preguntaban por este proyecto fin de carrera.

Gracias a Eva, por todo lo que me has dado sin pedir nada a cambio.

Por último y más importante, agradecer a mi familia el apoyo prestado, especialmente a mis padres, ejemplo en mi vida y artífices de mi educación. Todo lo conseguido por mí, es gracias a vosotros.

Índice

1 Introducción

MC-Spy surgió en un principio como un módulo que se debería de integrar dentro de un sistema de banca electrónica multicanal (MultiChannel Server, a partir de ahora MCServer), que se estaba desarrollando en la empresa TB-Solutions (a partir de ahora TBS). A través de MC-Server, los clientes de un banco, iban a poder realizar diversas operaciones bancarias desde su casa o lugar de trabajo mediante una conexión a Internet.

El número de las operativas disponibles en MC-Server era relativamente grande, y el grado de diversidad entre ellas, alto.

Estaba ya funcionando el sistema (al menos en una fase inicial), cuando se vio la necesidad, tanto por parte del cliente como de TBS, de conocer aquellas operativas que eran más utilizadas por los usuarios finales de la aplicación, al igual que aquellas que eran menos usadas, más desconocidas o peor utilizadas por los clientes. Así mismo, también era interesante conocer el tiempo que los usuarios de la aplicación tardaban en completar una operativa, y otros datos interesantes relativos al uso de dicha banca electrónica.

Para ello, era necesario desarrollar un módulo, ya sea dentro o fuera de la propia aplicación, que obtuviera de algún modo toda esta información requerida, y mostrara a los clientes (no a los usuarios finales, sino a los administradores de la aplicación y a los ejecutivos de la banca) estos datos de una forma gráfica, para su posterior estudio y tomar las medidas oportunas en función de los resultados que se obtuvieran.

Con estas necesidades iniciales, se propuso realizar MC-Spy como proyecto fin de carrera.

1.1 MC-Server

Pasemos ahora, en primer lugar, a explicar más detalladamente qué es MC-Server y como está estructurado, para entender cómo el desarrollo de MC-Spy ha estado condicionado inicialmente por dichas circunstancias.

MC-Server se puede definir, tal y como ya se ha dicho anteriormente, como un sistema de banca electrónica multicanal. Sin embargo, MC-Server es algo más que eso. MC-Server es un framework de trabajo para la implementación de operativas online, orientado a la banca virtual, pero adaptable a otros contextos y modelos de negocio. MC-Server ofrece una serie de servicios y facilidades para el desarrollo rápido de operativas, garantizando el rendimiento, escalabilidad y flexibilidad de la solución.

Es decir, MC-Server no se limita a ser únicamente el núcleo de una banca electrónica sobre la que implementar operativas, ya que lo que se pretende es adaptarse a cualquier otra solución; no obstante si que es verdad que cuando se propuso hacer este proyecto

fin de carrera, MC-Server se estaba desarrollando para ponerlo en funcionamiento como una banca electrónica para un banco andorrano.

El siguiente gráfico muestra un resumen de los servicios que puede prestar al usuario MC-Server:

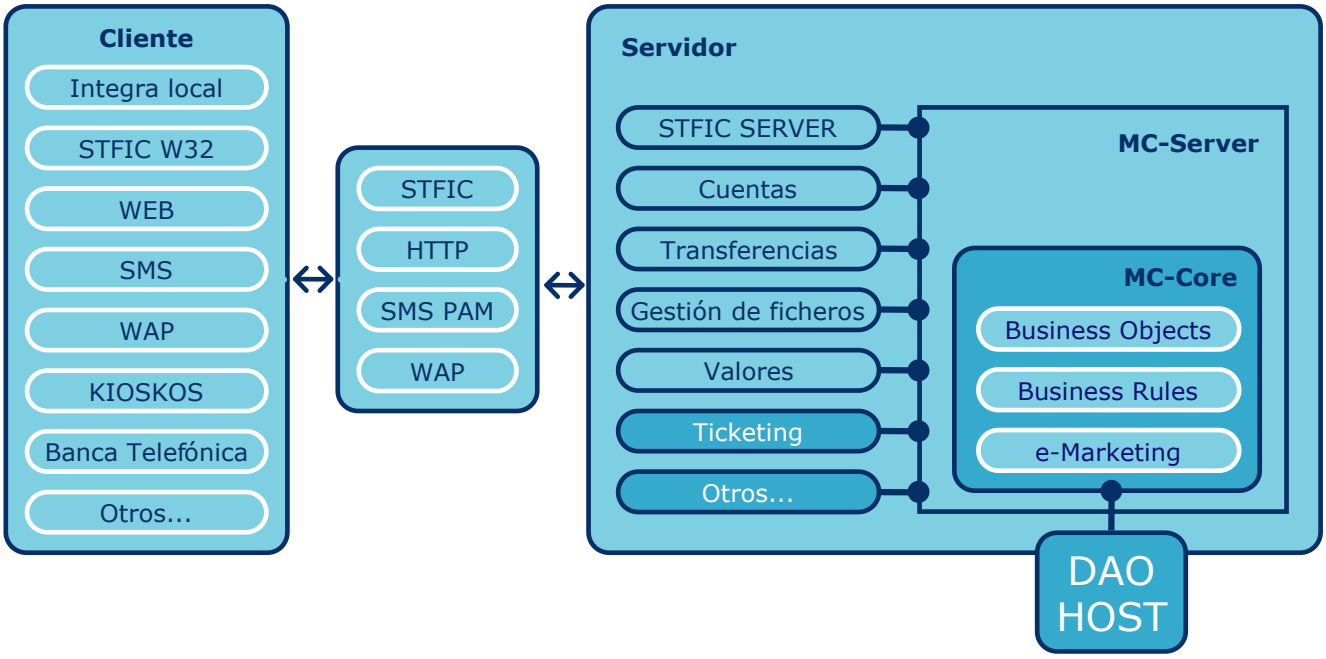


Figura 1- Servicios de MC-Server

El framework MC-Server se sustenta sobre las tecnologías más avanzadas en el campo de los entornos servidor J2EE. Comenzado a construirse bajo la plataforma Java 1.3, Enterprise Edition (J2EE), se ha seguido desarrollando e implementando hasta las versiones actuales de J2EE, siguiendo una serie de normas y patrones de diseño, que garantizan la flexibilidad, escalabilidad, disponibilidad e independencia de la plataforma, obteniendo al mismo tiempo un rendimiento excelente del sistema. Este último punto es fundamental y suele ser el punto débil de soluciones y arquitecturas diseñadas por empresas.

En la siguiente ilustración se muestra una visión general de la arquitectura técnica del producto, en la cual podemos observar una distribución de componentes en el Servidor Web y las dos partes fundamentales del Servidor de Aplicaciones: Motor de Servlets y Contenedor de EJBs.

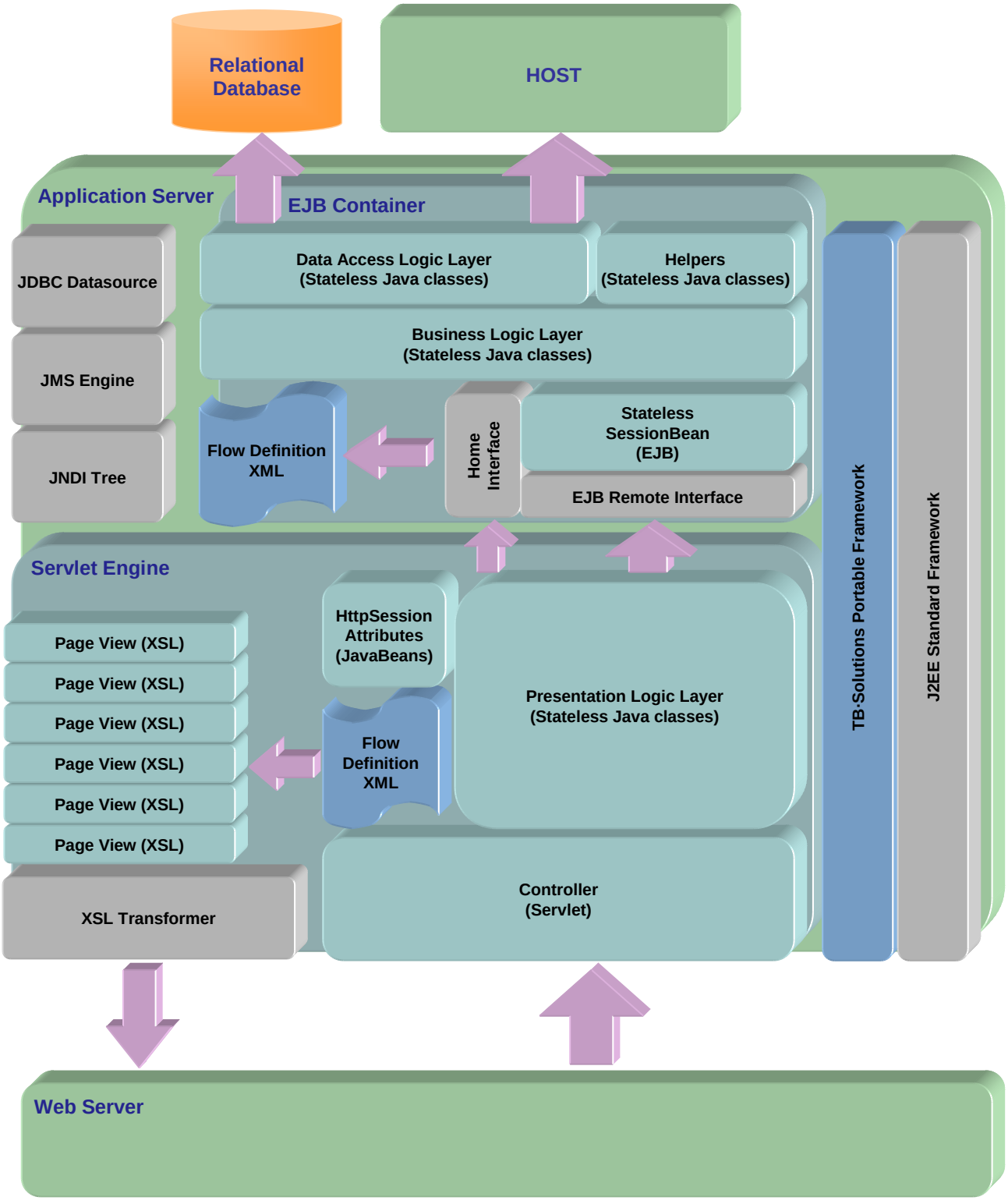


Figura 2- Arquitectura técnica MC-Server

Para detallar el flujo de una petición de una forma resumida, y simplificando la explicación para favorecer su comprensión, podemos decir que todas las peticiones se dirigen a un solo servlet el cual controla el tipo de petición que llega y decide el proceso de lógica de negocio tiene que ejecutar. Este proceso se comunica con el EJB sin estado (stateless), que ejecutará dentro de una transacción el proceso de lógica de negocio apropiado para esa petición. Este último será el que se comunique con la capa de acceso a datos que es la que se integra con la base de datos o con Host según sea necesario. Finalmente, la respuesta se devuelve al servlet, que decidirá qué transformador utilizar para la respuesta.

Las piedras angulares de la arquitectura son el uso del patrón Model-View-Controller junto con la minimización del uso de EJB y limitación del tipo de los mismos, dentro de la plataforma J2EE.

1.2 Situación inicial de partida

Como situación inicial, tenemos una aplicación de banca electrónica, con su base de datos particular, y con una arquitectura característica (explicadas anteriormente).

MC-Spy se debería de adecuar a todo ello, ya que como se ha explicado, estará integrado dentro de MC-Server, concretamente, en su administración. Todo ello supuso que diversas decisiones técnicas que se podían tomar a la hora de diseñar MC-Spy, vinieran ya impuestas por el ámbito en el cual se iba a implementar. Esto se explicará posteriormente, donde se desarrolla el Análisis del proyecto, en el anexo A de este documento.

1.3 Objetivo final

Como objetivo final se plantea conseguir los siguientes objetivos:

- Obtener una aplicación que muestre datos fiables y útiles para conocer el funcionamiento real de la banca electrónica. Lo que se pretende es obtener una serie de estadísticas sobre la utilización de la banca, y la obtención de predicciones sobre su evolución en un futuro a corto-medio plazo.
- Obtener una aplicación robusta y que no penalice apenas el rendimiento de la banca electrónica.
- Obtener una aplicación atractiva y sencilla de utilizar por cualquier empleado de la banca.

1.4 Ampliación

Sin embargo, una vez lanzado el proyecto, en sus primeras fases iniciales, se observó que MC-Spy no tenía por qué encorsetarse en proyectos destinados únicamente a Banca electrónica, sino que podía valer para cualquier otra aplicación web que tuviera diversos módulos u operativas y se quisiera hacer un estudio sobre su uso. Por lo tanto, se procuró que todas las decisiones que se tomaran a partir de ese momento fueran encaminadas a conseguir una mayor generalidad en los objetivos, y tener a la portabilidad como una de las prioridades fundamentales del proyecto.

Así pues, a los puntos anteriores que se plantearon en un principio como objetivos generales del proyecto, habría que añadir ahora uno más, surgido de esta observación, que sería:

- Conseguir la mayor portabilidad posible para poder integrar MC-Spy con otras distintas aplicaciones, incluso que no estén relacionadas con la banca electrónica.

Si bien, como hemos dicho, la portabilidad se convertía desde entonces en una prioridad, no debemos olvidar que la naturaleza de las aplicaciones desarrolladas en cualquier empresa es muy diversa. Las bases de datos, las arquitecturas, los servidores, y principalmente los datos que se requiere guardar y utilizar… varían de un proyecto a otro, con lo cual dicha portabilidad no va a poder ser ejecutada al 100% en general. Por lo tanto, dicha prioridad no debe ser tomada como una máxima ineludible, sino como una recomendación muy a tener en cuenta.

2 Requisitos

En los siguientes apartados se van a exponer los requisitos que debe cumplir este proyecto fin de carrera.

En primer lugar, se detallarán los requisitos funcionales para dar paso, a continuación, a los no funcionales. A modo de aclaración, conviene distinguir los dos tipos de requisitos: los no funcionales son aquellos que describen atributos del sistema o atributos del entorno del sistema; mientras que por el contrario, los requisitos funcionales, son los encargados de especificar las funciones que el sistema debe ser capaz de realizar, sin tener en cuenta restricciones físicas.

2.1 Requisitos funcionales

El objetivo funcional de esta aplicación expresado de forma genérica es la obtención de información acerca del uso de banca electrónica u otras aplicaciones Web, lo cual resulta prioritario en muchas entidades financieras para conseguir información precisa de la utilización del sistema por parte de los usuarios, obtener conclusiones acerca de qué operativas son excesivamente complejas y deberían revisarse, y finalmente para alimentar el sistema de marketing one-to-one, que muestra unos ciertos banners publicitarios específicos para cada usuario en función de su perfil.

Los requerimientos que en un principio se plantearon que debía de cumplir MC-Spy son los siguientes:

- MC-Spy debería ser capaz de mostrar gráficamente el número de usuarios que entraran a la aplicación durante un día, un mes o un año concreto, o a lo largo de una horquilla de fechas deseada.
- MC-Spy debería mostrar distintos tipos de gráficos acerca de diferentes datos interesantes para el funcionamiento de la banca electrónica, como por ejemplo la cantidad de dinero movido por transferencias en un intervalo de tiempo, la cantidad de ingresos hechos a lo largo de un día, o el número de consultas de movimientos realizados a lo largo del último año.
- MC-Spy debería de aportar información sobre los tiempos de ejecución de las distintas operativas de la banca electrónica, dato muy interesante para el cliente (es decir, el banco) para que pueda observar cuáles de estas operativas pueden no ser usadas por los clientes remotos debido a lo tedioso que puede llegar a resultar terminar dicha operativa.
- También se consideró importante el poder obtener datos sobre el porcentaje de operaciones que se realizaban con éxito, frente a las que terminaban con error, ya sea por algún tipo de falta de conexión con algún elemento de la banca, por caída da máquinas servidoras o por cualquier otro agente externo. Así mismo, se

sería deseable que además del porcentaje, se pudiera saber por cuál de estos errores las operaciones fallidas no habían terminado con éxito.

- Se planteó como muy interesante, el hecho de poder obtener lo que se denominó como "la sesión de un usuario". Entendemos por la Sesión de usuario a todas aquellas operaciones que ha realizado un cliente de la banca electrónica desde que se ha conectado a ella, hasta que la abandona. Este elemento sería muy útil a la hora de observar posibles errores reportados por los usuarios. De esta forma, se puede evitar el hecho de mirar logs.
  Se realizó una maqueta en html para concretar que era concretamente lo que se deseaba mostrar con este aspecto, la cual mostramos a continuación:

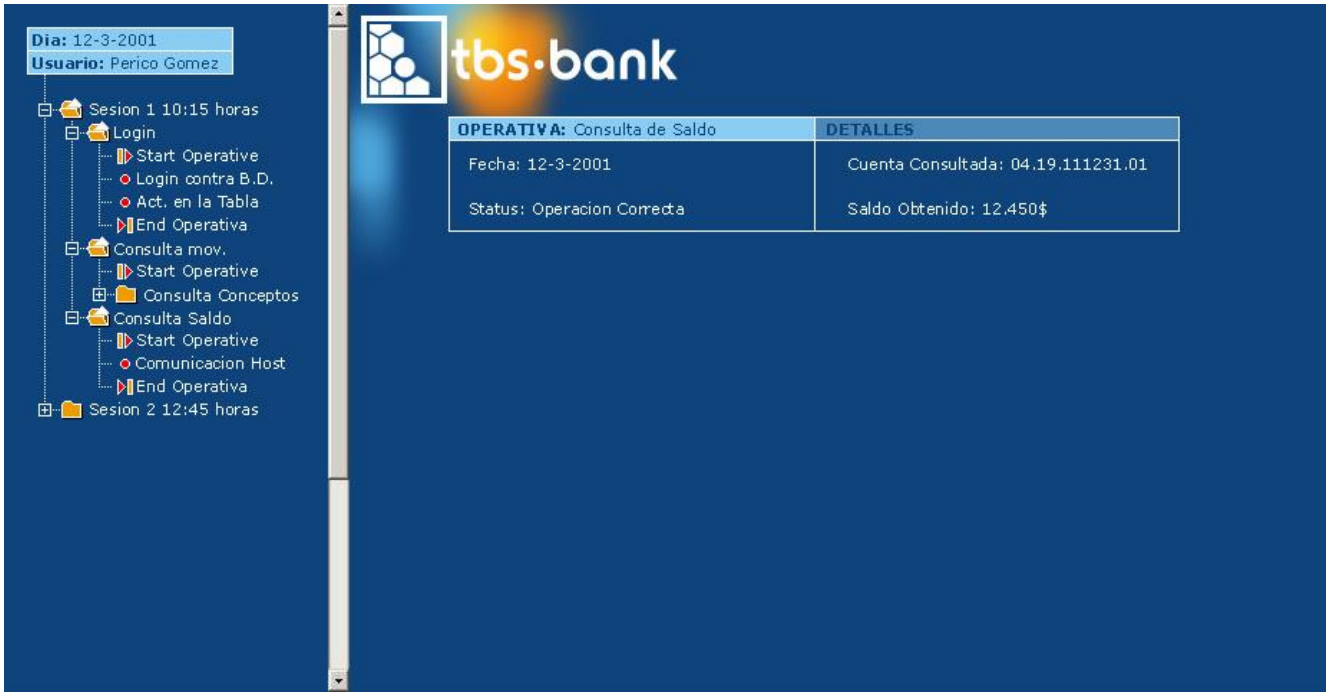


Figura 3- Aspecto maqueta de MC-Spy

- Se solicitó así mismo la posibilidad de realizar algo que podría resultar muy útil también para la entidad financiera. Se trataba de poder realizar previsiones futuras para los datos mostrados por las distintas estadísticas anteriormente nombradas. Es decir, dados unos datos a lo largo de un intervalo de tiempo, usando los algoritmos necesarios, hacer una previsión de cuales podrían ser los datos futuros (para un tiempo a corto plazo).
- Así mismo se considera interesante, aunque quizá no entrara dentro del proyecto, el realizar un visualizador de ficheros de traza.

Todos estos requerimientos fueron propuestos por los miembros del equipo que en ese momento se encargaba del diseño y desarrollo de la aplicación de Banca electrónica que se ha comentado anteriormente, todos ellos participantes en su análisis e implementación.

2.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen atributos del sistema o atributos del entorno del sistema. A continuación se detallan los requisitos no funcionales que se han intentado cumplir:

- Integración completa del módulo con el resto de la aplicación MC-Server.
- Posibilidad de reutilización de ciertos componentes utilizados.
- Compatibilidad con los navegadores más utilizados en el mercado.
- Uso de Java Enterprise Edition (JDK 1.3, …)
- Uso de Oracle como Gestor de bases de datos.

3 MC-Spy

Una vez conocidos los orígenes del proyecto, y los requisitos que se le piden, vamos a describir brevemente qué es y cómo funciona MC-Spy.

3.1 Breve descripción de MC-Spy

MC-Spy se podría definir brevemente como una aplicación web destinada al almacenamiento y gestión de cierta información para su posterior explotación estadística mediante informes y gráficos.

La información siempre es relativa a los usos que los clientes hacen a su vez de otra aplicación web. Como hemos dicho, inicialmente estaba destinado únicamente a servir a una banca electrónica (MC-Server), pero luego se ampliaron las posibilidades de aplicaciones destino. Sin embargo, a partir de ahora, nos referiremos siempre a MC-Spy como un módulo de MC-Server, ya que éste fue su objetivo inicial, y con el que surgió y se desarrolló este proyecto fin de carrera.

Así pues, MC-Spy es capaz de mostrar diversos tipos de gráficos e informes, alimentándose de información recopilada de base de datos, y que previamente ha sido almacenada por él mismo.

Para verlo más fácilmente, vamos a poner un ejemplo a través de una entrada por parte de un usuario en la banca electrónica:

- El usuario entra en la aplicación de banca electrónica MC-Server que su banco pone a su disposición, y realiza el login.

  En ese momento, MC-Spy registraría esta entrada del usuario en base de datos.

- Ahora el usuario realiza una consulta de saldo de su cuenta, para posteriormente realizar una transferencia. Lo intenta, pero justamente cuando intenta realizarla, se pierde la comunicación con la banca, por lo que da un error. Al rato lo intenta de nuevo, y esta vez si, consigue realizar la transferencia. Después de eso, abandona la banca.

  MC-Spy registraría estas tres operaciones en base de datos, incluida la transferencia que ha registrado error.

Una vez aquí, MC-Spy ha almacenado ya toda la información que necesita. Ahora es labor de los administradores de la banca o de sus empleados encargados de auditar esta banca electrónica, el solicitar a MC-Spy la información que ellos consideren necesario. Por ejemplo:

- El usuario que ha entrado anteriormente a la banca electrónica, llama alarmado a la central de llamadas del banco pidiendo que le expliquen porqué le ha dado un error mientras realizaba una transferencia. Para ello los administradores entran en MC-Spy y, tras observarlo, ven que ciertamente el usuario hace poco tiempo que ha entrado a la banca y ven que ha realizado una transferencia que no ha terminado satisfactoriamente. Observan el código de error, le informan la naturaleza del error, y tranquilizan al usuario.
- Los administradores de la banca han recibido el encargo de sus superiores de que les den datos reales del uso que se está realizando de la operativa de transferencias a través de la banca electrónica. Para ello, los administradores entran a MC-Spy, y consultan las siguientes estadísticas que esta aplicación les ofrece:
    - Numero de transferencias realizadas en el último año.
    - Importe medio de las transferencias realizadas durante los 12 últimos meses.
    - Importe total de las transferencias realizadas a lo largo de los 36 últimos meses.
    - Porcentaje y número total de transferencias que han terminado con error al ejecutarse a lo largo de toda la historia de la banca electrónica.
    - Porcentajes sobre los bancos destino de las transferencias realizadas.
    - Previsiones del importe movido por la banca en transferencias para los próximos 24 meses.

Mediante estos datos, los mandos directivos de la banca podrán observar gráficamente si la operativa de transferencias se está usando en gran medida en su banca electrónica, y no sólo si se usa, sino si mes a mes está aumentando su utilización por parte de los usuarios. También pueden ver qué entidad financiera es la que más transferencias recibe desde su banca electrónica. De igual modo pueden observar si el número de operaciones erróneas supera un límite a partir del cual podría ser preocupante, y podría llevar a sus usuarios a abandonarlos como usuarios. Finalmente, pueden prever de forma aproximada cuál va a ser el volumen de operaciones que se realicen en su banca a través de Internet durante los siguientes meses.

Esto sería en resumen un ejemplo breve (y real) de lo que MC-Spy puede ofrecer a toda aplicación que lo integre consigo.

3.2 Arquitectura del sistema

Pasamos ahora a reseñar brevemente cual es la arquitectura de MC-Spy. Del porqué de esta división y de otros aspectos de su creación se habla más extensamente en el Anexo A, que expone el Análisis.

Como ya se ha podido adelantar previamente, MC-Spy está formada por dos módulos principales:

- Módulo de recolección de datos
- Módulo de Administración y consulta de estadísticas.

3.2.1 Módulo de recolección de datos

El módulo de recolección de datos se encarga recoger la información de las operativas ejecutadas en la Aplicación Web monitorizada por MC-Spy y almacenarla en base de datos.

En el siguiente diagrama se puede ver el funcionamiento del módulo de recolección de datos.

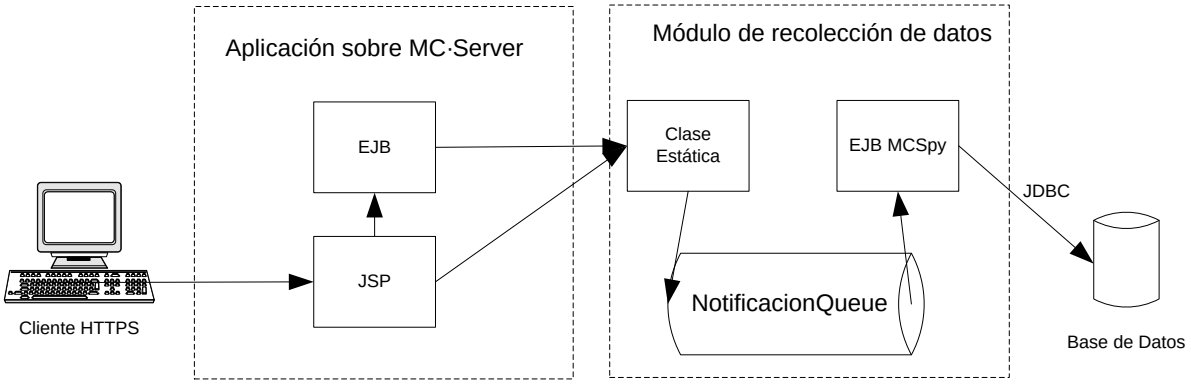


Figura 4- Modelo de Recolección de datos

Los componentes que se ejecutan sobre la plataforma MC-Server (EJB's, JSP's, Servlet's, etc.), cuando deseen registrar información relativa a una operativa de la aplicación, invocarán a una clase estática que encolará la información en una cola JMS, para su posterior almacenamiento en Base de datos. De este modo, la operación del registro de la información es asíncrona y no penaliza en rendimiento a la ejecución de la operativa de la aplicación "anfitrión" que se va a monitorizar.

Para más información sobre estos aspectos, podemos ir al Anexo B donde se explica el diseño de la aplicación.

3.2.2 Módulo de Administración y consulta de estadísticas

Este módulo consiste en una aplicación Web que permite a los usuarios administradores del sistema ver la información de los usuarios que han accedido al sistema, conocer el número de operativas que han ejecutado y consultar la estadísticas y previsiones que deseen.

En el siguiente diagrama se puede ver el funcionamiento de este módulo de Administración:

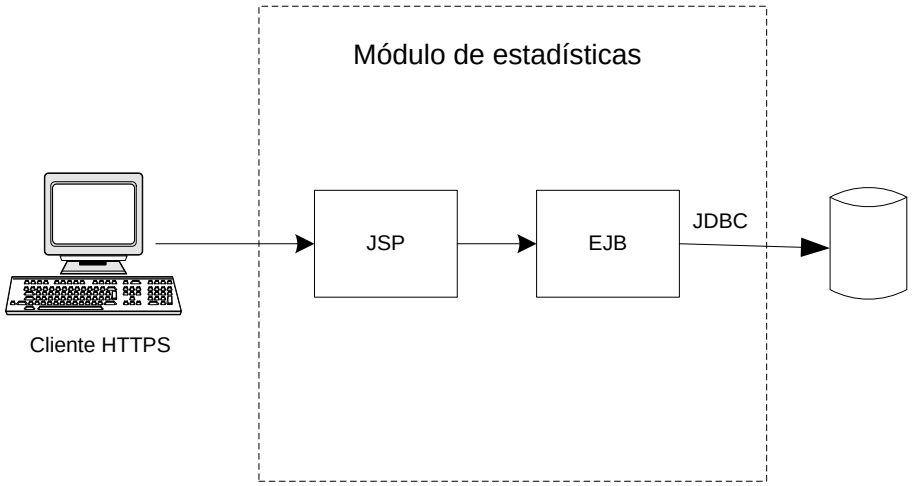


Figura 5- Módulo Administración o de Estadísticas

El Módulo de Administración consiste en una serie de páginas JSP's que mostrarán la información de Administración y estadísticas a los usuarios. Para el acceso a la base de datos las páginas JSP de la aplicación invocarán a un EJB.

Para un mayor detalle del diseño de este módulo, se recomienda ir al Anexo B de este mismo documento.

4 Desarrollo del proyecto

En este capítulo se va a intentar resumir el trabajo realizado en este proyecto fin de carrera de una forma general.

La realización de MC-Spy consta de dos fases perfectamente diferenciadas.

1. Una primera fase del proyecto consiste en la realización de una consultoría que proponga modificaciones de la información reflejada en los logs o almacenada en la base de datos de MC-Server, para su posterior explotación mediante MC-Spy. Las propuestas que surgen como resultado de este estudio deben tener como objetivo la ampliación y refinamiento de la información histórica almacenada para que la aplicación que se desarrollará en la segunda fase de este proyecto la analice.
2. La segunda fase del proyecto consiste en la creación de una aplicación web que explote la información obtenida en dichas fuentes y generando estadísticas de utilización de las distintas operativas que conforman el sistema de e-banking subyacente.

4.1 Fases del desarrollo

El desarrollo se ha dividido en varios puntos, para ir marcando hitos en su realización.

Todo el proceso de trabajo y la forma de realizarlo se explica brevemente en los siguientes puntos.

4.1.1 Fase inicial de estudio

- **Investigación y estudio a fondo de MC-Server**: Para poder integrar MC-Spy dentro del módulo de la Administración de MC-Server, hay que investigar como se ha desarrollado MC-Server (aplicación "anfitrión"), cuál es su arquitectura, como se define su base de datos u otros sistemas de almacenamiento de información, y otros datos referentes a su estructura y funcionamiento.
- **Consultoría**: Se estudian las posibles implicaciones que suponen la integración de MC-Spy dentro de MC-Server, teniendo en cuenta las graves consecuencias que puede tener el hecho de realizar algún cambio no oportuno para el funcionamiento correcto de la banca electrónica.
- **Estudio teórico sobre las series de tiempo**: La realización de previsiones basadas en series de tiempo, requieren que se realice un estudio relativamente detallado sobre éstas, concretamente sobre las relacionadas con la economía, como es este caso.

- **Análisis**: Se realiza el análisis técnico de la solución, que se incluye en los anexos a este mismo documento.

4.1.2 Fase de desarrollo

- **Cambios en MC-Server**: Realización de las modificaciones surgidas en el estudio de los puntos anteriores(al menos las referidas a la base de datos).
- **Diseño de las páginas de la Administración**: Se diseñan las pantallas sobre las que se mostrarán las gráficas, las que soliciten datos al usuario, además de los menús por los que se accederá a todas estas páginas.
- **Diseño e implementación del sistema de recolección de datos**: Se decide el sistema que se usaría para recoger la información de las operativas ejecutadas en la Aplicación Web monitorizada por MC-Spy y almacenarla en base de datos. Ésta es la segunda parte de los cambios que habría que realizar en MC-Server para su integración con MC-Spy (después de los realizados en la base de datos). Se trata en primer lugar de estudiar en qué punto del código se realizan cada una de las operativas que queremos monitorizar, para posteriormente, realizar los cambios oportunos y necesarios en el código.
- **Implementación del sistema de almacenamiento de datos**: Para conservar toda la información en la base de datos de lo que los usuarios operan en la banca electrónica, se decidió implementar un sistema de colas, en el cual la aplicación fuera dejando los datos, y ésta fuera insertándolos en la base de datos asíncronamente.
- **Implementación de un árbol dinámico**: Desarrollo de un árbol mediante javascript y html dinámico, para mostrar los datos que se requieren en el módulo del Árbol de usuarios.
- **Implementación de la operativa del Árbol de sesiones**: Se crean e implementan todas las clases necesarias para este módulo, tanto para acceder a la base de datos para ir a buscar la información, tanto como para tratar dichos datos y pasárselos al jsp para mostrarlos. También se desarrollan estos jsp, los cuales son los encargados de recoger los parámetros de búsqueda y a su vez mostrar la información requerida por el usuario.
- **Pruebas exhaustivas del Árbol de sesiones**: aunque tras cada pequeño avance se probaba la aplicación, en este punto se realiza un exhaustivo plan de pruebas al Árbol de usuarios, donde se corrigen algunos puntos surgidos en esta fase.
- **Implementación de la operativa de Estadísticas**: Se desarrollan las clases necesarias para este módulo, al igual que los jsp que muestran, ya sea de forma gráfica o en modo texto, la información sobre el uso de las operativas de la banca.

- **Pruebas exhaustivas de las Estadísticas**: Se prueban concienzudamente todas las gráficas y estadísticas que se ofrecen en este módulo, para comprobar su correcto funcionamiento, sobre todo la validez de sus datos.
- **Implementación de los algoritmos para las previsiones**: Se codifican los algoritmos que se eligieron en el estudio previo, para realizar las previsiones de las series de tiempo.
- **Implementación de las Previsiones**: Aunque está muy relacionado con el módulo anterior, ya que las previsiones también se muestran en forma de estadística, esta parte de desarrolla una vez terminada la fase anterior, para establecer una mayor independencia entre ellos.
- **Pruebas exhaustivas de las Previsiones**: Se realizan continuas pruebas mientras dura el desarrollo de esta parte, comparando los resultados obtenidos mediante los algoritmos codificados, y los que deberían haber salido mediante los algoritmos teóricos.

5 Modelo incremental

Tal y como se ha visto en al apartado anterior, el desarrollo se ha dividido en varios sub-desarrollos. Esto implica que algunos de ellos necesitaran que el desarrollo anterior estuviera plenamente operativo.

Debido a esto, se ha seguido un modelo incremental ya que de esta forma los desarrollos posteriores podían implementarse con la seguridad de que las funcionalidades que necesitan eran correctas.

Además, de esta forma se pudo seguir trabajando con nuevos módulos mientras se validaban los módulos implementados.

En la siguiente figura se muestra el esquema del modelo incremental:

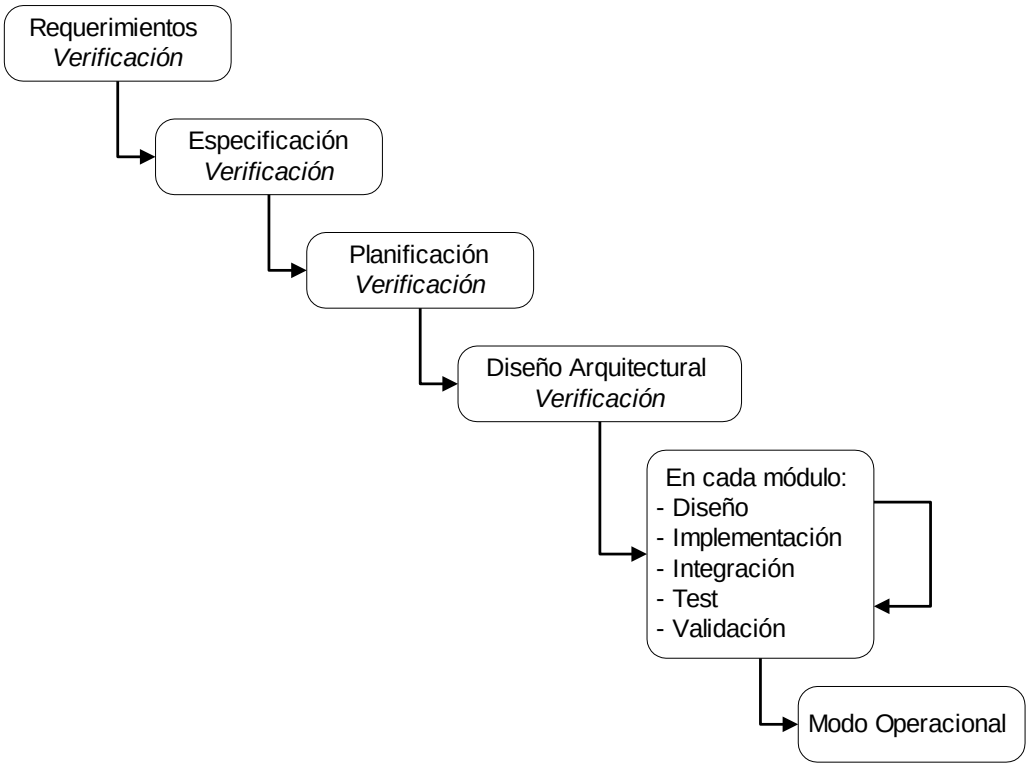


Figura 6- Modelo Incremental

6 Gestión del proyecto

En este apartado se explican las labores de gestión del proyecto que no están directamente relacionadas con el desarrollo del mismo, pero que son necesarias para conseguir el objetivo final.

6.1 Gestión de riesgos

Detectar de forma temprana los riesgos que puedan aparecer en el desarrollo del proyecto, contribuye a una mejor planificación del mismo.

Los principales riesgos detectados, y las acciones necesarias para superarlos son los siguientes:

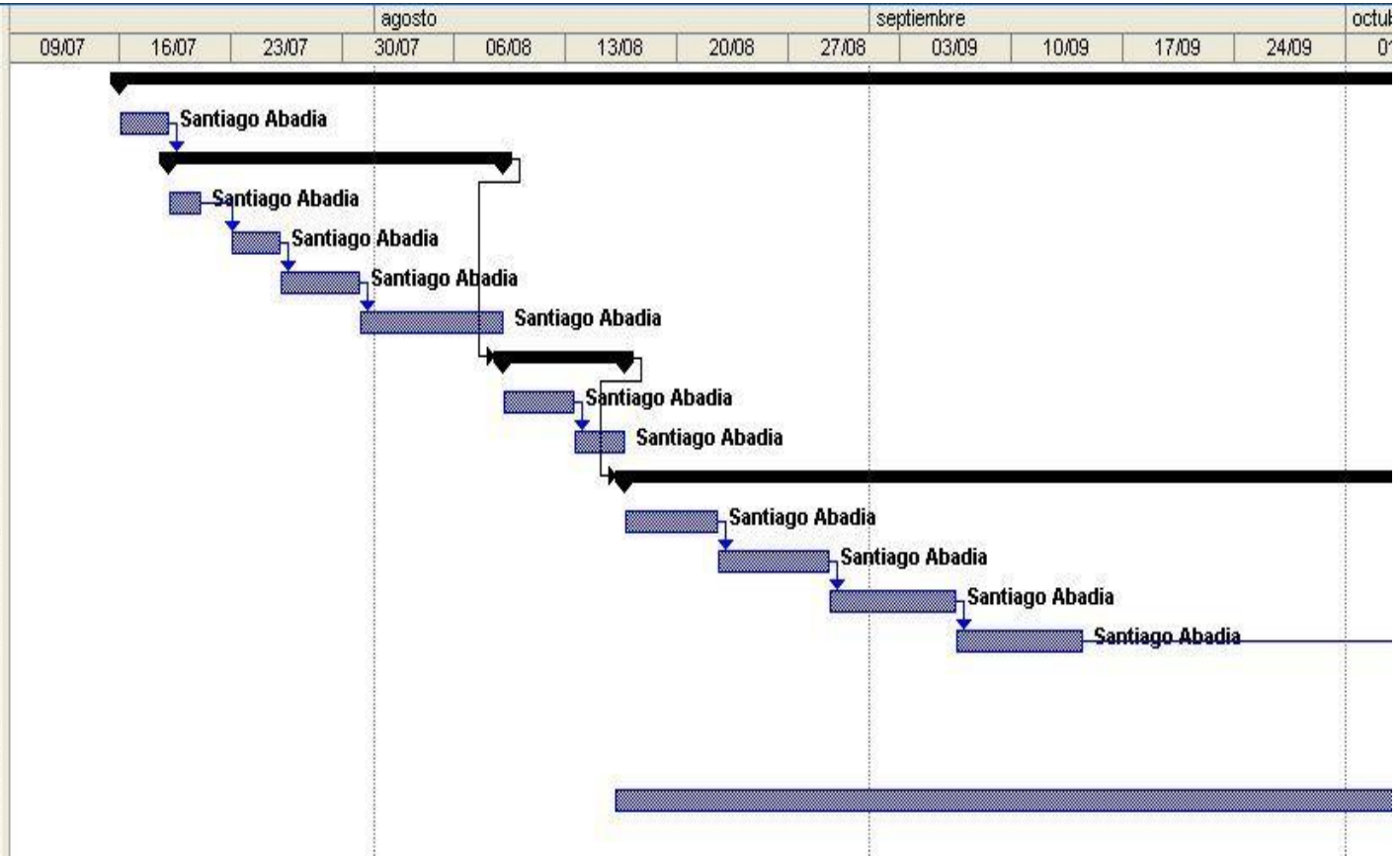
- **Dependencia de aplicaciones existentes**: ya que el desarrollo debe integrarse con unos módulos existentes, antes de comenzar el desarrollo de las nuevas funcionalidades debe conocerse a fondo estas aplicaciones; ya que empezar a realizar los nuevos componentes sin conocer la aplicación donde deben integrarse es una forma segura de tener que repetir los avances realizados.
- **Aplicaciones en desarrollo**: las aplicaciones en las que se integra el proyecto no están cerradas, sino que avanzan de versión continuamente. Debe ser necesario planificar un tiempo para integrar los nuevos desarrollos con las nuevas versiones. El tiempo necesario para realizar estas integraciones puede minimizarse si el proyecto se realiza de la forma más independiente posible de la aplicación existente.
- **Falta de experiencia:** debido a que este desarrollo es un proyecto fin de carrera, el proyectando no posee una gran experiencia en la realización de proyectos de esta envergadura. No hay solución fácil para este riesgo, simplemente participar en más desarrollos y dejarse aconsejar por gente con más experiencia.
- **Desconocimiento de algunos temas que se tratan:** asuntos como por ejemplo las previsiones de series de tiempo, son temas que el proyectando, a pesar de haber estudiado alguna asignatura relacionada con la estadística, no domina, con lo cual, es necesario realizar una tarea de estudio previa. Esto puede ser un cuello de botella si los conocimientos necesarios para desarrollar la implementación no se consiguen en los plazos previstos. Por tanto, no se asignan unos tiempos determinados para realizar labores de investigación. Todos sabemos lo complicado que resulta a veces encontrar lo que se busca.
- **Simultaneidad con otros proyectos**: también es digno de mencionar un hecho que puede ser considerado como un riesgo para la elaboración del proyecto, y es que a la vez que se realizaba este proyecto, el autor del mismo estaba trabajando en otros proyectos, siempre en la misma empresa. Como se podrá observar en el

apartado siguiente, durante el ciclo de vida del proyecto ha habido varios huecos en los que por motivos laborales el autor no pudo dedicarse a las labores de desarrollo, lo cual es el motivo principal del retraso en la fecha estimada inicialmente para la finalización del proyecto.

6.2 Plazos del proyecto

Ver gráfico a continuación.

	Nombre de tarea	Trabajo	Comienzo
1	**MC-Spy**	**507 horas**	**lun 16/07/01**
2	**Identificacion de requisitos**	24 horas	lun 16/07/01
3	**Analisis**	**120 horas**	**jue 19/07/01**
4	Estudio previo	16 horas	jue 19/07/01
5	Consultoria	24 horas	lun 23/07/01
6	Estudio series tiempo	24 horas	jue 26/07/01
7	Analisis tecnico	56 horas	mar 31/07/01
8	**Diseño**	**46 horas**	**jue 09/08/01**
9	Diseño gráfico	18 horas	jue 09/08/01
10	Diseño tecnico	28 horas	lun 13/08/01
11	**Implementacion**	**249 horas**	**jue 16/08/01**
12	Modificaciones MC-Server	28 horas	jue 16/08/01
13	Modulo insercion datos	40 horas	mié 22/08/01
14	Modulo almacenamiento datos	48 horas	mié 29/08/01
15	Modulo arbol de sesiones	48 horas	jue 06/09/01
16			
17	Modulo estadisticas	35 horas	lun 20/05/02
18	Modulo previsiones	50 horas	vie 24/05/02
19	Pruebas	60 horas	jue 16/08/01
20	Cierre	8 horas	lun 10/06/02



7 Conclusiones

7.1 Cumplimiento de objetivos

Una vez finalizado este proyecto fin de carrera se puede afirmar el cumplimiento global de los objetivos que se propusieron en sus inicios.

Si se repasan dichos objetivos que se plantearon en un principio para este proyecto, podemos decir con toda seguridad que el cumplimiento es total, y el grado de satisfacción de la calidad obtenida, tanto por parte de TB-Solutions, como por parte de los clientes receptores (que por lo tanto, usan la aplicación), es máxima.

Únicamente, si miramos uno a uno los requisitos que se enumeraron en el punto 2 de este documento, podríamos decir que el único punto de todos ellos que no se ha realizado es el que se refiere a la realización de un visualizador de ficheros de traza. Pero ya se indicó en su momento (al inicio del proyecto, y así se indica también en dicho punto de este documento), que este punto quizá estuviera fuera del proyecto, y solamente se realizaría en el caso de que los tiempos de finalización del resto de la aplicación se hubiesen cumplido con creces.

7.2 Extensiones no previstas en los objetivos iniciales

En este punto quisiera agradecer a los jefes de proyecto de MC-Server que no incluyeran durante el desarrollo de este PFC ningún añadido a lo que inicialmente se propuso por parte de ellos. Este hecho facilitó la realización y su conclusión, al no verse interrumpido por nuevas solicitudes que hubiera que habido que estudiar e integrar, con las consecuentes penalizaciones de tiempo.

Si acaso, reseñar el hecho de que una vez en fase de estudio (es decir, nada más comenzar el proyecto), se sugirió que la aplicación debería ser lo más portable posible, para poder ser utilizada no sólo en MC-Server, como era la proposición inicial, sino en cualquier otra aplicación web que se desarrollara en la empresa. Dicho esto, es justo decir que la sugerencia se hizo a tiempo, además de que desde un primer momento ya se intentó que este objetivo se consiguiera.

7.3 Implantación de MC-Spy en sistemas reales

Actualmente MC-Spy está siendo usado como un módulo más en los siguientes entornos reales:

- MC-Server para Banca Mora.
- YAFAR(Sistema de gestión de tributos) para la DGA.
- YAFAR para la Junta de Andalucía.

- YAFAR para la Junta de Castilla y León.
- YAFAR para el Gobierno de La Rioja.

Se llegó a poner también en funcionamiento en el portal denominado Supergestoría (un portal web orientado a las gestorías), cuyo servicio era ofrecido por el Banco Santander Central Hispano, pero dicho portal nunca llegó a salir a la luz, por problemas ajenos a la empresa que lo desarrolló, sino por motivos económicos o de marketing del citado banco.

7.4 Posibles extensiones futuras

Quedan abiertas para el futuro algunas líneas de trabajo que permitirán mejorar el producto actual. Sobre todo en el módulo de Previsiones, queda mucha labor para hacer en el caso de que esta parte en un futuro se quiera llevar a una comercialización más determinada.

Por lo demás, aunque siempre es posible mejorar la calidad de un producto, MC-Spy cumple perfectamente los objetivos que se le plantearon, como lo demuestra su uso en los proyectos anteriormente citados, y que para la integración con cada una de las respectivas aplicaciones no hubiera que hacer apenas modificaciones.

7.5 Valoración personal

Este proyecto ha permitido un marco para el desarrollo y puesta en práctica de los distintos conocimientos adquiridos a lo largo de los cinco años transcurridos en la universidad, tanto en las asignaturas obligatorias de la carrera como en aquellas asignaturas optativas y de libre elección. También ha supuesto una gran oportunidad para entrar en contacto con tecnología innovadora, aplicada a proyectos reales, así como el desarrollo de un proyecto de principio a fin dentro de la empresa.

La utilización del lenguaje Java en la realización de este proyecto me ha servido para profundizar mucho más en él.

También el desarrollo de este proyecto me ha hecho conocer un poco más el tema de las predicciones estadísticas, un tema muy poco conocido, y bastante interesante, en el cual no me hubiera adentrado posiblemente nunca a no ser por la realización de este proyecto fin de carrera.

7.6 Valoración por parte de la empresa

Por parte de la empresa, el resultado obtenido por parte de MC-Spy en todos aquellos proyectos en los que se ha incluido ha sido bastante positivo. La muestra de ello es que se incluyó como módulo en la mayoría de los grandes proyectos que se hicieron durante los años posteriores a su realización.

También a destacar la opinión de los clientes finales de esas aplicaciones, que mostraron su aprobación al resultado del proyecto.

8 Bibliografía

- Métodos de predicción en economía. 1ª ed. 1993
  Antonio Aznar Grasa y Francisco Javier Trívez.
- Forecasting and Time Series Analysis. 2ª ed. 1990
  Douglas C. Montgomery, Lynwood A. Johnson y John S. Gardiner.
- Artículo publicado en el Journal of Forecasting (nº 4) en 1985 (pags. 1-28)
  Por John S. Gardiner
- Forecasting: methods and applications. 3ª ed. 1998
  Spyros Makridakis, Steven C. Wheelwrigth y Rob J. Hyndman.
- The Unified Modeling Language Reference Manual. Ed. 1999
  James Rumbaugh
- Website de Java Sun MycroSystems
  Disponible en http://java.sun.com

Anexo A. Análisis

Se va a proceder a enumerar todos los tipos de análisis del proyecto que se hicieron, previos a la implementación del mismo, y que sirvieron para asentar sus bases y plantear cuales eran los requerimientos que se le pedían.

A.1 Primeras decisiones

Al comenzar la realización de este proyecto, lo primero que se plantearon fueron diversas opciones sobre cómo realizarlo. Algunas de ellas eran dudas que se puede decir que estaban resueltas casi antes de plantearse, ya que la decisión no dependía del proyectando, ni tan siquiera algunas de ellas dependían de la empresa, sino que ya estaban determinadas por el cliente o por las circunstancias del sistema en el cual se debía de integrar.
De todas formas, pasamos a enumerar todas estas decisiones, para que conste el flujo de ideas que surgió.

A.1.1Aplicación local vs. Aplicación web

Esta es una de esas decisiones que hemos nombrado, la cual no se pudo ni plantear desde un principio, ya que se vio que tenía mucho más sentido que MC-Spy estuviera integrado dentro de la administración de MC-Server, y al ser ésta una aplicación web, no tenía sentido hacerla separada.
De todas formas se plantearon los pros y los contras de ambas opciones. Aunque no se ha dicho hasta ahora, siempre se da por supuesto que la aplicación local también sería realizada bajo tecnología Java. Dicho esto, ninguna de las dos impedían la realización en si del proyecto, ya que las dos permitían hacer lo que se quería(al menos a grandes rasgos), y aunque ambas tenían sus ventajas e inconvenientes, salió vencedora de este duelo la opción de la aplicación web, al ser la tecnología web mucho más accesible que la otra.
Aparte de estas razones que por si solas ya bastaban como para justificar la decisión, también habría que añadir que el proyectando desconocía casi por completo los componentes gráficos de Java de carácter local (Swing,…), con lo cual es de entender que el proyectando también prefiriera esta opción.
Con esto también se determinaron otras decisiones, tales como el servidor de aplicaciones a utilizar, que en un principio fue JBoss, aunque el servidor no debía de ser algo dependiente para que la aplicación funcionara, ya que este aspecto debería de ser independiente para su correcto funcionamiento.

A.1.2 Base de datos vs. Log

Esta fue una de las decisiones que más discusiones y tiempo empleado requirieron para su resolución.
En resumen, la situación era la siguiente. Se vio que MC-Spy iba requerir información para poder mostrar todos los datos que se le pedían. La cuestión era de donde sacar toda esa información, donde iba a estar almacenada y cómo se iba a poder obtener.
En la banca, en el momento de iniciar este proyecto, la información estaba distribuida entre los logs, en los cuales escribía la aplicación, y entre la base de datos, la cual era actualizada por MC-Server al ir accediendo los usuarios a alguna de sus operativas. Pero en ninguna de las dos opciones la información era homogénea, ni, sobre todo, en ninguna de las dos estaba toda la información que se creía se iba a tener que necesitar para poder mostrar todos los datos solicitados.
Así pues, había que tomar una decisión, sobre si adoptar uno de los dos métodos como el único para rescatar todos los datos necesarios (eso si, modificando aquel que se eligiera de ellos en profundidad), o bien adoptar una solución intermedia, escogiendo la información de un lado o de otro, según el caso.
Para ello, se realizó una recolección de elementos con los que calificar cada una de las opciones, que aquí planteamos.

A.1.2.1 Espacio en disco

Este era un punto fundamental para tomar una decisión sobre el problema que se planteaba. La gran cantidad de datos que se preveía que se iba a tener que conservar, hacía que la optimización del espacio que hiciera la opción elegida debería ser un aspecto muy a tener en cuenta.
Por lo tanto, se observó la capacidad que cada una de las dos opciones tenían para poder almacenar la máxima cantidad de información, ocupando la menor cantidad de espacio en disco posible. Para ello, dado que el proyecto MC-Server ya estaba desarrollado, se observó el espacio que ocupaban los ficheros de logs que generaba la aplicación, que ya de por si eran grandes, y se calculó que aproximadamente, al añadir la información que MC-Spy requería, éstos podían llegar a crecer más de un 50%, pudiendo significar esto varios megas de espacio, según el volumen de negocio de la banca o de la aplicación que lo contuviera.
Sin embargo con la base de datos se pudo estimar que el nivel de crecimiento no llegaría nunca hasta esos niveles, además de que mediante las bases de datos se pueden establecer criterios de optimización de espacio mediante la definición de los tipos de datos, u otras técnicas que pueden llegar a ofrecer los sistemas gestores de bases de datos.

A.1.2.2 Accesibilidad

Nos referimos con accesibilidad a la facilidad que el proyectando iba a tener a la hora de poder recoger la información almacenada. En este aspecto no había duda alguna de que

las bases de datos ofrecen mucha mejor accesibilidad que los ficheros de log, ya que el acceso a éstos está muy penalizado en los lenguajes de programación, al tener que leerlos de disco y cargarlos en memoria. Por supuesto se podían adoptar algoritmos y funciones para acelerar ese rendimiento, pero por otro lado estaba también el problema antes comentado, del posible tamaño de los ficheros, lo que dificultaría enormemente de nuevo el acceso a ellos.

A.1.2.3 Seguridad

En este aspecto ambas opciones estaban equilibradas, ya que se supone que los administradores de la banca o de la aplicación que albergue MC-Spy, realizarán las tareas de restricción de permisos necesarias para que tanto la base de datos como los ficheros de logs sean solo accesibles por los administradores y las personas que tengan los privilegios necesarios para ello.
De todas formas, existe un aspecto que puede llegar a ser problemático, dentro de los temas estrictamente legales, y es que tanto en los ficheros de log, como en las tablas de bases de datos, habrá datos que en ningún caso pueden ser vistos por personas ajenas que no sean el usuario mismo de la banca, o personas a las que se les hayan otorgado permisos. Esto es entendible, ya que entre los datos que se van a guardar se encuentran cantidades de dinero movidas entre cuentas, préstamos solicitados, o cualquier otro dato que se pueda relacionar con las acciones de una banca electrónica. Y ante esto, y debido a (como ya se ha citado anteriormente) la experiencia de las empresas en este tipo de proyectos, es algo normal que si algo falla en la aplicación, se manden los logs a los desarrolladores para que puedan averiguar la causa del error. Con lo cual estaría en poder de personas no autorizadas datos que no deberían de estar.
También se apuntó que este último problema se podría solucionar duplicando el sistema de log, es decir, con la existencia de dos ficheros de logs, uno con la actividad normal, y otro exclusivamente para los datos requeridos por MC-Spy.

A.1.2.4 Disponibilidad

Nos referimos a disponibilidad con la posibilidad de que algún sistema que compone la aplicación se caiga, haciendo que la obtención de datos resulte imposible.
En ninguna de las dos opciones esto parece ser un problema. Si es la base de datos la que se cae, desde luego que hará que MC-Spy no funcione, pero eso haría que tampoco funcionara la aplicación general, con lo cual no sería un problema (para MC-Spy se entiende, para la aplicación general claro que lo sería). Esto se podría solucionar ubicando las tablas necesarias para MC-Spy en otra base de datos distinta a la de la aplicación, pero esto no parece tener mucho sentido.
Por otro lado, en principio no hay nada que impida obtener información de los ficheros de log, a no ser que sea el mismo MC-Spy el que esté caído, con lo cual no sería mayor problema, ya que ni siquiera se podrían solicitar los datos.

A.1.2.5 Información

En este aspecto nos referimos a la posibilidad de que con alguna de las dos opciones se pueda dar el caso de que alguno de los datos que se pretenden conservar para su posterior extracción, no pudieran ser guardados por algún motivo.
Sin embargo no apreciamos motivo alguno para que este aspecto pueda llegar a ser un problema, ya que no observamos ninguna razón para que ni los ficheros de log ni la base de datos impidan la conservación de algún tipo de dato.

Con todas estas conclusiones parciales obtenidas, se decidió que la opción que más ventajas ofrecía para la correcta obtención de los datos era la de la utilización de la base de datos. La elección del sistema gestor de base datos a elegir estaba claro ya que MC-Server utilizaba Oracle, aunque el sistema a usar no debería de ser un problema (por supuesta fuera de la definición de las tablas en sí, pero no así en su utilización), ya que MC-Spy debería ser lo más portable posible, y debería permitir el uso de otros sistemas gestores.

A.1.2 Composición del sistema

Siguiendo las restricciones y los requerimientos impuestos, todos ellos comentados ya anteriormente, podemos ya realizar el análisis del proyecto, dividiéndolo en varias líneas de proyecto que en conjunción completarían lo que sería MC-Spy.

- Recolección de datos
- Consulta de estadísticas
- Árbol de sesiones
- Previsiones

A.1.2.1 Recolección de datos

El módulo de recolección de datos será el encargado de recolectar toda la información necesaria, y almacenarla en la base de datos para su posterior recolección por parte de los otros módulos.
Para realizar todo esto, es necesario modificar el código del proyecto en el cual se integre MC-Spy, es decir, para el cual se pretende obtener información a través de sus estadísticas. El porqué de la necesidad de la modificación del código del proyecto padre está claro, solamente desde allí se podrá saber qué operativa se ha ejecutado, además de otros datos como por ejemplo cuando, quien y con qué objetivo y final se ha realizado. Cualquier otra forma de querer obtener esta información sería mucho más complicada e innecesaria.

Habrá pues que identificar los puntos del código donde será necesario insertar las llamadas oportunas (explicadas posteriormente en el Diseño, Anexo B). Para ello es necesario estudiar con cierto detalle la implementación de dicho proyecto, ya que es imprescindible acertar con esta ubicación, ya que de ello depende que con posterioridad los datos que muestren las estadísticas de MC-Spy, sean datos fiables y se refieran a información verídica.
Por supuesto, el lugar donde se va a almacenar toda esta información, va a ser en base de datos. Como se prevé que el número de inserciones puede ser alto (este número dependerá de los usuarios simultáneos que haya en la banca, o en cualquier otro sistema donde esté integrado MC-Spy, en un momento dado), habrá que diseñar algún modo de realizar las inserciones sin que por ello se penalice el funcionamiento normal del resto de la aplicación.

A.1.2.2 Consulta de estadísticas

Este módulo será el encargado de mostrar a los administradores de la banca, los datos que necesiten para comprobar su correcto funcionamiento, o para argumentar posibles decisiones sobre su uso por parte de los usuarios.
El módulo será básicamente una aplicación web, la cual, mediante la solicitud de una serie de datos que se le pedirán al usuario, mostrará unas gráficas y unos datos, para lo cual utilizará la información previamente almacenada por el módulo de Recolección de datos.
En este punto es interesante resaltar que también se hizo un análisis de cuáles eran las estadísticas que se iban a tener que mostrar en la aplicación. Para ello nos pusimos en la piel de un administrador de una banca electrónica, el cual tenía que poder demostrar a sus superiores cualquier hecho que se produzca en ella, así como presentarles informes regulares sobre su uso por parte de los clientes. Es por ello por lo que se realizó un listado de todas las estadísticas que resultaría interesante que MC-Spy tratase, el cual mostramos a continuación:

- Número y detalle de los establecimientos de sesión (entradas a la banca) en un periodo de tiempo:
    - Por un cliente.
    - Desde una IP. ***
    - Desde un región, país... ***
- Número y detalle de consultas de saldo realizadas en un intervalo de tiempo:
    - Por un cliente.
    - Sobre una determinada cuenta.
    - Por una modalidad.
- Número y detalle de consultas de movimientos realizadas en un intervalo de tiempo:
    - Por un cliente.

- Sobre una determinada cuenta.

- Intervalo de tiempo medio por el que se realizan consultas de movimientos.
- Número y detalle de consultas de valores realizadas en un intervalo de tiempo:
  - Por un cliente.
  - Sobre una determinada cuenta.
- Número y detalle de traspasos entre cuentas propias realizados en un intervalo de tiempo:
  - Por un cliente.
  - Desde una determinada cuenta.
  - Teniendo como destino una determinada cuenta.
  - Con un importe máximo.
  - Con un importe mínimo.
- Importe medio en los traspasos entre cuentas propias.
- Número y detalle de cambios de moneda en un intervalo de tiempo:
  - Por un cliente.
  - Desde una determinada cuenta.
  - Teniendo como destino una determinada cuenta.
  - Con un importe máximo.
  - Con un importe mínimo.
- Importe medio en los cambios de moneda realizados en un intervalo de tiempo.
- Número y detalle de traspasos a cuentas ajenas realizadas en un intervalo de tiempo:
  - Por un cliente.
  - Desde una determinada cuenta.
  - Teniendo como destino una determinada cuenta.
  - Con un importe máximo.
  - Con un importe mínimo.
- Importe medio en los traspasos a cuentas ajenas realizados en un intervalo de tiempo.
- Número y detalle de transferencias realizadas en un intervalo de tiempo:
  - Por un cliente.
  - Desde una determinada cuenta.
  - Teniendo como destino una determinada cuenta.
  - Con un importe máximo.
  - Con un importe mínimo.

- Importe medio en las transferencias realizadas en un intervalo de tiempo.
- Número y detalle de consultas de traspasos y transferencias realizadas en un intervalo de tiempo:
    - Por un cliente.
- Número y detalle de las consultas de detalles de traspasos y transferencias realizadas en un intervalo de tiempo:
    - Por un cliente.
- Número y detalle de consultas de información general realizadas en un intervalo de tiempo:
    - Por un cliente.
- Número y detalle de los créditos documentarios realizados en un periodo de tiempo:
    - Por un cliente.
    - Por una cuenta.
    - Por importe máximo.
    - Por importe mínimo.
- Importe medio en los créditos documentarios realizados en un intervalo de tiempo.
- Número y detalle de las asignaciones de referencias realizadas en un periodo de tiempo:
    - Por un cliente.
- Número y detalle de los cambios de clave de acceso realizadas en un periodo de tiempo:
    - Por un cliente.
- Número y detalle de los cambios de clave de acceso realizadas en un periodo de tiempo:
    - Por un cliente.
- Número y detalle de los mensajes enviados a la entidad en un periodo de tiempo:
    - Por un cliente.
    - Para un destinatario concreto.
- Número y detalle de los riesgos de cedentes (totales y en detalle) consultados en un intervalo de tiempo:
    - Por un cliente.
    - Por una cuenta.
- Número y detalle de las consultas de efectos en un intervalo de tiempo:
    - Por un cliente.

- Por una cuenta.

- Número y detalle de consultas de ficheros realizadas en un intervalo de tiempo:
  - Por un cliente.
- Número y detalle de las consultas de detalles de ficheros realizadas en un intervalo de tiempo:
  - Por un cliente.
- Número y detalle de las personalizaciones del perfil de usuario realizadas en un periodo de tiempo:
  - Por un cliente.
- Número y detalle de las firmas realizadas sobre una transferencia en un periodo de tiempo:
  - Por un cliente.
- Número y detalle de las firmas realizadas sobre un fichero en un periodo de tiempo:
  - Por un cliente.
- Número y detalle de los borrados de firmas realizados sobre una transferencia en un periodo de tiempo:
  - Por un cliente.
- Número y detalle de los borrados de firmas realizados sobre un fichero en un periodo de tiempo:
  - Por un cliente.
- Número y detalle de las recepciones realizadas por la banca en un periodo de tiempo:
  - De un cliente.
- Número y detalle de las ayudas consultadas en un periodo de tiempo:
  - Por un cliente.
- Cantidad de dinero movido en la banca en un periodo de tiempo.
- Tiempos medios de ejecución de cada operativa.
- Gráfico indicando los porcentajes de qué operativas ejecutan más los clientes.
- Gráfico detallando los porcentajes de cada una de las modalidades de consultas de saldo.
- Gráfico con los intervalos mas utilizados en las consultas de movimientos.
- Gráfico con los intervalos de importes realizados en los traspasos entre cuentas propias.
- Gráfico con las monedas con las que se hacen los traspasos entre cuentas propias.
- Gráfico con los intervalos de importes realizados en los cambios de moneda.

- Gráfico con las monedas con las que se hacen los cambios de moneda.
- Gráfico con los intervalos de importes realizados en los traspasos a cuentas ajenas.
- Gráfico con las monedas con las que se hacen los traspasos a cuentas ajenas.
- Gráfico con los intervalos de importes realizados en las transferencias.
- Gráfico con las monedas con las que se hacen las transferencias.
- Gráfico comparativo entre varios meses de la cantidad de dinero movido en transferencias y traspasos.
- Gráficos con la procedencia de los accesos a la banca (IP´s, países,...).
- Gráfico con el número de visitas por día en el último mes(o semana,...).
- Gráfico con el número de visitas que hace un cliente en un día, una semana, ...
- Gráfico de la actividad de la banca en los distintos días de la semana.
- Gráfico de la actividad de la banca en las distintas horas del día.
- Gráfico con el número de operativas realizadas por cliente en una sesión.
- Gráfico con las operativas que han dado error al ejecutarse.
- Gráfico comparativo con los tiempos medios de ejecución de cada una de las operativas.
- Gráfico comparativo del dinero movido en la banca en dos periodos de tiempo.

De todas formas, el diseño de la aplicación se realizará de tal forma, que este listado dependerá exclusivamente del cliente, y de la información que los administradores de la banca consideren necesario que hay que mostrar.

A.1.2.3 Árbol de sesiones

Este módulo será el encargado de mostrar la información relativa a la actividad generada por un usuario a lo largo de una sesión en la banca electrónica.
Por medio de un árbol que mostrará gráficamente la información requerida, se podrá seguir fácilmente todo el recorrido que el cliente haya hecho a través de la aplicación bancaria, conociendo si en alguna operación ha ocurrido algún error, y en el caso de que así sea, cual ha podido ser el motivo de dicho fallo.
Para ello habrá que dotar al modelo de algún sistema que establezca el nexo entre todas las operativas que ejecuta un usuario, para que después, mediante la aplicación gráfica, poder unirlas, asociándolas al mismo usuario y a la misma sesión abierta en la banca por él.

A.1.2.3 Previsiones

Este módulo va muy íntimamente ligado al de Consulta de estadísticas, ya que el funcionamiento es muy similar al de Consulta de estadísticas, y aunque en este punto de la construcción del proyecto podría haberse integrado perfectamente en el módulo citado, ya prevemos que las previsiones van a requerir un tratamiento muy distinto al de las estadísticas normales, sobre todo a la hora del procesado de los datos.
El sistema gráfico de las previsiones es semejante al de las estadísticas, ya que se mostrarán en gráficas (y en forma de texto si así se cree conveniente), distintas previsiones de cómo se cree que van a comportarse los datos para un plazo medio-corto.
Como decimos, ahora quizá no se vea aun la distinción, pero en posteriores estudios se verá su necesidad.

A.1.3 Casos de uso

Dentro de la aplicación MC-Spy identificamos una serie de Casos de Uso que describen las posibles interacciones del usuario con el sistema.
Para la descripción de estos casos de uso se ha utilizado UML.

A.1.3.1 Uso de una operativa de la banca electrónica

Se trata de la operación que llevará a cabo un usuario de la banca electrónica cuando realice cualquier operativa dentro de la banca electrónica, desde hacer login, hasta desconectarse de ella, pasando por envió de transferencias, consulta de saldos o cualquier otra acción que la banca permita a un cliente.

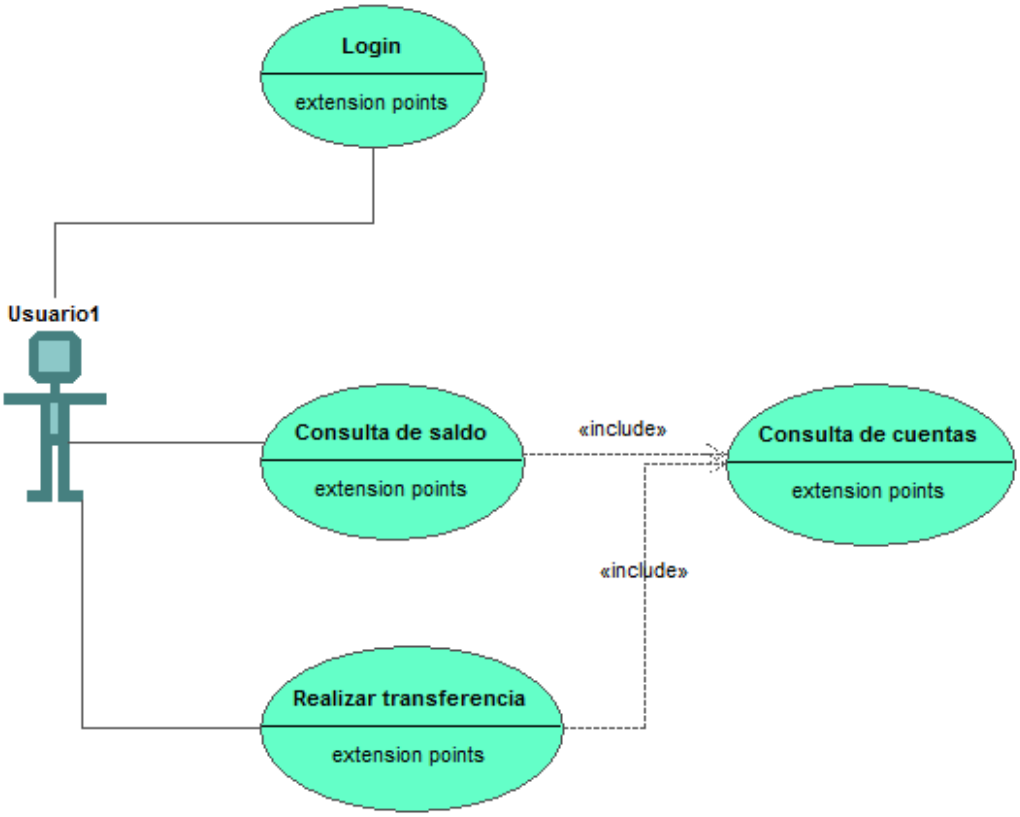


Figura 7- Caso de uso de una operativa en la banca

Todas estas acciones, tengan el resultado que tengan, quedarán registradas en la base de datos.

CASO DE USO	**Realizar una transferencia en la banca electrónica**	
Objetivo	Realizar un traspaso de dinero entre 2 cuentas, ya sean propias, de la misma o de otra entidad	
Precondiciones	Haberse logado dentro de la banca electrónica. Tener alguna cuenta desde la que poder hacer el traspaso, y que tenga saldo suficiente.	
Condición de fin con éxito	Se añade una orden de transferencia, para que el proceso batch de la banca lo ejecute cuando corresponda.	
Condición de fin erróneo	No se realiza la transferencia	
Actores principales	Usuario	
Actores secundarios	Ninguno	
Disparador	Petición del usuario	
DESCRIPCIÓN	**Paso**	**Acción**
	1	El usuario accede al módulo de transferencias dentro de la banca

	2	El sistema muestra sus cuentas disponibles
	3	El usuario elige la cuenta, la cantidad y demás datos necesarios para completarla.
	4	El usuario clicka en el icono de ejecutar
	5	El sistema añade la orden con la transferencia
	6	El sistema muestra un resumen con la transferencia realizada por el usuario

A continuación se muestra el diagrama de secuencia asociado a este caso de uso, en concreto de una operación de transferencia, para mostrar un ejemplo de una operativa medianamente completa, y mostrar que la banca se comunica con MC-Spy, pero no al revés:

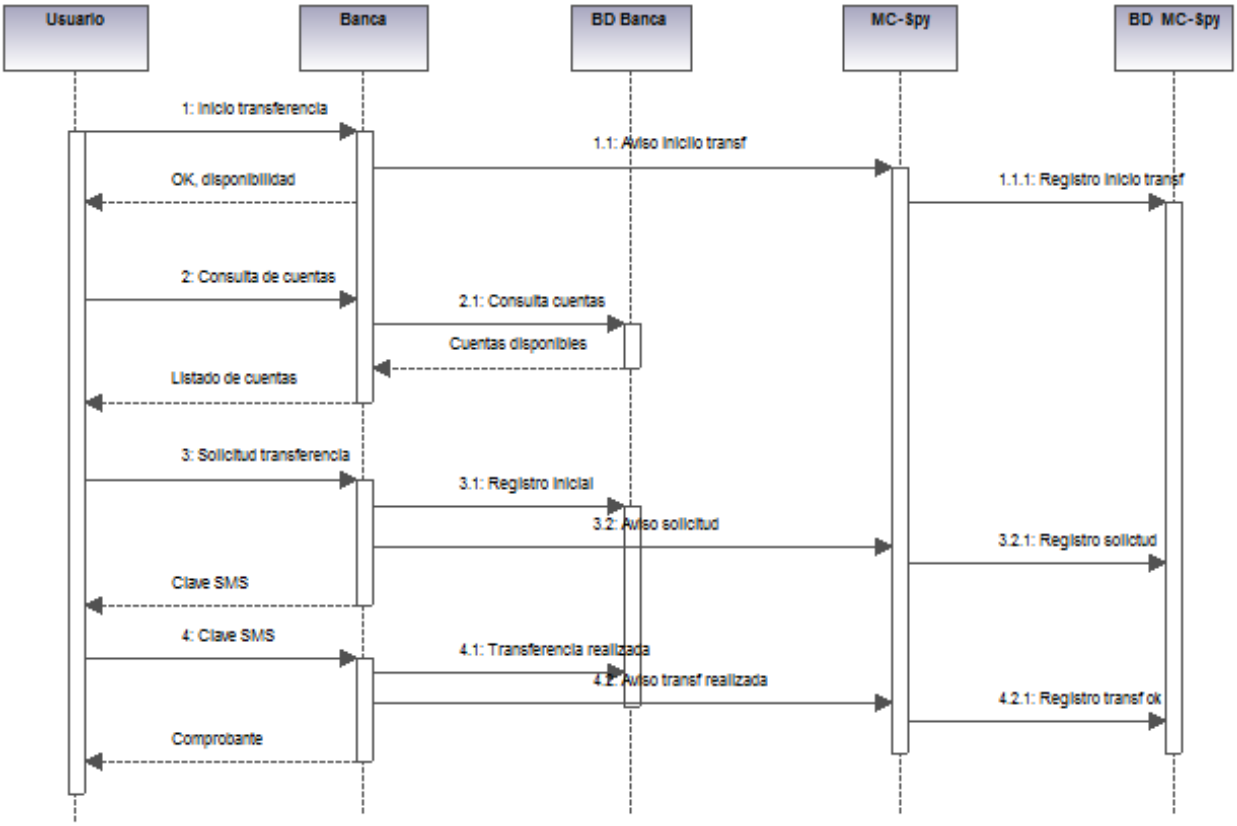


Figura 8- Diagrama de secuencia de una transferencia

Como se puede ver en esta secuencia, el registro de operaciones en MC-Spy es totalmente transparente para el usuario, ya que el usuario no se apercibe que se

producen dichos registros de sus acciones, siendo consciente únicamente de su comunicación con la banca.

A.1.3.2 Consulta de una estadística

En este caso se trata de la operación que llevará a cabo un administrador de la banca electrónica, cuando desee consultar algún dato referente a su uso por parte de los clientes.

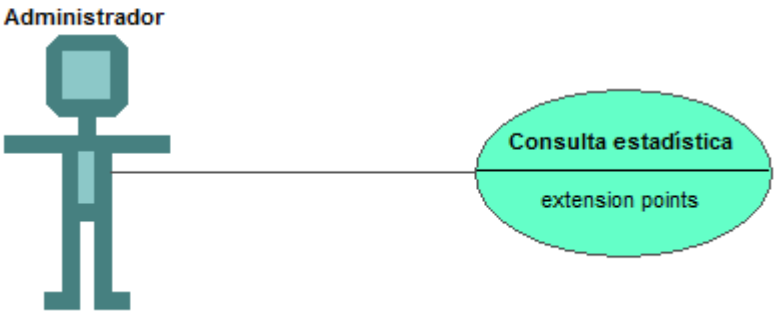


Figura 9- Caso de uso Consulta estadística

CASO DE USO	**Realizar una consulta de una estadística en MC-Spy**	
Objetivo	Consultar datos sobre alguna de las posibles estadísticas que MC-Spy ofrece a los administradores	
Precondiciones	Haberse logado dentro de MC-Spy.	
Condición de fin con éxito	Se muestra un gráfico, o una serie de datos con la estadística seleccionada	
Condición de fin erróneo	No se mostrará nada.	
Actores principales	Administrador	
Actores secundarios	Ninguno	
Disparador	Petición del administrador	
DESCRIPCIÓN	**Paso**	**Acción**
	1	El administrador accede al módulo de estadísticas dentro de MC-Spy
	2	El sistema muestra las estadísticas disponibles
	3	El administrador elige los datos necesarios

	4	El sistema muestra la estadística

A continuación se muestra el diagrama de secuencia asociado a este caso de uso:

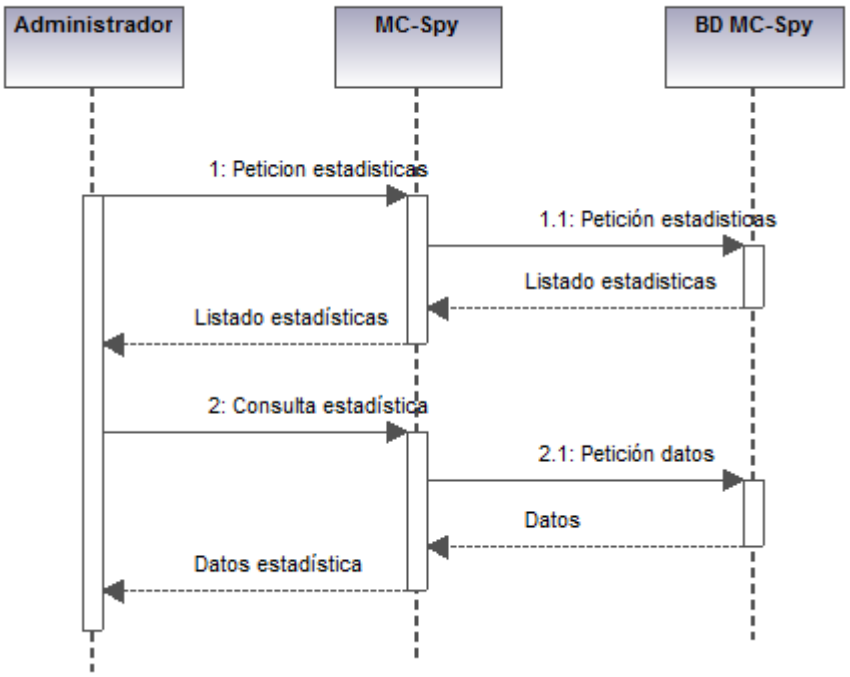


Figura 10- Diagrama secuencia Consulta estadística

A.1.3.3 Consulta del árbol de sesiones

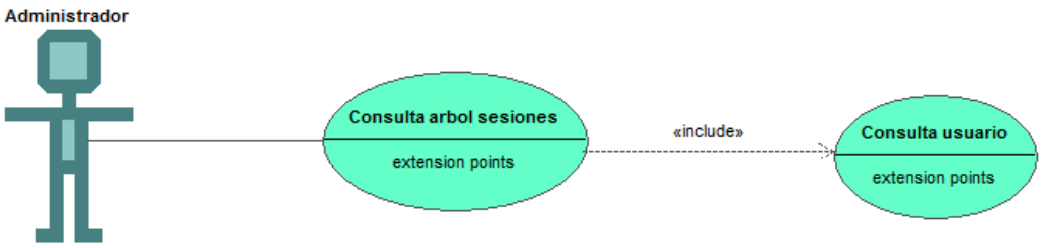


Figura 11- Caso de uso Consulta árbol de sesiones

CASO DE USO	**Realizar una consulta del árbol de sesiones en MC-Spy**	
Objetivo	Consultar datos sobre algún usuario a través del árbol de sesiones	
Precondiciones	Haberse logado dentro de MC-Spy.	
Condición de fin con éxito	Se muestran datos sobre la actividad del usuario dentro de la banca electrónica	
Condición de fin erróneo	No se mostrará nada.	
Actores principales	Administrador	
Actores secundarios	Ninguno	
Disparador	Petición del administrador	
DESCRIPCIÓN	**Paso**	**Acción**
	1	El administrador accede al módulo de Árbol de sesiones dentro de MC-Spy
	2	El sistema muestra la solicitud de datos necesarios
	3	El administrador elige los datos necesarios
	4	El sistema muestra el árbol

A continuación se muestra el diagrama de secuencia asociado a este caso de uso:

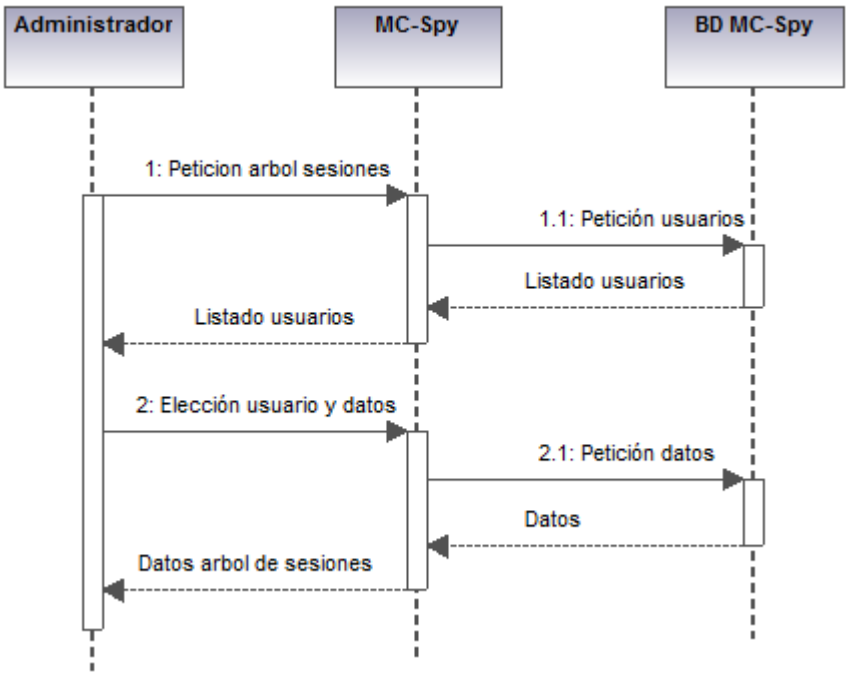


Figura 12- Diagrama de secuencia Consulta árbol de sesiones

A.1.4 Estudio teórico de las predicciones

Para la parte de previsiones de la Administración de MC-Spy, se realizó un estudio teórico sobre las series de tiempo, en concreto sobre las relacionadas con la economía, como es nuestro caso. Tras analizar la naturaleza de la información sobre la que se va a trabajar en banca (series de tiempo, es decir, un conjunto de datos a lo largo de un intervalo de tiempo), se optó por trabajar con 4 tipos distintos de series, según la existencia o no de la *tendencia* (variación continuada a lo largo de un gran período de tiempo) y la *estacionalidad* (conjunto de fluctuaciones intraanuales que se repiten más o menos regularmente todos los años):

- Serie tipo 1: Series sin tendencia y sin estacionalidad.
- Serie tipo 2: Series sin tendencia y con estacionalidad.
- Serie tipo 3: Series con tendencia y sin estacionalidad.
- Serie tipo 4: Series con tendencia y con estacionalidad.

Como casos más representativos, a continuación vamos a mostrar dos gráficos con series, el primero de ellos de una serie con estacionalidad y el segundo de una serie con tendencia.

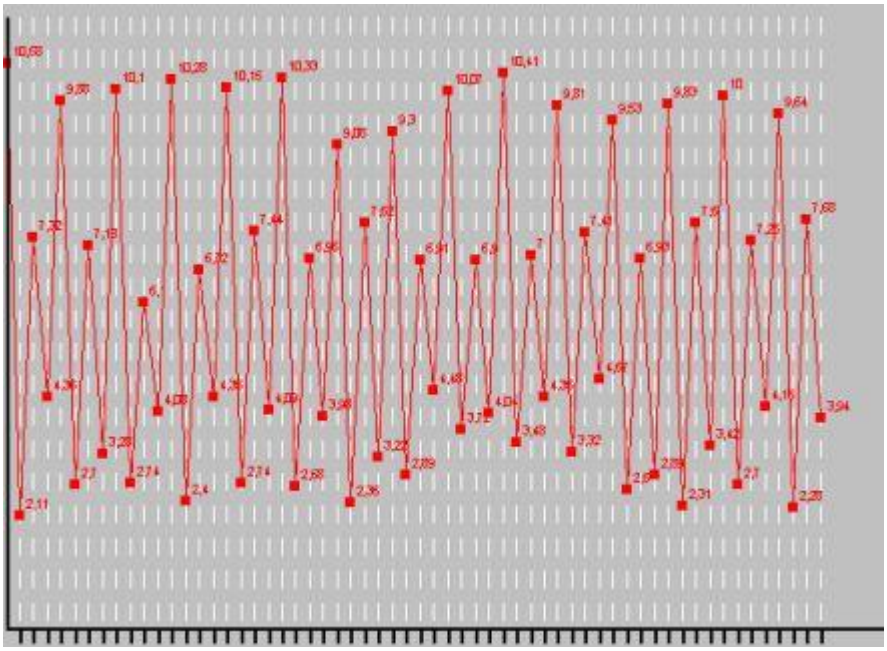


Figura 13- Ejemplo de serie con estacionalidad

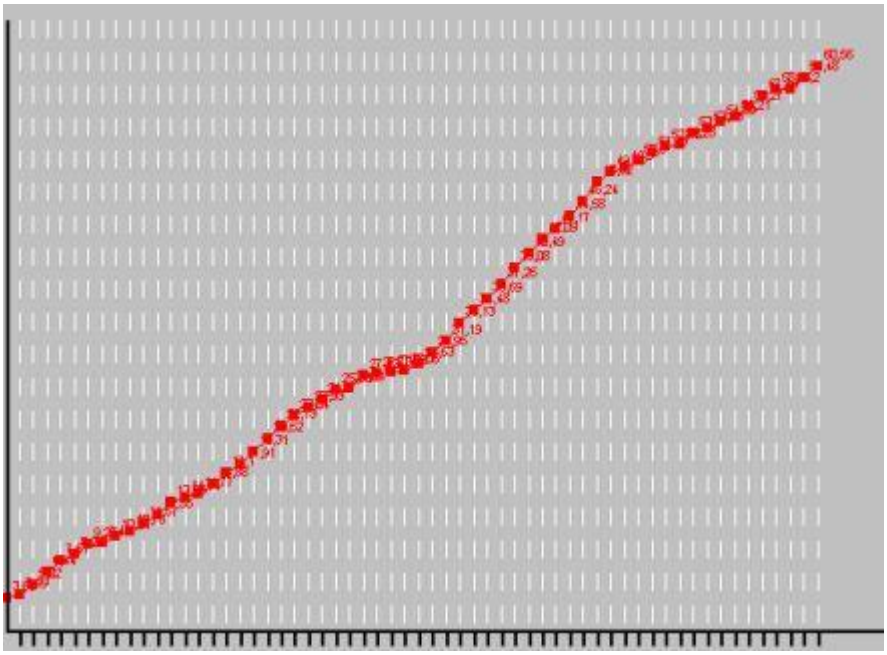


Figura 14- Ejemplo de serie con tendencia

Una vez conocidos los posibles tipos de series de tiempo sobre los que se puede trabajar (al menos con los que vamos a trabajar aquí, ya que habría otras variables más concretas

y complicadas, que dado el tiempo que se tiene y que no es el objetivo propio de este proyecto, no se van a estudiar ni a tratar para no complicar el proyecto ni poner en riesgo su realización), hay que saber detectar de qué tipo es una serie observando sus datos (ya que aunque gráficamente lo podamos ver muy intuitivamente, es necesario saberlo teniendo únicamente la serie de los datos almacenada "digitalmente"). Para ello se utilizarán los contrastes de Daniel (para conocer la tendencia) y el de Kruskal-Wallis (para conocer la estacionalidad). Dichos contrastes comprueban, mediante un conjunto de cálculos realizados sobre los elementos de la serie, la existencia de las características anteriormente mencionadas. Para conocer pues el tipo de serie, será necesaria la creación de un algoritmo que aplique estos dos contrastes para averiguarlo.

Tras conocer de qué tipo es una serie, hay que aplicar también un algoritmo para por fin hallar las previsiones que buscábamos. Para cada uno de los tipos, existe un algoritmo de predicción:

- Serie de tipo 1: Método de alisado exponencial. Realiza la predicción mediante una suma ponderada de todos los valores previos de la serie.
- Serie de tipo 2: Método de medias estacionales. Realiza la predicción para cada período a partir de los períodos con idéntico componente estacional.
- Serie de tipo 3: Método de alisado exponencial lineal de Holt. La predicción se basa en una estimación de la tendencia y la pendiente para cada período muestral, utilizando para ello todos los valores previos de la serie.
- Serie de tipo 4: Método de alisado exponencial de Holt-Winters. Se basa en la estimación a lo largo de todos los períodos muestrales de la tendencia, la pendiente y el componente estacional mediante unas ecuaciones de actualización.

A.1.5 Herramientas de desarrollo y auxiliares

Para el desarrollo del proyecto y para todo lo que se pueda requerir, se emplearán los siguientes productos, utilizados siempre (cuando así era requerido por el fabricante) bajo licencia:

- Edición de textos: Microsoft Word.
- Presentaciones: Microsoft PowerPoint.
- Herramienta acceso a BD: TOAD.
- Herramienta CASE: Altova.
- Navegadores: Internet Explorer, Firefox.
- Edición JSPs: Scite.
- Herramienta desarrollo y compilación: Eclipse.

Anexo B. Diseño

A partir del contenido del Análisis del sistema, el presente capítulo pasa a describir las decisiones de Diseño que han sido adoptadas.

B.1.1 Diseño lógico del sistema

En este apartado se va a realizar el modelado del sistema sin tener en cuenta el lenguaje de programación ni la base de datos utilizados. El diseño lógico es, por lo tanto, independiente de dichos elementos y permanecería inamovible en caso de decidir modificar los elementos del diseño físico (diseño que se cimenta en el lógico, y en el que si se tienen en cuenta consideraciones dependientes del lenguaje y otros aspectos tecnológicos utilizados. Se podrá ver con más detalle en el siguiente apartado de este mismo capítulo).
De todas formas, la línea que separa ambas partes del diseño es muy fina, y suele ser inexistente, ya que aun si así quererlo, durante el diseño lógico se van a tener en cuenta consideraciones de lenguaje o arquitecturales, que debieran ser tomadas en consideración más adelante.
Además, en proyectos pequeños quizá no tenga excesivo sentido realizar dicha distinción, ya que el modelado lógico puede resultar muy escaso.
Aun así, se ha optado por intentar delimitar la separación entre ambos modelados, para ayudar a personas que no pueden tener conocimientos de JAVA, a saber el Cómo se ha realizado el proyecto.

En este modelo lógico, lo que se va a realizar es una extensión del análisis de requisitos del sistema, puesto que lo que se hace es diseñar el sistema y las partes que lo forman de modo que se adapte a los requisitos especificados para MC-Spy. Mientras que en el análisis se mostraba el *qué* hay que realizar, en el diseño lógico se muestra el *cómo* hay que desarrollarlo.

Para explicar el modelado lógico, se van a utilizar los diagramas de casos de uso, ya sean horizontales (para la distribución de procesos) o verticales (para descubrir la funcionalidad del mismo). En concreto van a ser tres diagramas los que se van a mostrar, los dos primeros horizontales, y el último vertical:

- El Modelo de Subsistemas, que presenta de forma somera los distintos subsistemas identificados y las relaciones de uso existentes entre ellos, sin entrar en detalles de los módulos que forma cada subsistema.
- El Modelo de Módulos de los Subsistemas, que muestra un desarrollo lógico más detallado en el que se muestran los módulos o casos de uso horizontales más importantes de cada subsistema, mostrando las relaciones de uso existentes entre sí.

- El Modelo Funcional, que muestra en el ámbito conceptual las distintas funcionalidades disponibles para el usuario final y las relaciones (o usos) existentes entre ellas.

B1.1.1 Modelo lógico de subsistemas

La identificación de cada subsistema, su distribución y las relaciones existentes con el resto de subsistemas resultó bastante clara y sencilla desde el primer momento en que se planteó.
Como puede verse en el siguiente gráfico, MC-Spy consta de dos subsistemas. Cada una de las instancias que se ejecutan de cada subsistema está pensada para ejecutarse en un nodo, aunque debería darse la posibilidad de que en un mismo nodo se ejecuten varios procesos distintos.
Los dos subsistemas identificados son los siguientes:

- Recolector de datos, que se encarga de recoger la información de las operativas ejecutadas en la Aplicación Web monitorizada por MC-Spy y almacenarla en base de datos.
- Cliente de consultas, que es el proceso que ejecutan los usuarios del sistema –los administradores de la banca-. El cliente accederá a los datos que previamente ha almacenado el recolector.

Esta distribución de procesos se modela de forma general en la siguiente ilustración, en la cual se ha puesto como ejemplo un sistema integrado por tres usuarios que van a utilizar el recolector de datos a través de MC-Server, y dos administradores que acceden para consultar los datos.

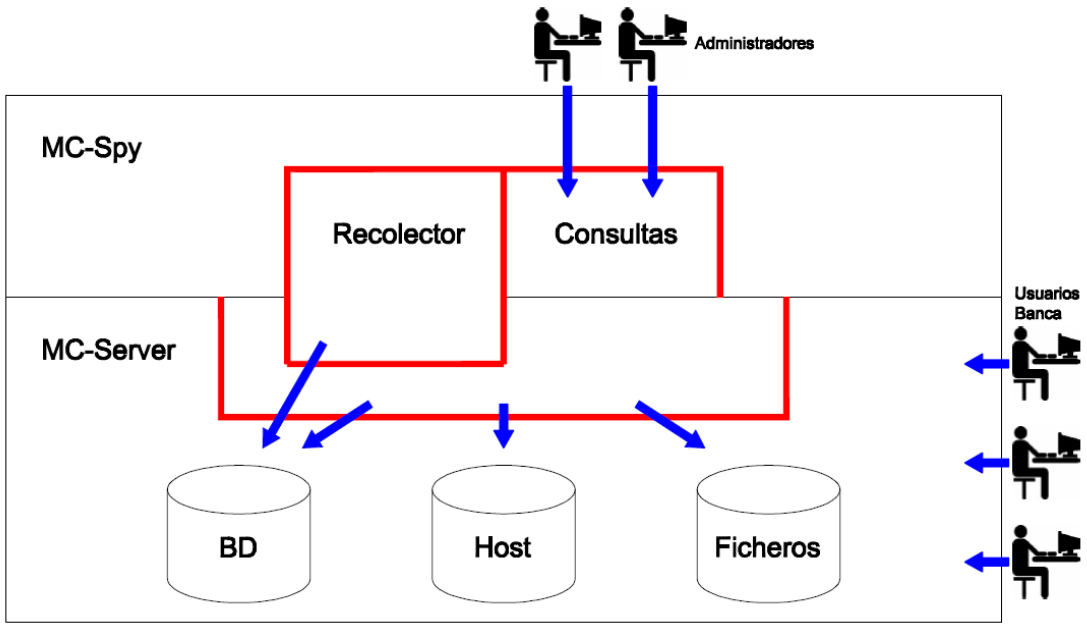


Figura 15- Modelo lógico

Tal y como se deduce del Análisis de Requisitos, y como muestra en la ilustración anterior, el sistema que se está diseñando será utilizado por los administradores, los cuales utilizarán el Cliente de consultas para acceder a la información almacenada en el sistema.

B1.1.2 Modelo lógico de módulos de los subsistemas

Como puede observarse en el modelo de la ilustración que a continuación se muestra, los dos subsistemas han sido diseñados de tal forma que tienen los dos un módulo de acceso a base de datos, aunque las semejanzas entre ellos allí se acaban.
Como se puede apreciar, se han identificado dos actores distintos que pueden usar el sistema:

- Usuario de MC-Server. Es el usuario de la banca electrónica, que genera información que se almacenará en la base de datos.
- Administrador de la banca electrónica. Es quien va a consumir dichos datos, a través de las peticiones al módulo de interfaz gráfica del Cliente de consultas.

En el subsistema Recolector de datos se han identificado los siguientes módulos, desde el punto de vista horizontal:

- Módulo de acceso a base de datos. En este caso se trata de la inserción de los datos.
- Módulo de gestión temporal de datos. Se encargará de gestionar la inserción en la base de datos de forma que no cause perjuicio en el resto de la aplicación. De este modo se colabora a garantizar el requisito de fiabilidad y estabilidad.
- Módulo de integración con MC-Server: Es el módulo usado para conectarse con MC-Server y encargarse de recibir toda la información que este le facilite.

Figura 16- Subsistema Recolector de datos

El subsistema Cliente de consultas, se encarga de mostrar de forma gráfica toda la información almacenada por el subsistema anterior, por supuesto organizada toda ella según peticiones de los usuarios. En este subsistema se han identificado los siguientes módulos, desde el punto de vista horizontal:

- Módulo de acceso a base de datos. Es el módulo usado para conseguir toda la información que soliciten los usuarios.
- Módulo de interfaz gráfica. Es el módulo usado por los administradores de la banca (ellos serán generalmente los usuarios de la aplicación), y que se encarga de interactuar con el módulo de acceso a base de datos. Este módulo, como su propio nombre indica, presenta de una forma visual la información solicitada.

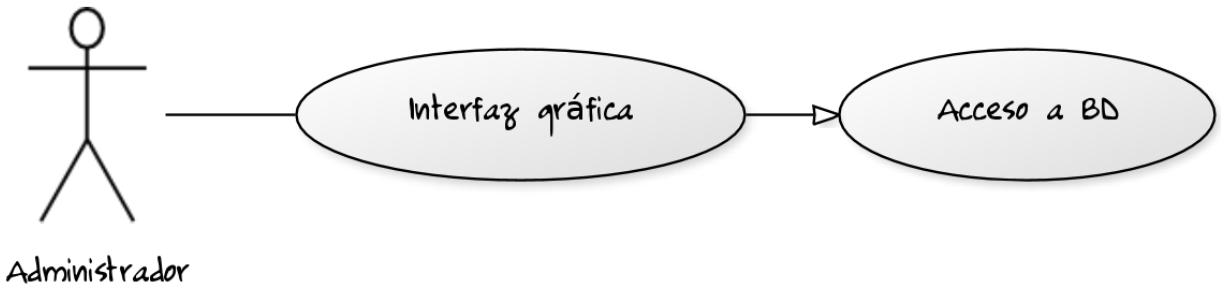


Figura 17- Subsistema Cliente de consultas

B.1.2 Diseño físico del sistema

El Diseño Físico consiste en una especificación detallada del sistema resultante del Diseño Lógico, teniendo en cuenta el lenguaje de programación y las bibliotecas que se van a utilizar para el acceso a datos.
Como se especifica en el capítulo 2.2 Requisitos no funcionales, el proyecto está desarrollado utilizando Java (JDK 1.3, 1.5,…) como lenguaje de programación y Oracle como gestor de base de datos.

La generación de estos modelos físicos del sistema es simple, ya que se trata de una evolución de los modelos lógicos horizontales del sistema presentados en apartados anteriores.
Para cada uno de los módulos de los subsistemas identificados, se ha especificado una clase principal. Cada una de estas clases está asociada con otras clases principales

siguiendo los usos entre casos especificados en los diagramas anteriores de Casos de Uso de los que evoluciona este diagrama de clases.

B.1.2.1 Modelo físico general del sistema

El modelo físico inicial se muestra en la siguiente ilustración. Este modelo no presenta ningún detalle adicional del sistema.
Como se ha introducido antes, cada una de las clases que aparecen en este diagrama es la clase principal de cada uno de los módulos identificados para cada subsistema. La especificación detallada de los métodos de estas clases no aparece en este modelo, ya que aún no han sido definidos en este punto del diseño físico. Tampoco se especifican el resto de las clases utilizadas en cada uno de los módulos –clases auxiliares- ni las relaciones existentes entre ellas.

Como puede observarse en la imagen, se nombra para cada uno de los módulos una clase que se podría llamar como "principal" o localizadora. En algunos casos no es posible identificar a una sola clase para tal fin, con lo que se nombra el paquete en el cual residen las clases que cometen dicha labor.

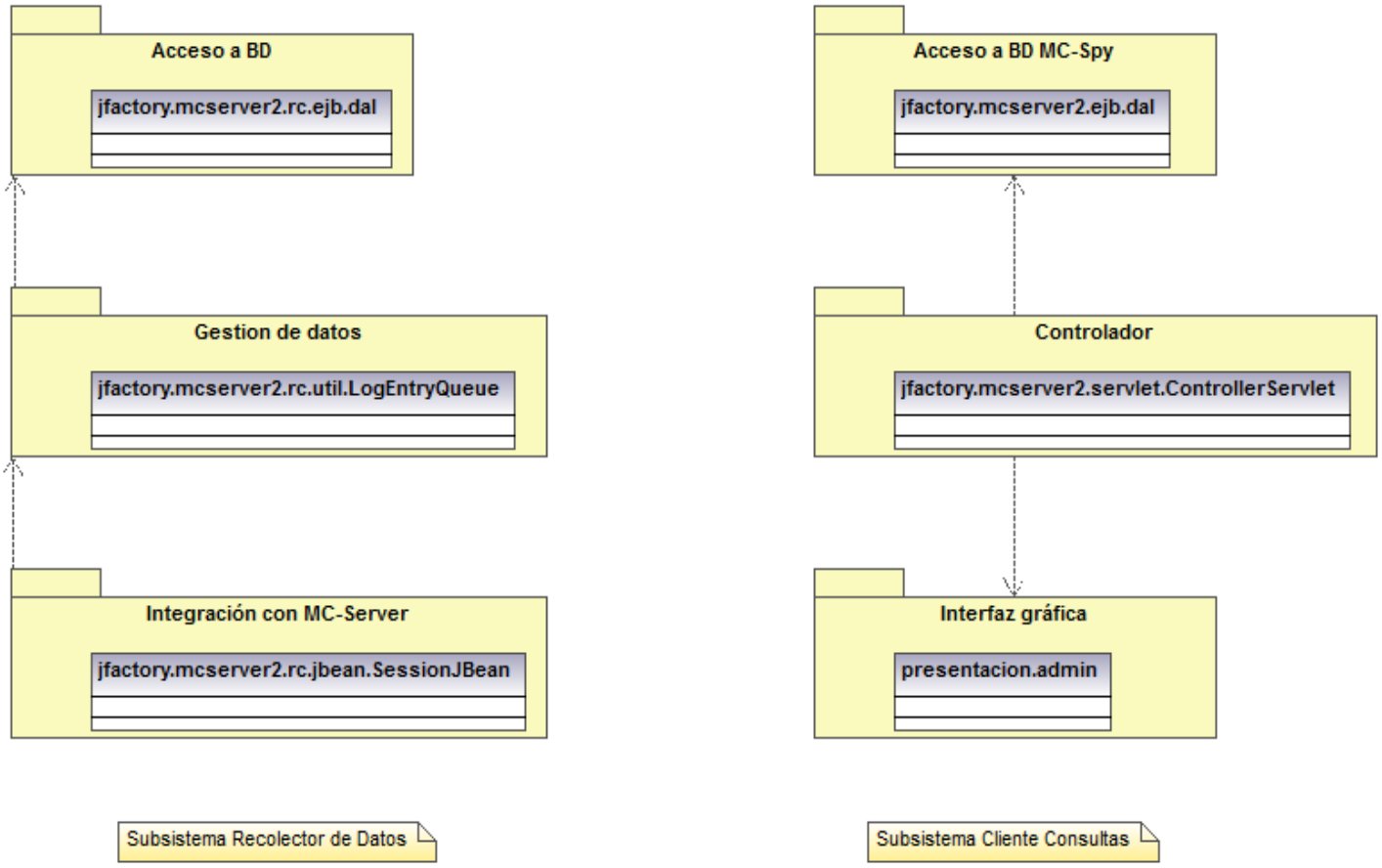


Figura 18- Modelo físico

El sistema está dividido en dos subsistemas, los cuales representan procesos independientes:

- Por un lado tenemos el subsistema Recolector de datos, que se encarga de interaccionar con MC-Server y recoger los datos que esté le facilita para su posterior uso.
- Finalmente tenemos el subsistema de Cliente de consultas que se encarga de interrogar al sistema para obtener la información que se le solicite.

Cada uno de estos subsistemas será descrito en los puntos siguientes de este mismo apartado.

B.1.2.1.1 Subsistema Recolector de datos

El subsistema Recolector de Datos está compuesto por tres clases o paquetes principales, cada una de ellas equivalente a un módulo del diagrama de casos de uso que se muestra anteriormente.

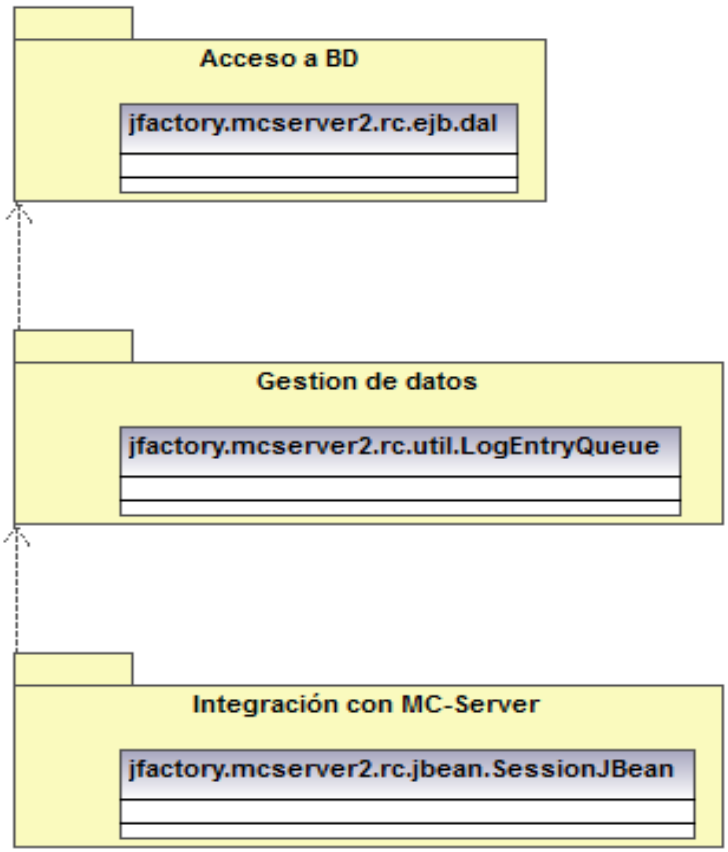


Figura 19- Subsistema recolector de datos

La clase LogEntryQueue (además de otras clases auxiliares que ésta utiliza) es la encargada de implementar el sistema de gestión de recepción de información desde MC-Server y su priorización y ordenamiento para su posterior almacenamiento en base de datos.

El paquete de Acceso a BD se encargará precisamente de este almacenamiento en la base de datos.

La clase SessionJBean centralizará la comunicación directa con MC-Server, recibiendo de este sistema todas sus invocaciones, las cuales contienen la información a almacenar en la base de datos.

B.1.2.1.2 Subsistema Cliente de consultas

Al ser el subsistema del Cliente de consultas una aplicación web, el modelo físico y sus subsistemas se pueden identificar con cada uno de los componentes del patrón Model-View-Controller, que como ya se ha dicho anteriormente en este documento, es el que se iba a utilizar para la implementación de este PFC. Así pues, vamos en un primer lugar a explicar brevemente la filosofía de este patrón.

B1.2.1.1.1 Patrón Model-View-Controller

El patrón Model-View-Controller (MVC ó Model 2) merece un especial interés puesto que se trata de una pieza clave en la arquitectura física de la aplicación, por lo que se describirá brevemente a continuación.
El siguiente esquema muestra los componentes básicos del patrón MVC que se ha aplicado a la arquitectura propuesta para el sistema:

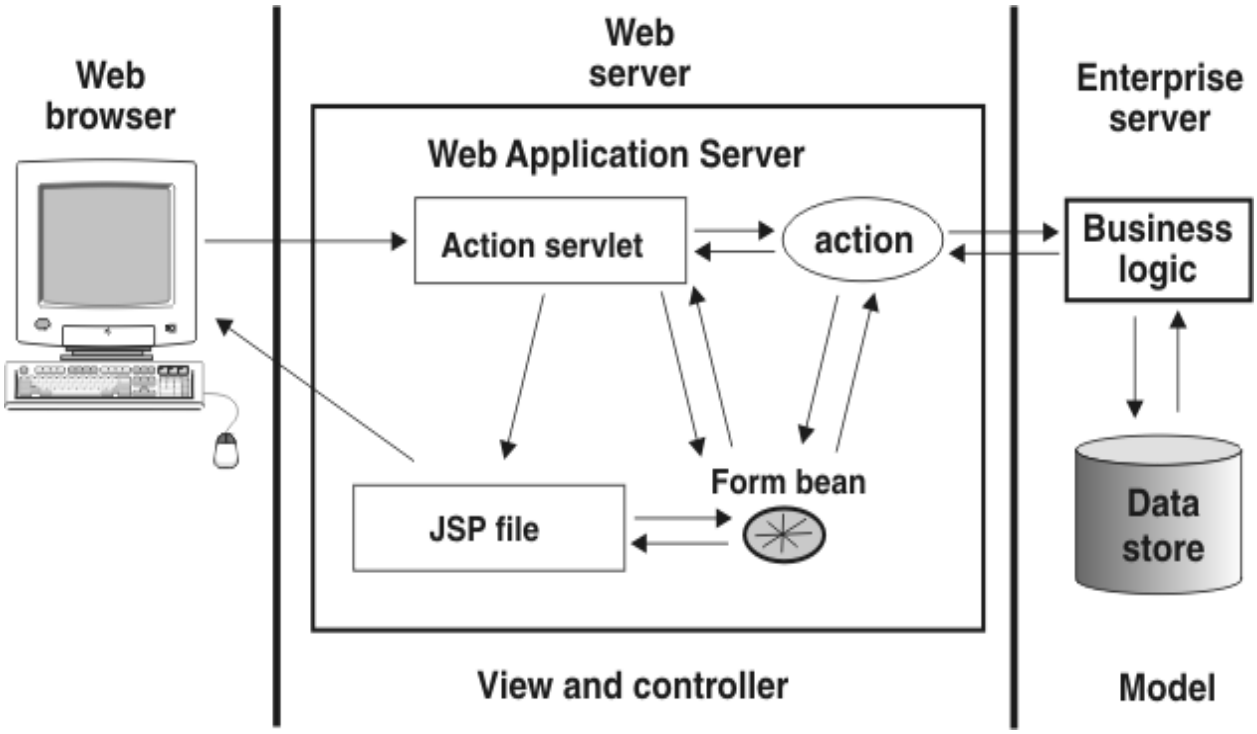


Figura 20- Esquema patrón MVC

La filosofía MVC consiste básicamente en separar claramente estos tres conceptos:

- **Model**: equivalente a los niveles de lógica de negocio y acceso a datos. En este nivel se implementa el acceso y modificación de datos de negocio. Como se ha introducido anteriormente, se unifican físicamente estas dos capas, aunque a nivel lógico continúen perfectamente delimitadas las responsabilidades de cada una de ellas.
- **View***:* equivalente al nivel de presentación. Se implementa siempre en un entorno web a través de JSP o de XSL. Nunca podrá un servlet enviar información de presentación al navegador de forma directa; únicamente podrá hacerlo a través de un JSP o de una plantilla XSL (que transforma la información XML).
- **Controller***:* equivalente al nivel de control. Se implementa siempre a través de un servlet o un conjunto de servlets. Un controlador recibe una petición desde el cliente, invoca a la lógica de negocio y elige un *view* (JSP) apropiado para que éste realice el formateo de la información que será presentada al cliente. El servlet utiliza uno o varios *JavaBean* (JBean), que serán los encargados de interactuar con los componentes de negocio (EJB) siguiendo así el patrón de diseño *"Command"*.

El uso del patrón MVC dentro de una arquitectura J2EE, y usando (pero no abusando) componentes EJB, garantiza la **flexibilidad** del sistema y la interacción con múltiples actores, tanto en lo que se refiere a los clientes del sistema (navegador, móvil, PDA, o extranets para su sindicación) como en lo que se refiere a los interfaces de interacción con las distintas fuentes de datos actuales o futuras.

Una vez explicado el sistema del Model-View-Controller, pasamos a explicar cada uno de las clases o paquetes principales que conforman el subsistema que estamos explicando:

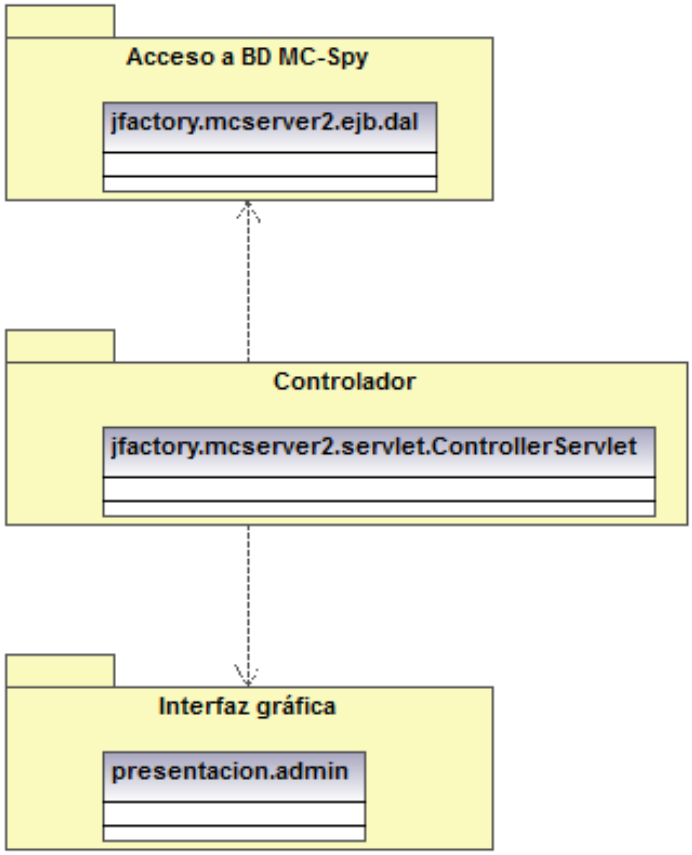


Figura 21- Subsistema Cliente de consultas

El paquete de la interfaz gráfica contendrá los jsps encargados de mostrar gráficamente al usuario la información solicitada a través del navegador.

La clase ControllerServlet será la que gestione tanto el flujo desde el navegador hacia la parte *Model*, como la inversa, es decir, decidirá el jsp a mostrar según la acción que se haya ejecutado.

El paquete de Acceso a BD se encargará precisamente de las búsquedas en la base de datos.

B.1.3 Diseño de los orígenes de datos (BD)

En este apartado se va a detallar cual va a ser el origen de toda la información que MC-Spy va a poder mostrar, y cómo la va a almacenar. Como se verá a continuación, el principal método será a través de base de datos.

Se adoptan los siguientes acuerdos de nomenclatura para tener en cuanta a lo largo del presente documento:

- El nombre de una tabla se especificará entre corchetes:

  [nombre_tabla]

- Aquellos campos que sean clave de la tabla, se marcarán con un subrayado:

  Campo_PK

B.1.3.1 Diseño lógico de MCSpy BD

El modelo lógico de la base de datos de MC-Spy se ha realizado mediante un modelo Entidad-Relación. Este modelo es una representación lógica de las entidades existentes en modelo de datos, y las relaciones existentes entre ellas.
Este modelo representa un esquema conceptual (es decir, es un modelo lógico) de la información que desea almacenarse en la base de datos. A partir de estos conceptos se realizará la decisión física de la metodología que va a utilizarse – bien relacional o bien orientada a objetos-. La utilización del modelo ER retrasa la elección entre un modelo relacional o un modelo orientado a objetos, ya que a partir de un MER puede inferirse cualquiera de las dos metodologías, en función de las necesidades o restricciones existentes en el sistema en desarrollo.
Después de esta primera introducción, veamos en la siguiente ilustración el esquema entidad-relación definido para MC-Spy. Como puede observarse en la ilustración, nos encontramos ante un esquema simple en el que las entidades están perfectamente definidas y el descubrimiento de las relaciones existentes entre ellas resulta simple. El modelo que parece en la mencionada imagen ha sido realizado a partir de la descripción funcional realizada en el Análisis de Requisitos.
Las entidades que forman la columna vertebral del esquema son dos: Item_Informacion y Estadística, mientras que el resto de entidades representan información auxiliar y necesaria según los requisitos.
Una breve explicación de las entidades:

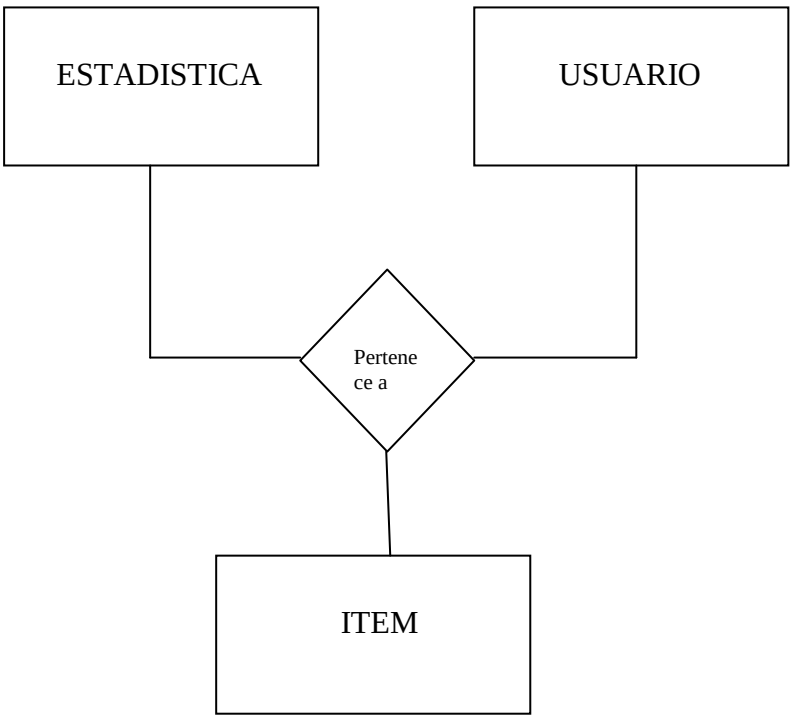
- Item_Informacion: Representa una acción que realiza cada uno de los usuarios que accede a la banca electrónica. Cada operativa que ejecute, quedará registrada como una entrada en esta entidad. Sus atributos serán: usuario, fecha, estado,…
- Usuario: Cada uno de los usuarios de la banca.
- Estadística: Representa cada una de las estadísticas que la aplicación podrá mostrar. Sus atributos podrán ser su nombre, el tipo, la leyenda a mostrar,…

Las relaciones que aparecen en el modelo lógico son muy sencillas, puesto que representan asociaciones reales entre las entidades. En la fase de diseño físico se puede

decidir la adición de más relaciones (generalmente redundantes) para mejorar la explotación de la información contenida en la base de datos.

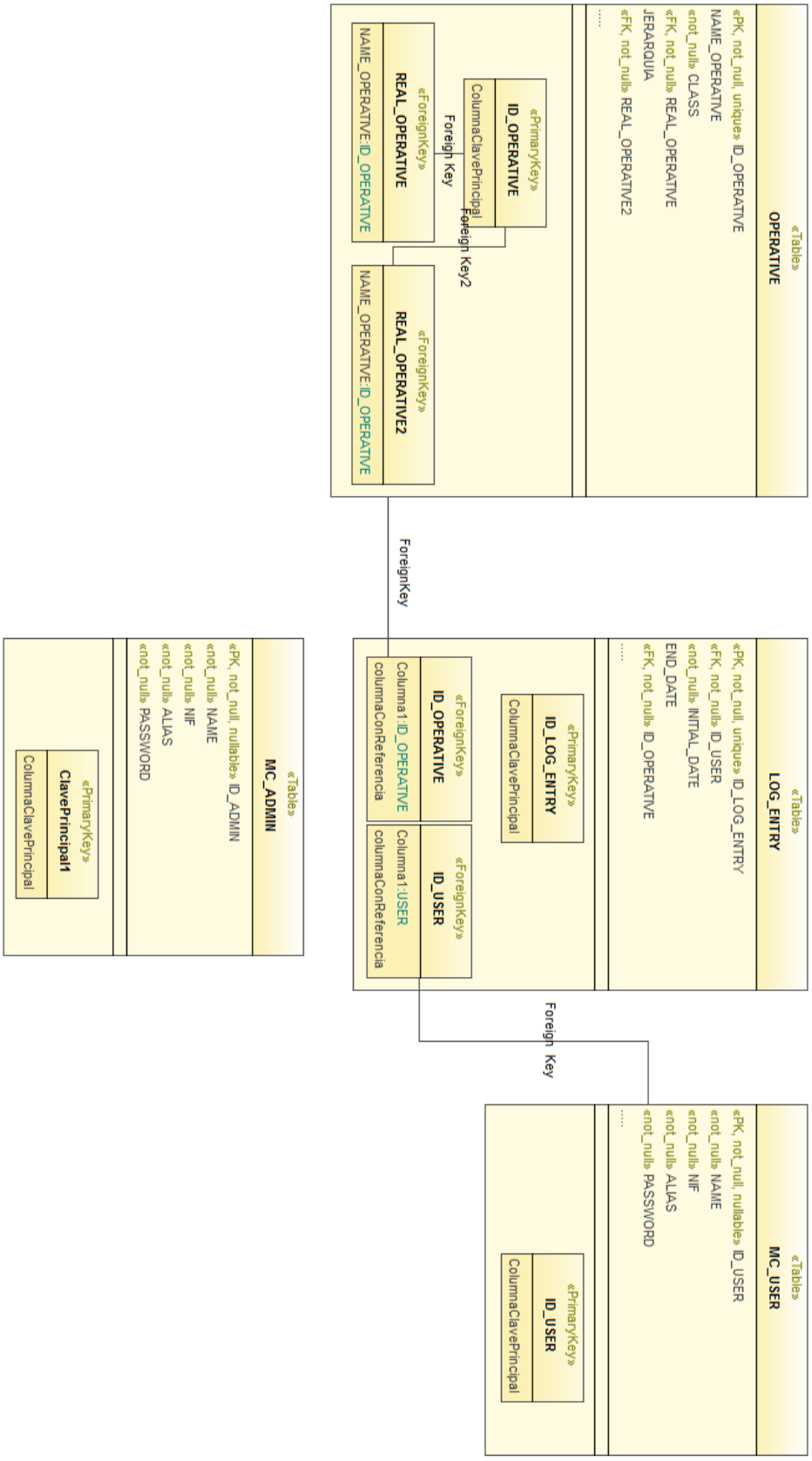
Las relaciones existentes en el esquema son las siguientes:

- Un Item pertenece a una cierta estadística.
- Un Item pertenece a un cierto usuario.



B.1.3.2 Creación de la base de datos

Una vez obtenido el diagrama entidad-relación de la base de datos, es la hora de pasar éste a unas tablas físicas que poder crear en el sistema gestor de base de datos.
La base de datos en sí, es la que muestra el siguiente gráfico:


«Table»
OPERATIVE
«PK, not_null, unique» ID_OPERATIVE
NAME_OPERATIVE
«not_null» CLASS
«FK, not_null» REAL_OPERATIVE
JERARQUIA
«FK, not_null» REAL_OPERATIVE2
.....
«PrimaryKey»
ID_OPERATIVE
ColumnaClavePrincipal
Foreign Key
Foreign Key2
«ForeignKey»
REAL_OPERATIVE
NAME_OPERATIVE:ID_OPERATIVE
«ForeignKey»
REAL_OPERATIVE2
NAME_OPERATIVE:ID_OPERATIVE
ForeignKey
«Table»
LOG_ENTRY
«PK, not_null, unique» ID_LOG_ENTRY
«FK, not_null» ID_USER
«not_null» INITIAL_DATE
END_DATE
«FK, not_null» ID_OPERATIVE
.....
«PrimaryKey»
ID_LOG_ENTRY
ColumnaClavePrincipal
«ForeignKey»
ID_OPERATIVE
Columna1:ID_OPERATIVE
columnaConReferencia
«ForeignKey»
ID_USER
Columna1:USER
columnaConReferencia
Foreign Key
«Table»
MC_USER
«PK, not_null, nullable» ID_USER
«not_null» NAME
«not_null» NIF
«not_null» ALIAS
«not_null» PASSWORD
.....
«PrimaryKey»
ID_USER
ColumnaClavePrincipal
«Table»
MC_ADMIN
«PK, not_null, nullable» ID_ADMIN
«not_null» NAME
«not_null» NIF
«not_null» ALIAS
«not_null» PASSWORD
«PrimaryKey»
ClavePrincipal1
ColumnaClavePrincipal

Como puede observarse, la base de datos que utiliza MC-Spy consta de cuatro tablas. Inicialmente puede parecer un número bajo o insuficiente de tablas, o dicho de otra forma, que la base de datos que se utiliza tiene una complejidad mínima.
Sin embargo esto no es así, ya que lo que se ha intentado en todo momento es conseguir dicha sencillez en la base de datos, para poder conseguir con posterioridad una portabilidad a otros proyectos de una forma fácil y sin problemas.

La tabla **[LOG_ENTRY]** almacena toda la información recogida para cada operativa ejecutada en la aplicación Web monitorizada mediante MC·Spy. Los campos de esta tabla son los siguientes:

	Atributo	TipoDato	Not null	Unique	Descripción
PK	**ID_LOG_ENTRY**	Number	YES	YES	Identificador autonumérico de la información almacenada para una operativa
FK	**ID_USER**	Varchar2	YES	NO	Identificador del usuario que ejecutó la operativa
	INITIAL_DATE	Date	YES	NO	Fecha de inicio de la operativa
	END_DATE	Date	NO	NO	Fecha de inicio de la operativa
	ID_SESSION	Varchar2	YES	NO	Identificador de la sesión del usuario que ejecutó la operativa
	STATUS	Number	NO	NO	Estado de la finalización de la operativa
	AMOUNT1	Number	NO	NO	Primer campo numérico para almacenar una cantidad de la operativa
	AMOUNT2	Number	NO	NO	Segundo campo numérico para almacenar una cantidad de la operativa
FK	**PROVINCIA**	Number	NO	NO	Campo para almacenar la información con la provincia de una operativa
	AUXILIAR1	Varchar2	NO	NO	Primer campo para almacenar una cadena de texto con información de la operativa
	AUXILIAR2	Varchar2	NO	NO	Segundo campo para almacenar una cadena de texto con información de la operativa
FK	**ID_OPERATIVE**	Number	YES	NO	Identificador de la operativa
	TIPO	Number	NO	NO	Tipo de la operativa
	SUBTIPO	Number	NO	NO	Subtipo de la operativa
	AUXILIAR3	Varchar2	NO	NO	Tercer campo para almacenar una cadena de texto con

					información de la operativa, permite almacenar textos de hasta 250 caracteres

La tabla **[OPERATIVE]** almacena las distintas estadísticas que se pueden consultar desde las paginas de administración de MC·Spy. Los campos de la tabla son los siguientes:

	Atributo	TipoDato	Not null	Unique	Descripción
PK	**ID_OPERATIVE**	Number	YES	YES	Identificador de la estadística
	NAME_OPERATIVE	Varchar2	NO	NO	Nombre de la estadísticas
	CLASS	Number	YES	NO	Tipo de estadística
FK	**REAL_OPERATIVE**	Number	YES	NO	Operativa sobre la que se realizará la estadística
	JERARQUIA	Varchar2	NO	NO	Permite organizar las estadísticas según una jerarquía
FK	**REAL_OPERATIVE2**	Number	YES	NO	En caso de tratarse de una estadística que compare dos operativas, este campo indica la segunda de ellas
	LEYENDA	Varchar2	NO	NO	Leyenda de la gráfica
	MOSTRAR	Number	NO	NO	Este campo permite ocultar o mostrar una estadística en la administración

La tabla **[MCUSER]** contiene todos los usuarios posibles que pueden acceder a la banca electrónica, y que por lo tanto, podrán aparecer como posibles usuarios, y de los cuales se podrá obtener información a través de MC-Spy sobre la utilización que hacen de la banca. Los campos de la tabla son los siguientes:

	Atributo	TipoDato	Not null	Unique	Descripción
PK	**ID_USER**	Number	YES	YES	Identificador único del usuario
	NAME	Varchar2	YES	NO	Nombre y apellido del usuario
	NIF	Varchar2	YES	NO	DNI/NIF del usuario
	ALIAS	Varchar2	YES	NO	Alias con el cual el usuario se puede identificar en la banca

					electrónica (a la hora de hacer login en ella)
	PASSWORD	Varchar2	YES	NO	Password encriptado con el cual acceder a la banca electrónica
	ESTADO	Number	YES	NO	Estado del usuario. Usado en el caso de que al usuario se le inhabilite del uso de la banca temporalmente, o se detecte que hace mucho tiempo que ya no la usa
	FECHA_ALTA	Date	NO	NO	Fecha en la que se dio de alta en la banca electrónica para su uso
	FECHA_ULTIMO_ACCESO	Date	NO	NO	Fecha en la cual ha hecho el usuario su último acceso a la banca electrónica

La tabla **[MCADMIN]** contiene todos los posibles administradores que van a poder acceder a MC-Spy, y por lo tanto van a tener acceso a toda la información que este sistema puede mostrar. Los campos de la tabla son los siguientes:

	Atributo	TipoDato	Not null	Unique	Descripción
PK	**ID_ADMIN**	Number	YES	YES	Identificador único del administrador
	NAME	Varchar2	YES	NO	Nombre y apellido del administrador
	NIF	Varchar2	YES	NO	DNI/NIF del administrador
	ALIAS	Varchar2	YES	NO	Alias con el cual el administrador se puede identificar en el acceso a MC-Spy (a la hora de hacer login en el)
	PASSWORD	Varchar2	YES	NO	Password encriptado con el cual acceder a MC-Spy

B.1.3.3 Diseño físico de la base de datos

Partiendo de las tablas que acabamos de obtener, el paso para obtener el diseño físico de la base de datos es simple. Tenemos que adecuar dichas tablas a clases, y organizar el flujo de información entre ellas.
Las clases e interfaces que van a presentarse a continuación conforman tanto el gestor de la base de datos, como la base de datos propiamente dicha, puesto que se trata de una base de datos orientada a objetos.
De este modo, en las propias clases están definidos tanto los datos y estructuras que se almacenan, como la lógica de negocio referente a cada clase (permisos, actualizaciones en cascada, etc.).
El esquema que representa dicha organización es el siguiente:

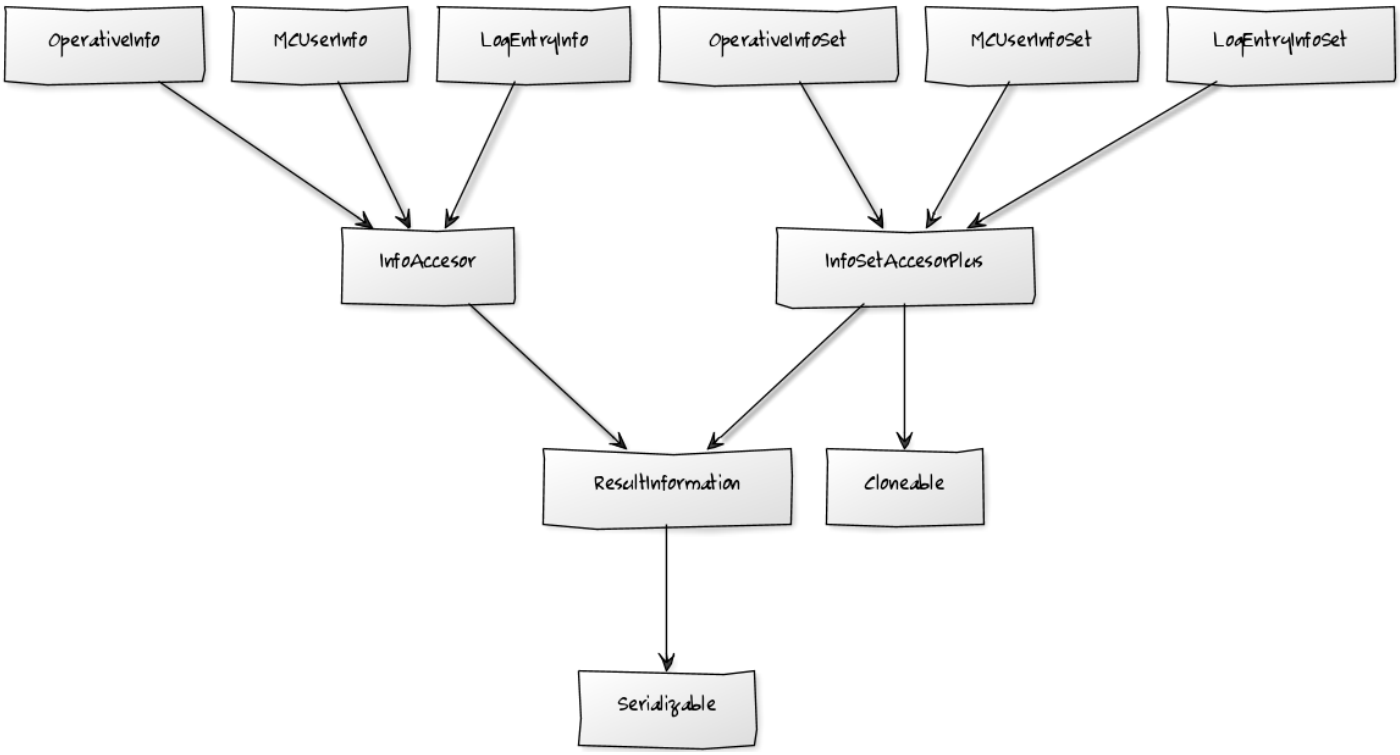


Figura 22- Diseño físico BD

Vamos a explicar a continuación la lista de clases e interfaces que conforman este módulo.

- El interfaz *ResultInformation* será un interfaz el cual tendrá que ser implementado por todas aquellas clases que necesiten devolver un *ResultInfo*, como es el caso.

- La clase *InfoAccessor*, implementa como hemos dicho el interfaz *ResultInformation*, y almacena la información de la instancia de un solo elemento, con el añadido de que esta clase ofrece métodos para convertir sus datos a xml, a *String*, a CSV, a Array.
- La clase *InfoSetAccessorPlus*, implementa también el interfaz *ResultInformation*, y almacena varias instancias de *InfoAccesor*. A su vez, ofrece funcionalidades añadidas, como la ordenación de estos elementos por alguno de sus campos.
- La clase *LogEntryInfo*, extiende de *InfoAccessor*, y contiene la información de una fila de la tabla **[LOG_ENTRY]**. Es decir, contiene información sobre una acción realizada por un usuario en la banca.
- La clase *McUserInfo*, extiende de *InfoAccessor*, y contiene información de una fila de la tabla **[MCUSER].** Es decir, contiene información sobre un usuario de la banca.
- La clase *OperativeInfo*, extiende de *InfoAccessor*, y contiene información de una fila de la tabla **[OPERATIVE]**. Es decir, contiene información sobre una posible estadística que debe mostrar MC-Spy.
- La clase *LogEntryInfoSet*, extiende de *InfoSetAccessorPlus*, y contendrá varias instancias de la clase *LogEntryInfo*.
- La clase *McUserInfoSet*, extiende de *InfoSetAccessorPlus*, y contendrá varias instancias de la clase *McUserInfo*.
- La clase *OperativeInfoSet*, extiende de *InfoSetAccessorPlus*, y contendrá varias instancias de la clase *OperativeInfo*.

B.1.3.4 Principales decisiones en el diseño de la BD

- Lo primero que se tuvo que tener en cuenta a la hora de diseñar la base de datos es, por supuesto, que se pudiera conservar la máxima información posible, para que la muestra posterior de datos pudiera ser lo más representativa posible.
  Para ello, en un primer lugar, se dudó entre crear una sola tabla que albergara toda la información, o crear varias tablas, concretamente una para cada una de las posibles operativas que pudiera ejecutar el usuario en la banca electrónica. Se optó por la primera opción, principalmente por dos razones: porque el número de operativas puede ser relativamente grande, y porque en el caso de ampliar su número, no habría que hacer grandes cambios en la base de datos.

- Una vez decidido esto, el gran inconveniente con el que nos encontramos fue la enorme cantidad de información que hay que guardar, y en la diversidad de dicha información, según sea de una operativa u otra.
  Para el problema de la cantidad de datos, no hay solución sencilla, por lo menos aparentemente. La información es la que es, y si se desea mostrar la mayor cantidad de ella, habrá que almacenarla de alguna forma para luego poder obtenerla. Con lo cual, el único remedio que hay es seleccionar bien que datos hay que conservar y cuáles no, por ser innecesarios o irrelevantes, y por otro lado usar algún método de optimización de conservación de datos en la base de

datos. Se pensó en un primer momento en tener en la tabla **[LOG_ENTRY]** los datos únicamente unos meses, para posteriormente, con un proceso batch o por cualquier otro medio, pasarlos a una tabla de históricos que estuviera en otra base de datos distinta, sin embargo de desechó la idea, porque en la mayoría de los casos, para una misma consulta habría que acceder a las dos bases de datos, lo que complicaría la extracción de datos y su posterior tratamiento. Además, con una sola base de datos, se tiene centralizada toda la información.

- Otro problema que surgió desde los inicios del proyecto, y que se resolvió mediante la base de datos, fue el permitir reflejar de alguna forma que las operativas que se ejecutan en una banca electrónica puede estar anidadas unas dentro de otras. Vamos a poner un ejemplo para verlo mucho mejor: Estando dentro de la banca electrónica, vamos a realizar una transferencia a otra cuenta. Estando en esa pantalla, queremos ver el listado de los distintos códigos de conceptos que existen, para indicarlo en la transferencia. Este es el ejemplo de una operativa (consulta de códigos de concepto) que se ejecuta dentro de otra (transferencia). A la hora de mostrar el árbol de sesiones (del que se ha hablado anteriormente en este documento), es muy útil poder conocer este hecho.
  Se optó por una solución sencilla, que no supusiera mucho desarrollo ni complejidad, ya que no es un caso muy habitual dentro de una banca (de hecho, se hizo una revisión de todas las operativas que se podían ejecutar en ella, y este hecho solo ocurría en un caso).
  Para ello, lo que se hizo fue añadir un campo más a la tabla **[OPERATIVE]**, que es *jerarquia*, que indica el nivel de anidamiento que supone para esa operativa ejecutada.

- Otro problema que apareció fue el poder gestionar de una manera sencilla el hecho de poder crear estadísticas que no solo se refirieran a datos de una sola operativa, sino que se pudieran comparar varias de ellas, para poder cotejarlas juntamente.
  Este hecho se consiguió también mediante la adición de un campo más a la tabla **[OPERATIVE]** : REAL_OPERATIVE2, el cual identifica la segunda variable (que en este caso sería la operativa) por la cual se hará la comparación de los datos.

- Cada interacción con la base de datos se realizará a través de transacciones atómicas ("*all-or-nothing*"); es decir, si todas las operaciones que componen una transacción se ejecutan satisfactoriamente, se realiza un *commit*, y si alguna de ellas falla, se aborta la transacción, para llevar a la base de datos al mismo estado que tenía antes de comenzar la transacción: un *rollback*.

- Es normal también que una vez instalado MC-Spy dentro de un sistema, éste se desarrolle y crezca. Por ejemplo, en el caso de una banca electrónica, es normal que el número y la variedad de funciones que ofrezca se aumente con el tiempo. Pensando en esto, se añadió a la tabla **[LOG_ENTRY]** un campo que podríamos llamar "auxiliar", pero que llegado el momento puede llegar a ser muy util. Es el campo *auxiliar3*, que es un texto, el cual, como decimos, cuando se cree una nueva operativa que contenga un dato que hasta ahora no se ha

tenido en cuenta en el desarrollo de MC-SPy, se pueda añadir dicho dato en este campo, haciendo que no se tenga que modificar el código de MC-Spy en nada, o al menos mínimamente.

B.1.4 Diseño de la interfaz gráfica

En este apartado se detallará el diseño que se hizo de la interfaz gráfica de la parte de la administración de MC-Spy.

B.1.4.1 Árbol de usuarios

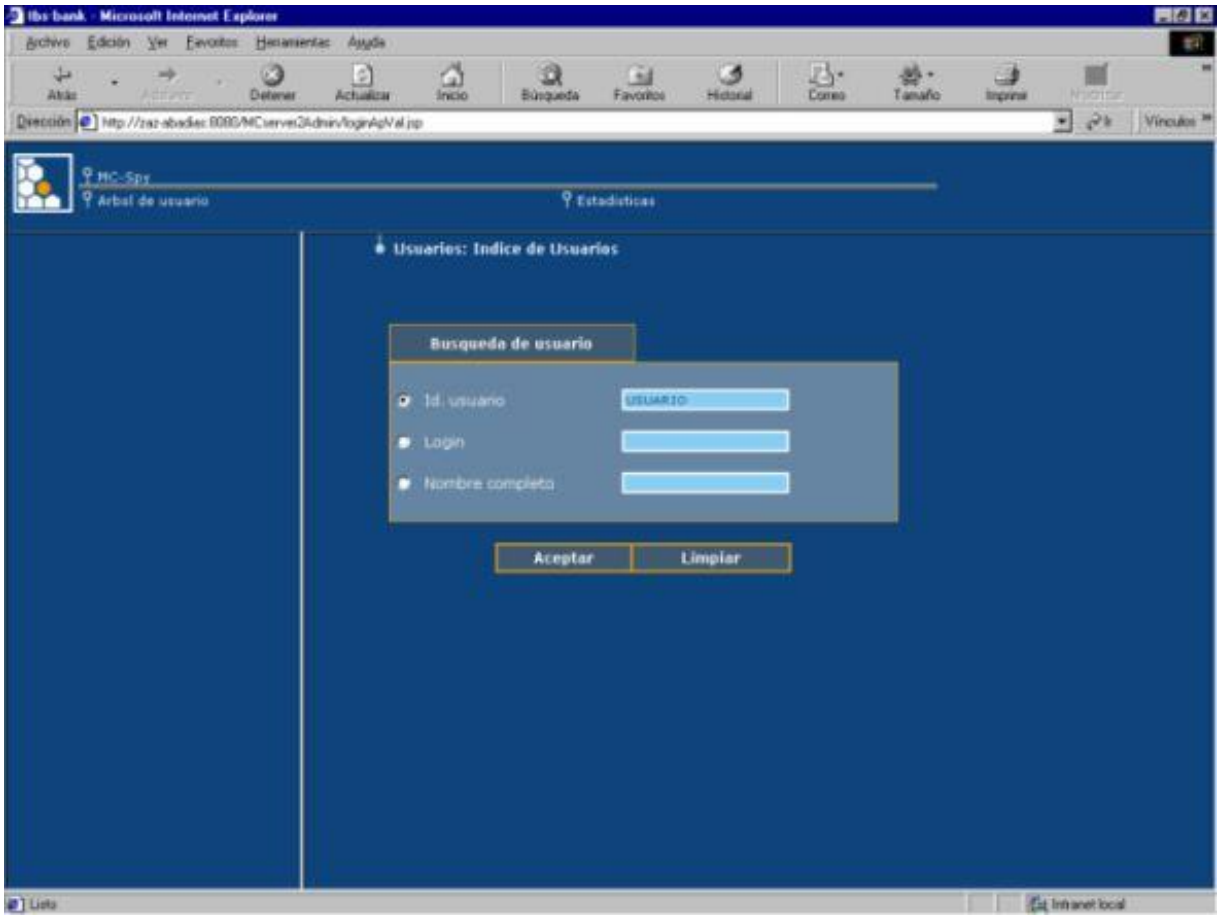


Figura 23- Página selección o búsqueda usuario

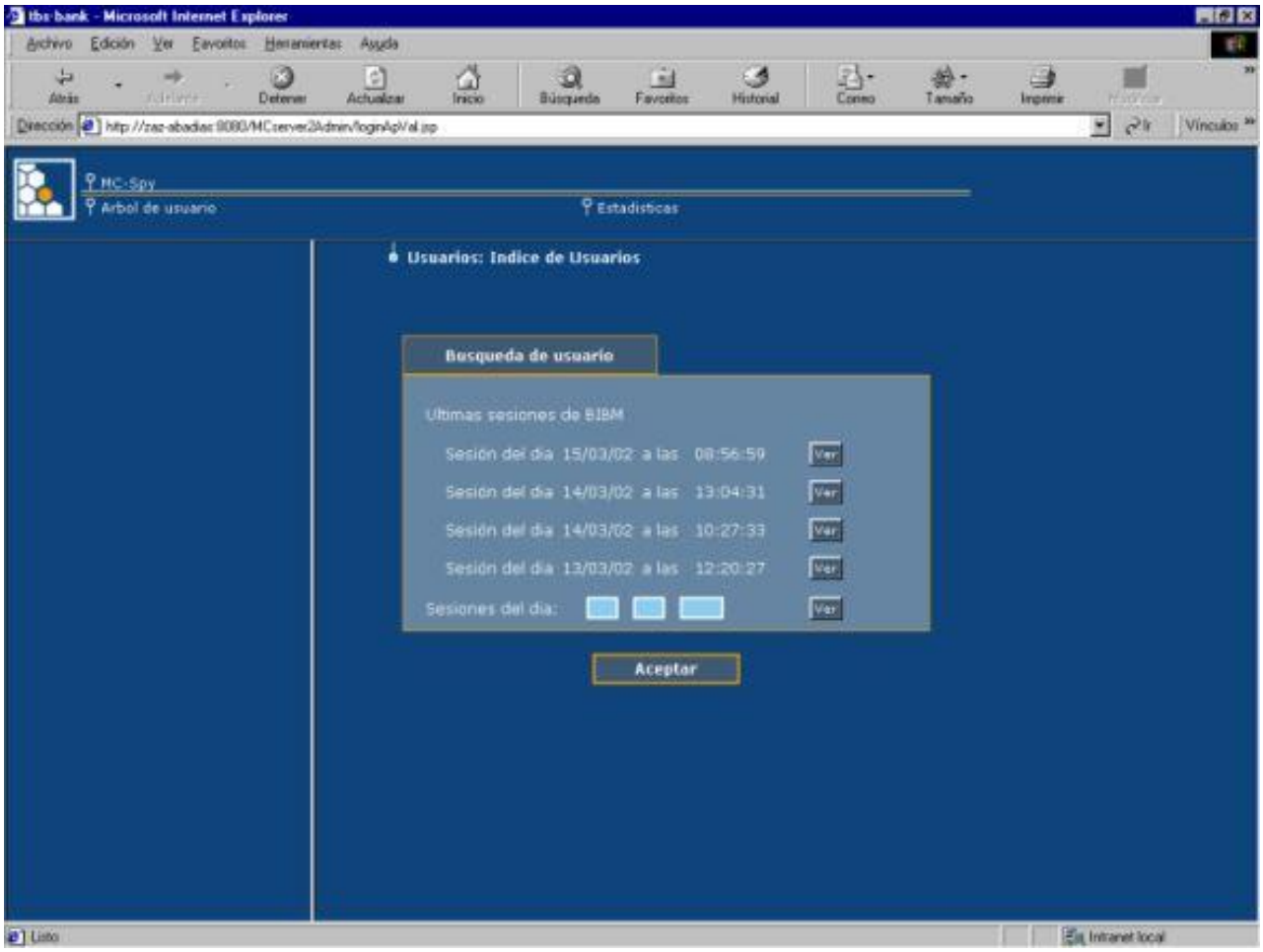


Figura 24- Última sesión usuario seleccionado

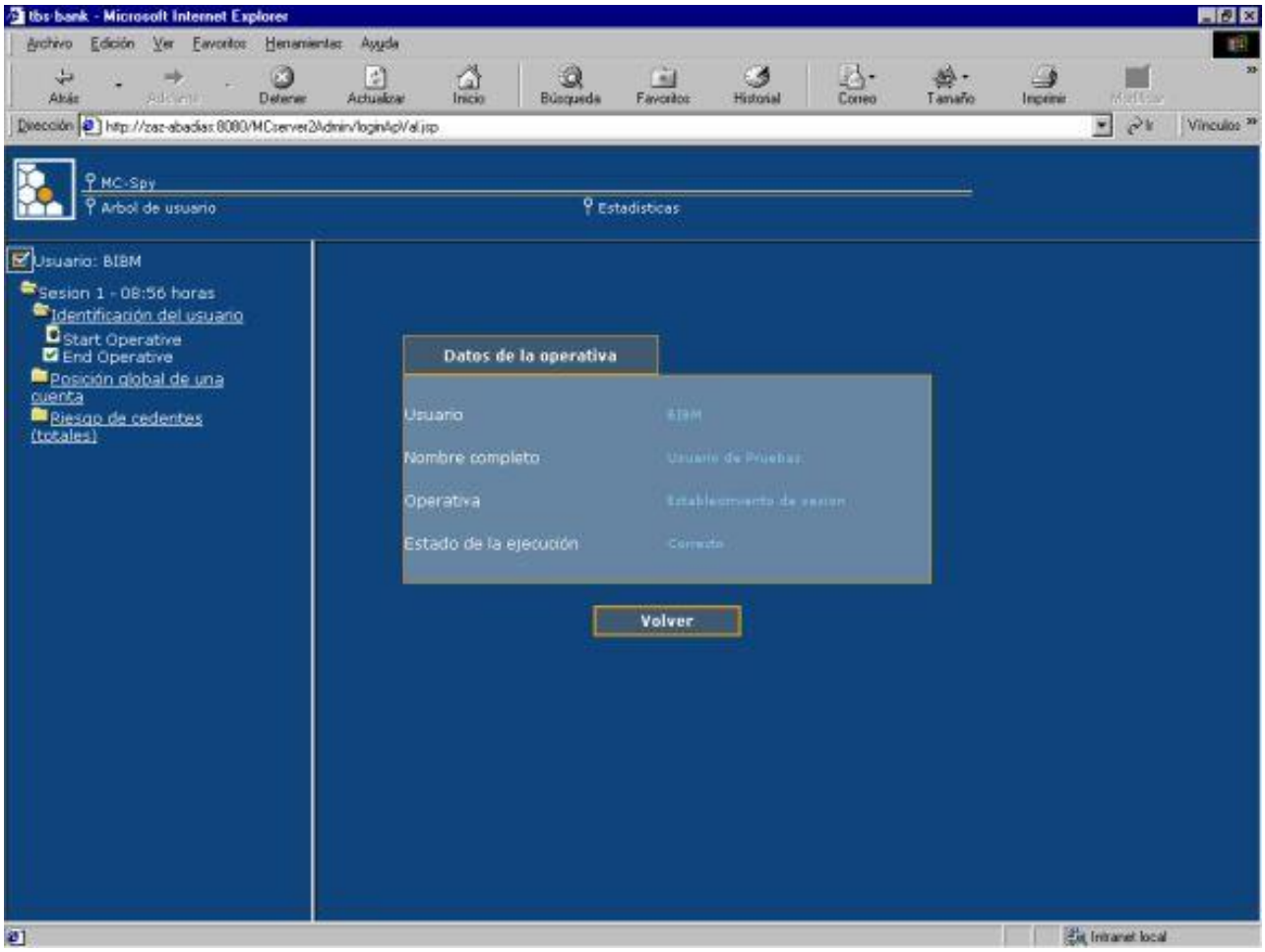


Figura 25- Información de la operativa seleccionada, junto con sesión en árbol

B.1.4.2 Consulta de estadísticas

Estas mismas pantallas servirán tanto para las consultas de estadísticas como para el cálculo de previsiones.

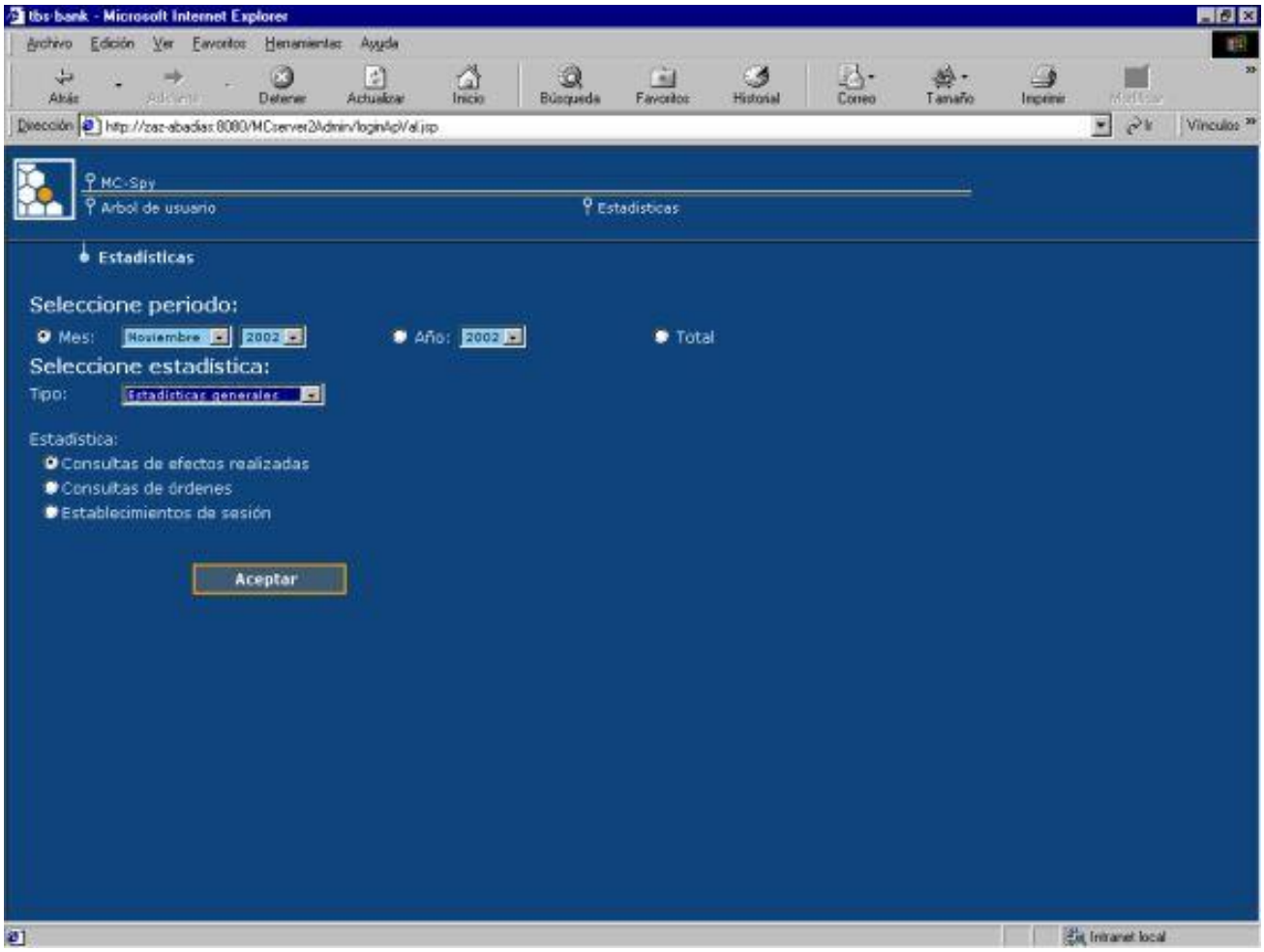


Figura 26- Selección de datos y selección de estadística

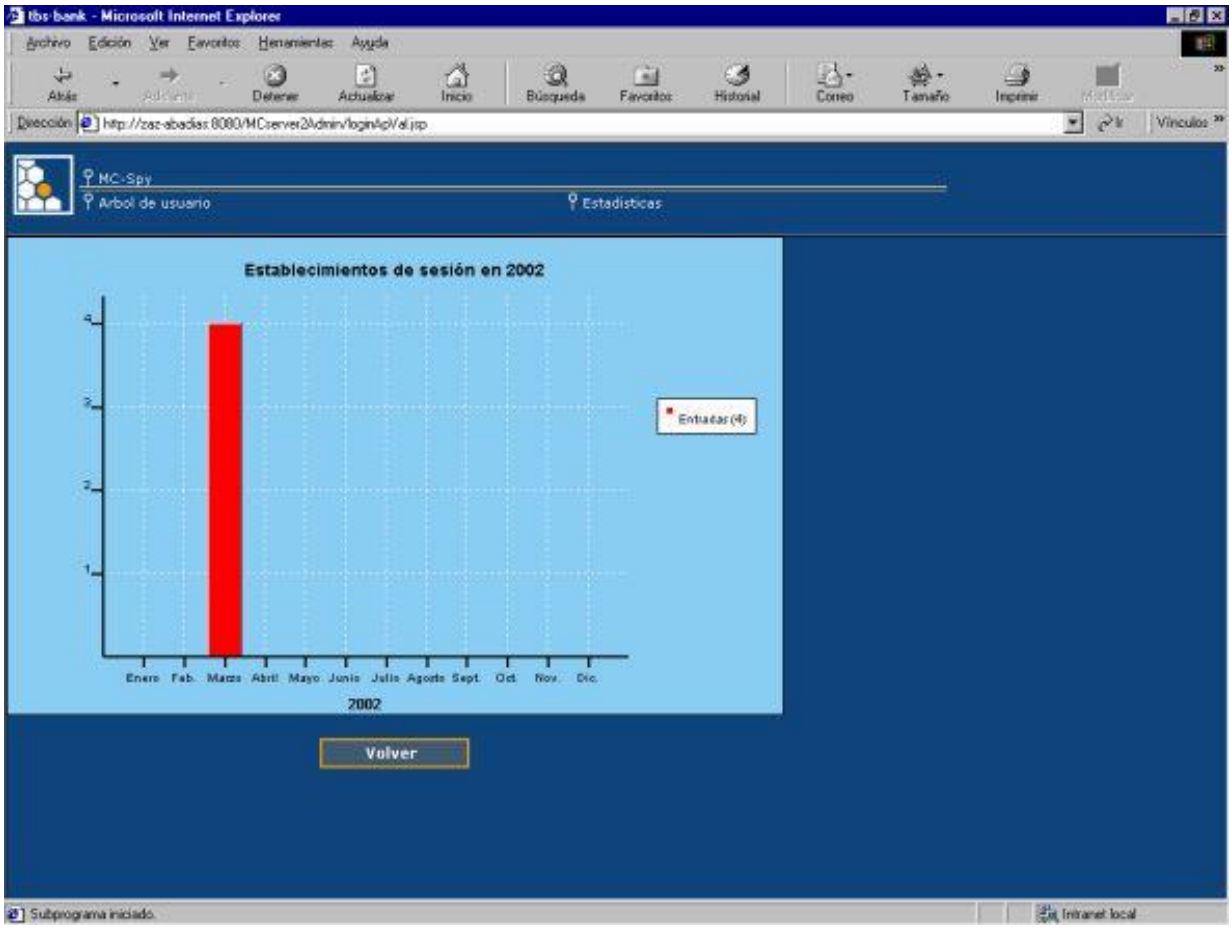


Figura 27- Ejemplo de serie con estacionalidad

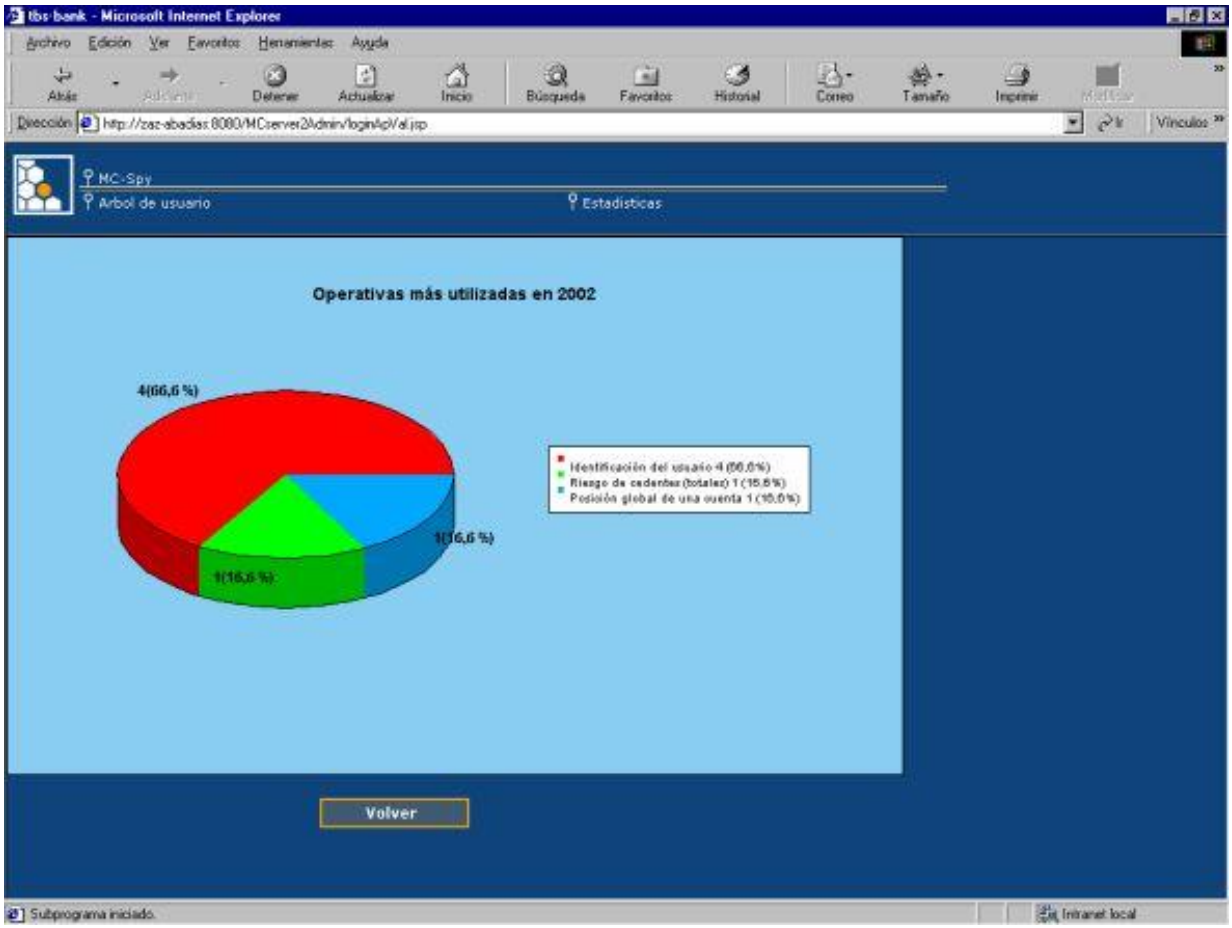


Figura 28- Ejemplo de gráfica con estadísticas

Anexo C. Implementación

Durante el desarrollo del proyecto se ha creado una implementación de los distintos subsistemas, todo ello en lenguaje Java (JDK 1.3, 1.5,…). Debe notarse que el diseño físico ya se realizó teniendo en cuenta esta decisión.
En este capítulo se va a mostrar la estructura de clases e interfaces desde el punto de vista de implementación.
Se mostrarán así mismo los aspectos de implementación más relevantes, así como las decisiones tomadas y las dificultades encontradas, y los pasos que se dieron para solventarlas.

C.1 Paquetes que componen MC-Spy

La distribución de clases e interfaces en Java se realiza mediante paquetes, que no es más que una forma de jerarquizar lógicamente estos elementos.
Los paquetes que se han creado para el proyecto MC-Spy están estructurados en dos grandes bloques:

- Paquetes del Recolector de datos, que tienen el prefijo *jfactory.mcserver2.mcspy.rc*. Todos estos paquetes se corresponden con la implementación de dicho subsistema.
- Paquetes del Cliente de consultas, que tienen el prefijo *jfactory,mcserver2.mcspy* Todos estos paquetes se corresponden con la implementación de dicho subsistema.

Hacer notar que el prefijo *jfactory* es una norma de TB·Solutions, empresa en la cual recordamos que se realizó este proyecto. En todos los proyectos que se realizan en el área de Java, todos sus paquetes deben comenzar por ese prefijo, seguido del proyecto en el cual se está trabajando, De allí también que el segundo prefijo sea *mcserver2*, que es el nombre del proyecto en el que en un principio se englobó MC-Spy.
Comentar sobre los dos siguientes diagramas, que en los nombres de todos los paquetes que se muestran a continuación, tanto en el caso de los paquetes correspondientes al Recolector de datos (mcspy.rc), como al del Cliente de consultas (mcspy), les faltaría delante de todos ‘jfactory.mcserver2’, tal y como apuntaba la metodología de la empresa y que se ha comentado anteriormente.

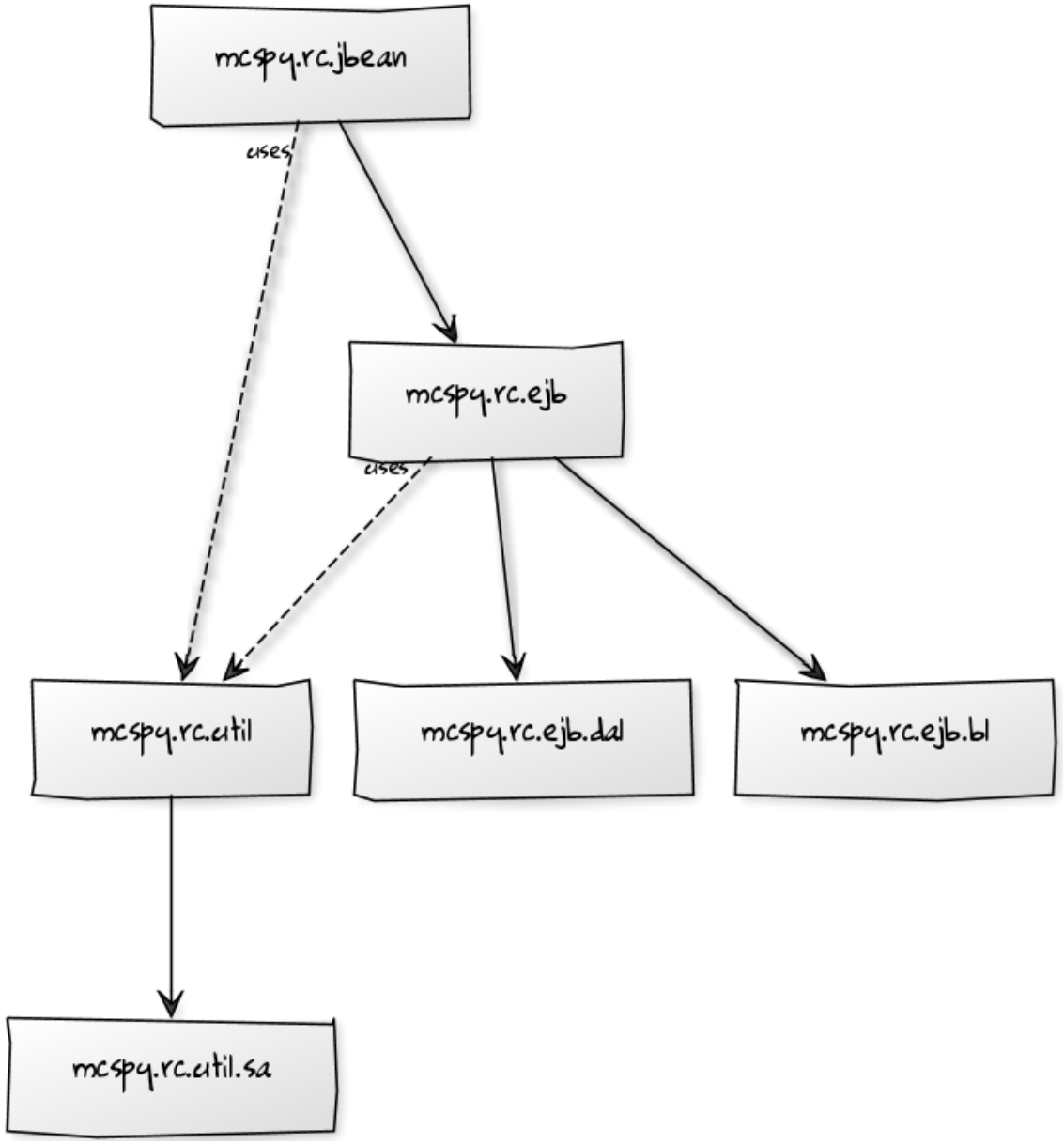


Figura 29- Paquetes genéricos Recolector de datos

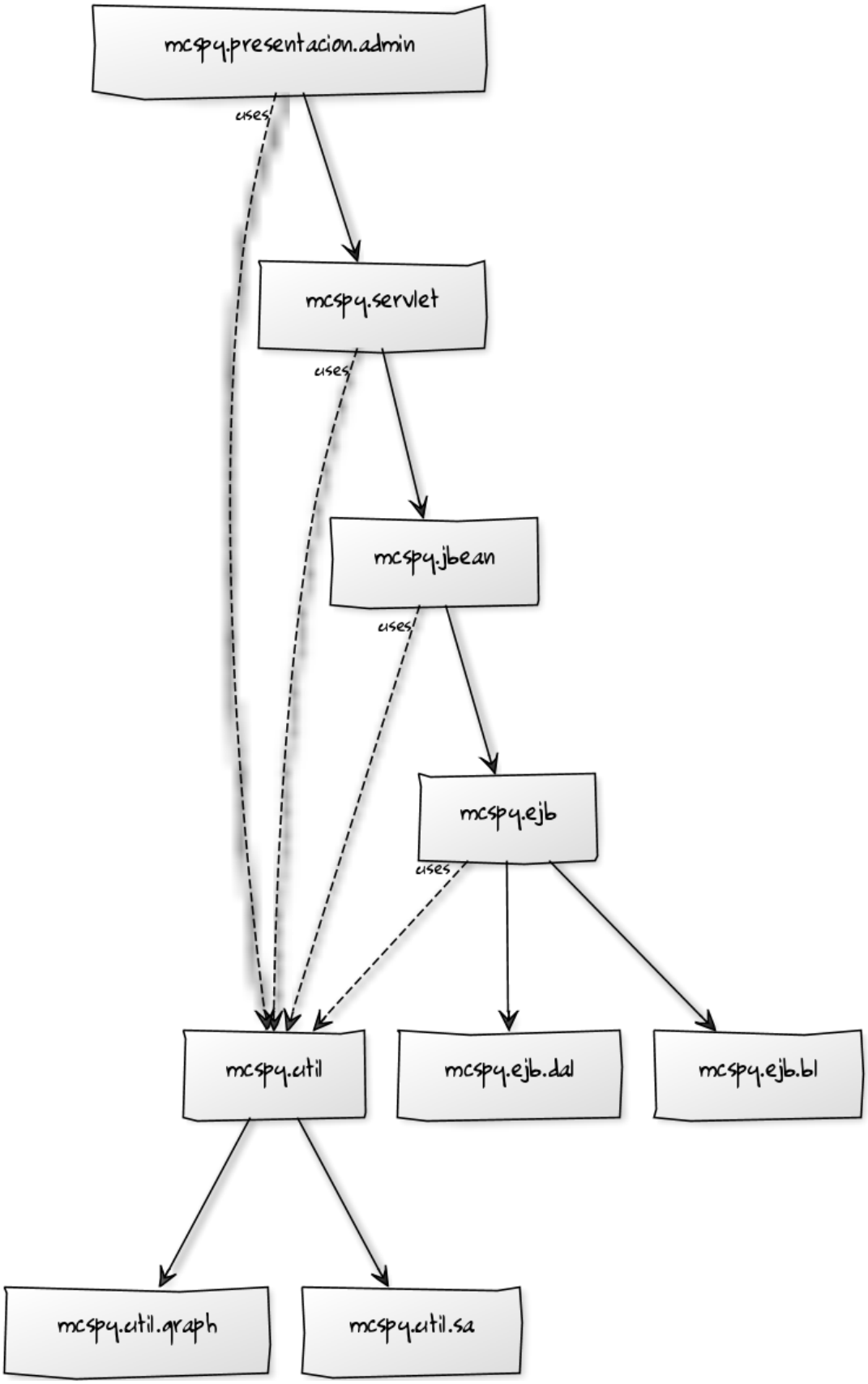


Figura 30- Paquetes genéricos Consultas

Los paquetes forman una jerarquía, tal y como se muestra en las ilustraciones anteriormente mostradas.
También hacer notar que en los gráficos se ha establecido un orden en el flujo de invocación entre los distintos paquetes. Este flujo es el que en las invocaciones que se harían al ejecutarse las funcionalidades del proyecto, se seguiría en un orden normal.
De igual forma, se han identificado las clases que van a ser utilizadas por otras para el "transporte" de información de unas a otras, es decir, actuando éstas como posibles "*beans*".
Simplificando y por regla general, podríamos decir que casi todos los módulos identificados y diseñados anteriormente en el diseño, generan un paquete Java en la fase de implementación. En este proyecto no siempre se ha seguido esta regla, unificando varios módulos en un mismo paquete de implementación, o viceversa, derivando un mismo módulo en varios paquetes de implementación. Así pues, la correspondencia entre un módulo del diseño lógico y un paquete Java no es exacta, pero se respeta el ánimo de cada uno de los módulos identificados. El autor estimó conveniente no realizar una correspondencia directa, lo cual da mucha mayor flexibilidad si en un futuro se rediseña el sistema y se redistribuyen funcionalidades entre procesos, o se crean nuevas.

C.1.1 Paquetes generales

Se han creado varios paquetes generales, los cuales corresponden con clases e interfaces que proveen funcionalidades verticales; es decir, se trata de paquetes genéricos independientes del sistema que se está diseñando y que pueden y deben ser reutilizados por otros proyectos.
A continuación se va a proceder a enumerar todos y cada uno de los paquetes de los que consta el proyecto, así como la mayoría de las clases que lo componen. No se relatan todas estas clases, sino solamente aquellas que se consideran suficientemente importantes, y que su utilidad tiene cierta relatividad para el desarrollo y ejecución del proyecto.
Los paquetes generales creados son los siguientes:

- Paquete *jfactory.mcserver2.util.sa* : Este paquete contiene clases e interfaces necesarios para almacenar información, y transportarla a lo largo del flujo de invocación entre las clases implicadas en éste. Contiene pues todos los '*infos*' e '*infosets*'[1] que sirven para mantener la información a lo largo de la ejecución de una funcionalidad del proyecto. Las clases que contiene este paquete son las siguientes:

  - *ResultInformation*: Interfaz que extiende de *Serializable*, y será el interfaz que deberán implementar para las demás clases de este paquete. Utiliza los siguientes paquetes ajenos al proyecto:

    - *java.io.Serializable*

  - *ResultInfo*: Clase que implementa *ResultInformation*, y que implementa las estructuras necesarias para que las clases que van a extender de ella, contengan lo necesario para mantener información sobre posibles errores, mensajes o códigos de error, y códigos de terminación de una ejecución. Utiliza los siguientes paquetes ajenos al proyecto:

    - *java.io.Serializable*

[1]: En TB-Solutions, por razones "históricas" se suelen llamar infos e infosets a los contenedores de información, a los beans que se utilizan para el mantenimiento de datos entre unas clases y otras.

- *InfoAccesor*: Clase abstracta de la que van a extender todos los '*infos*', es decir, todas las clases que se utilizan como mantenedores de información (asociándose casi siempre a la información de una fila de una tabla de la base de datos). Contiene métodos de utilidades para esas clases, como por ejemplo, transformar todo su contenido a *String*, a *xml*, a *CSV*, etc. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.lang.reflect.Method*
    - *java.util.ArrayList*

- *InfoSetAccessorPlus*: Clase abstracta de la que van a extender todos los '*infoset*', es decir, todas las clases que se utilizan como mantenedores de información, conteniendo varias instancias de '*infos*' (es decir, se asocia a la información de una tabla, o mejor dicho, de varias filas de una tabla de base de datos). Realiza una gestión de forma óptima estos '*infos*' para poder ejecutar sobre ellas operaciones como búsqueda, etc. También contiene métodos que sirven como utilidades para estas clases que extenderán de ella. Por ejemplo, es muy útil el método que ordena los '*infos*' que contiene, según el parámetro que se le diga. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.lang.reflect.Method*
    - *java.util.Iterator*
    - *java.util.ArrayList*
    - *java.io.Serializable*

- *LogEntryInfo*: Contenedor de información de una fila de la tabla **[LOG_ENTRY]**. Extiende de la clase abstracta *InfoAccessor*. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.util.Date*

- *LogEntryInfoSet*: Contenedor de información de varias filas de la tabla **[LOG_ENTRY]**. Extiende de la clase abstracta *InfoSetAccessorPlus*. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.util.Collection*
    - *java.util.Vector*
    - *java.util.Iterator*

- *OperativeInfo*: Contenedor de información de una fila de la tabla **[OPERATIVE]**. Extiende de la clase abstracta

InfoAccessor. Utiliza los siguientes paquetes ajenos a este proyecto:

- *java.util.Date*

- *OperativeInfoSet*: Contenedor de información de varias filas de la tabla **[OPERATIVE]**. Extiende de la clase abstracta *InfoSetAccessorPlus*. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *java.util.Collection*
  - *java.util.Vector*
  - *java.util.Iterator*

- *MCUserInfo*: Contenedor de información de una fila de la tabla **[MCUSER]**. Extiende de la clase abstracta *InfoAccessor*. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *java.util.Date*

- *MCUserInfoSet*: Contenedor de información de varias filas de la tabla **[MCUSER]**. Extiende de la clase abstracta *InfoSetAccessorPlus*. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *java.util.Collection*
  - *java.util.Vector*
  - *java.util.Iterator*
- *MCAdminInfo*: Contenedor de información de una fila de la tabla **[MCADMIN]**. Extiende de la clase abstracta *InfoAccessor*. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *java.util.Date*

- *MCAdminInfoSet*: Contenedor de información de varias filas de la tabla **[MCADMIN]**. Extiende de la clase abstracta *InfoSetAccessorPlus*. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *java.util.Collection*
  - *java.util.Vector*
  - *java.util.Iterator*

- Paquete *jfactory.mcserver2.util*: Este paquete contiene todas las clases que sirven de ayuda al resto de las clases y paquetes del proyecto. Contienen métodos utilizados por todos ellos, tanto por el subsistema de Recolección de datos como el del Cliente de consultas, ya que así se decidió para tener todas estas clases mas centralizadas y poder favorecer la reutilización de las mismas a lo largo del proyecto. Las clases que contiene este paquete son las siguientes:

- *Constantes*: Clase que contiene todas las constantes utilizadas en todo el proyecto.

- *MCServerUtil*: Clase que contiene varios métodos e utilidades para el manejo de distintos tipos de datos, transformación y migración entre ellos, etc. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.Collection*
    - *java.util.Vector*
    - *java.util.Iterator*
    - *java.io.StringWriter*
    - *java.io.PrintWriter*

- *Propiedades*: Clase que extiende la clase *java.util.Properties*, y que se utiliza para obtener los valores contenidos en los ficheros de propiedades. De la clase que extiende, le añade varias funcionalidades, como por ejemplo, se han añadido los métodos *getString*(), *getBoolean*(), *getInt*(), *getDate*() y *getFloat*(), los cuáles devuelven la propiedad pasada como parámetro comoo un *String*, un *boolean*, un *int*, un *Date* y un *float* respectivamente. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.Properties;*
    - *java.io.File;*
    - *javax.ejb.EJBObject;*
    - *javax.naming.Context;*
    - *javax.naming.InitialContext;*
    - *javax.naming.NamingEnumeration;*
    - *javax.naming.NamingException;*
    - javax.naming.Binding;

- *Log*: Interface para escribir los mensajes de error y el fichero de log de la aplicación. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.Properties;*

- *LogFactory*: Clase encargada de instanciar la clase que implementa el interface Log.

- Paquete *jfactory.mcserver2.servlet*: Paquete que contiene los *servlets* necesarios para llevar el control entre la parte '*View*' y la parte '*Model*' del proyecto. Es decir, decide el jsp a mostrar según la acción ejecutada, y

decide que acción ejecutar según la acción que se haya realizado en el jsp. En nuestro caso se decidió que esta función la podía realizar un solo *servlet*, ya que el número de jsps y de acciones no era muy elevada. Las clases que contiene este paquete son las siguientes:

- ControllerServlet: Servlet que realiza el control de las acciones a ejecutar y los jsps a mostrar. Utiliza los siguientes paquetes ajenos a este proyecto:
  - *javax.servlet.HttpServletRequest;*
  - *javax.servlet.HttpServletResponse*
  - *javax.servlet.http.HttpRequest;*
  - *java.util.HashMap;*
  - *java.util.ArrayList;*
  - *java.util.Calendar;*
  - *java.io.File;*

- Paquete *jfactory.mcserver2.jbean*: Paquete que contiene las clases que hacen de unión entre el *servlet* y los *EJB's* que se encargarán de consultar a la base de datos. Se procura mover la mayor cantidad de código a estas clases, para evitar a los *EJB's* esta carga. Las clases que contiene este paquete son las siguientes:
  - *BaseJBean*: Clase abstracta de la cual tendrán que extender las restantes clases pertenecientes a este paquete. Ofrece a éstas servicios de paginación para los listados de elementos, además de otras servicios como log, propiedades, contexto para acceso a los ejbs, etc. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.util.ArrayList;*
  - *AccountsJBean*: Clase que hace de nexo entre el *servlet* y el *EJB* asociado a la tabla de cuentas de la base de datos (es una tabla ya existente, perteneciente al proyecto MC-Server, por eso no se nombró en el diseño). A través de ella (y por extensión, todas las que ella invoca y utiliza) se obtienen los datos bancarios de los usuarios, tales como cuentas, bancos,etc. Extiende de la clase *BaseJBean*. Utiliza los siguientes paquetes ajenos a este proyecto:
    - *java.util.ArrayList;*
  - *LogEntryJBean*: Clase que hace de nexo entre el *servlet* y el *EJB* asociado a la tabla **[LOG_ENTRY]** de la base de datos. Extiende de la clase *BaseJBean.* Utiliza los siguientes paquetes ajenos a este proyecto:

- *java.util.ArrayList;*
- *java.util.Date;*

- *OperativeJBean*: Clase que hace de nexo entre el *servlet* y el *EJB* asociado a la tabla **[OPERATIVE]** de la base de datos. Extiende de la clase *BaseJBean.* Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.util.Date;*

- *UsersJBean*: Clase que hace de nexo entre el *servlet* y el *EJB* asociado a la tabla **[MCUSER]** de la base de datos. Extiende de la clase *BaseJBean.* Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.util.Date;*

- *McAdminJBean*: Clase que hace de nexo entre el *servlet* y el *EJB* asociado a la tabla **[MCADMIN]** de la base de datos. Extiende de la clase *BaseJBean.* Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.util.Date;*

- Paquete *jfactory.mcserver2.ejb.bl*: Paquete que contiene las clases de los *EJBs* necesarios para toda la aplicación. Como se sabe, cada *EJB* lo componen 3 clases, el *Home*, el *Bean* y el *Remote*. En este paquete además, añadimos para cada uno de los *EJBs*, una clase más: el *BLImpl* (*business logic*), en la cual insertamos todo la lógica de negocio, sacándola del *EJB*, para evitar que su ejecución lo cargue en exceso. Esta clase hace también de nexo de unión entre el *EJB* y la clase que accede directamente a la base de datos. Vamos a ir relatando cada uno de los *EJBs* que componen este paquete:

  - EJB Accounts:

    - Accounts, AccountsBean, AccountsHome: Clases que componen el *EJB* de Accounts. Utilizan los siguientes paquetes ajenos a este proyecto:

      - *javax.ejb.EJBObject;*
      - *java.rmi.RemoteException;*

- *java.util.ArrayList;*
- *java.util.Iterator;*
- *javax.ejb.FinderException;*
- *javax.ejb.SessionContext;*
- *javax.ejb.SessionBean;*
- *javax.ejb.CreateException;*
- *javax.ejb.EJBHome*

- AccountsBlImpl: Clase con la implementación de la lógica de negocio del *EJB* Accounts. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.rmi.RemoteException;*
  - *java.util.Date*

- EJB LogEntry:

  - LogEntry, LogEntryBean, LogEntryHome: Clases que componen el *EJB* de LogEntry. Utilizan los siguientes paquetes ajenos a este proyecto:

    - *javax.ejb.EJBObject;*
    - *java.rmi.RemoteException;*
    - *java.util.ArrayList;*
    - *java.util.Iterator;*
    - *javax.ejb.FinderException;*
    - *javax.ejb.SessionContext;*
    - *javax.ejb.SessionBean;*
    - *javax.ejb.CreateException;*
    - *javax.ejb.EJBHome*

  - LogEntryBlImpl: Clase con la implementación de la lógica de negocio del *EJB* LogEntry. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.ArrayList;*
    - *java.rmi.RemoteException;*
    - *java.util.Date*

- EJB Operative:

  - Operative, OperativeBean, OperativeHome: Clases que componen el *EJB* de Operative. Utilizan los siguientes paquetes ajenos a este proyecto:

    - *javax.ejb.EJBObject;*
    - *java.rmi.RemoteException;*

- *java.util.ArrayList;*
- *java.util.Iterator;*
- *javax.ejb.FinderException;*
- *javax.ejb.SessionContext;*
- *javax.ejb.SessionBean;*
- *javax.ejb.CreateException;*
- *javax.ejb.EJBHome*

- OperativeBlImpl: Clase con la implementación de la lógica de negocio del *EJB* Operative. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.rmi.RemoteException;*
  - *java.util.Date*

- EJB McUser:

  - McUser, McUserBean, McUserHome: Clases que componen el *EJB* de McUser. Utilizan los siguientes paquetes ajenos a este proyecto:

    - *javax.ejb.EJBObject;*
    - *java.rmi.RemoteException;*
    - *java.util.ArrayList;*
    - *java.util.Iterator;*
    - *javax.ejb.FinderException;*
    - *javax.ejb.SessionContext;*
    - *javax.ejb.SessionBean;*
    - *javax.ejb.CreateException;*
    - *javax.ejb.EJBHome*

  - McUserBlImpl: Clase con la implementación de la lógica de negocio del *EJB* McUser. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.ArrayList;*
    - *java.rmi.RemoteException;*
    - *java.util.Date*

- EJB McAdmin:

  - McAdmin, McAdminBean, McAdminHome: Clases que componen el *EJB* de McAdmin. Utilizan los siguientes paquetes ajenos a este proyecto:

    - *javax.ejb.EJBObject;*
    - *java.rmi.RemoteException;*

- *java.util.ArrayList;*
- *java.util.Iterator;*
- *javax.ejb.FinderException;*
- *javax.ejb.SessionContext;*
- *javax.ejb.SessionBean;*
- *javax.ejb.CreateException;*
- *javax.ejb.EJBHome*

- McAdminBlImpl: Clase con la implementación de la lógica de negocio del *EJB* McAdmin. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.rmi.RemoteException;*
  - *java.util.Date*

- Paquete *jfactory.mcserver2.ejb.dal*: Paquete que contiene las clases de acceso a datos (DAL*: Data access logic*). Son realmente las clases que, a través del driver necesario para acceder a la base de datos de *Oracle*, ejecutan las sentencias SQL. Son también ellas las encargadas en transformar las respuestas que obtienen en instancias de objetos entendibles por el resto de las clases del proyecto. Lógicamente, existe una clase *DAL* por cada una de las tablas a las que accedemos en la base de datos. Las clases que contiene este paquete son las siguientes:

  - *AccountsDALImpl*: Clase de acceso a base de datos para todas aquellas tablas que contienen datos bancarios sobre los usuarios. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.ArrayList;*
    - *java.sql.Timestamp;*
    - *java.util.Date;*
    - *java.sql.Time;*
    - *java.sql.Connection;*
    - *java.sql.ResultSet;*
    - *java.sql.PreparedStatement;*
    - *java.sql.SQLException*

  - *LogEntryDALImpl*: Clase de acceso a base de datos para la tabla **[LOG_ENTRY]**. Utiliza los siguientes paquetes ajenos a este proyecto:

    - *java.util.ArrayList;*
    - *java.sql.Timestamp;*
    - *java.util.Date;*
    - *java.sql.Time;*

- *java.sql.Connection;*
- *java.sql.ResultSet;*
- *java.sql.PreparedStatement;*
- *java.sql.SQLException*

- *OperativeDALImpl*: Clase de acceso a base de datos para la tabla **[OPERATIVE]**. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.sql.Timestamp;*
  - *java.util.Date;*
  - *java.sql.Time;*
  - *java.sql.Connection;*
  - *java.sql.ResultSet;*
  - *java.sql.PreparedStatement;*
  - *java.sql.SQLException*

- *McUserDALImpl*: Clase de acceso a base de datos para la tabla **[MCUSER]**. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.sql.Timestamp;*
  - *java.util.Date;*
  - *java.sql.Time;*
  - *java.sql.Connection;*
  - *java.sql.ResultSet;*
  - *java.sql.PreparedStatement;*
  - *java.sql.SQLException*

- *McAdminDALImpl*: Clase de acceso a base de datos para la tabla **[MCADMIN]**. Utiliza los siguientes paquetes ajenos a este proyecto:

  - *java.util.ArrayList;*
  - *java.sql.Timestamp;*
  - *java.util.Date;*
  - *java.sql.Time;*
  - *java.sql.Connection;*
  - *java.sql.ResultSet;*
  - *java.sql.PreparedStatement;*
  - *java.sql.SQLException*

- Paquete *jfactory.mcserver2.rc.jbean*: Paquete que contiene las clases que hacen de unión entre el *servlet* y los *EJB's*, dentro del recolector de datos, y por lo tanto, integradas en MC-Server. Contiene un jBean por cada uno de

los módulos que tiene la banca, por lo tanto, son demasiados para enumerarlos aquí, además de algunos comunes que si que pasamos a detallar a continuación:

- *SessionJBean: Es la clase que realiza la comunicación entre MC-Server y MC-Spy, y que por lo tanto, actua de nexo de unión entre las dos aplicaciones.*
- *BaseJBean;*
- *PaginationCache;*
- *PropApplication;*
- *Sequences;*
- *…;*

- Paquete *jfactory.mcserver2.rc.ejb.bl*: Paquete que contiene las clases de los *EJBs* necesarios para toda la aplicación. Como se sabe, cada *EJB* lo componen 3 clases, el *Home*, el *Bean* y el *Remote*. En este paquete además, añadimos para cada uno de los *EJBs*, una clase más: el *BLImpl* (*business logic*), en la cual insertamos todo la lógica de negocio, sacándola del *EJB*, para evitar que su ejecución lo cargue en exceso. Esta clase hace también de nexo de unión entre el *EJB* y la clase que accede directamente a la base de datos. Al igual que el caso de los JBeans, existen demasiados paquetes dentro de éste para exponerlos.

- *Paquete jfactory.mcserver2.rc.ejb.dal*: Paquete que contiene las clases de acceso a datos (DAL: Data access logic). Son realmente las clases que, a través del driver necesario para acceder a la base de datos de Oracle, ejecutan las sentencias SQL. Son también ellas las encargadas en transformar las respuestas que obtienen en instancias de objetos entendibles por el resto de las clases del proyecto. Lógicamente, existe una clase DAL por cada una de las tablas a las que accedemos en la base de datos. Como es fácil de suponer, la infinidad de tablas que pueden existir en la base de datos, hace imposible poder aquí detallar la lista de todo los DALS que existen dentro de este paquete.

- Paquete *jfactory.mcserver2.rc.util*: Este paquete contiene todas las clases que nos pueden servir de ayuda para la implementación del resto de clases. En este caso, aparte de las anteriormente mencionadas ya en el paquete *jfactory.mcserver2.util*, y que son las clases "útiles" que se utilizan tanto en el recolector de datos como en las consultas, se utilizan también las siguientes clases exclusivamente:

  - LogEntreQueue: Es la clase encargada de gestionar la inserción de los datos recibidos desde MC-Server en la BD.

Se explicará posteriormente con más detalle las características de esta clase.

- *Date*: Clases para el manejo de fechas.

- *JCrypt*: Clase que contiene varios métodos e utilidades para la encriptación de datos.

- *Fast*: Clases para facilitar el manejo de datos dentro de objetos de datos destinados para ello. Por ejemplo se pueden usar:

    - *fastHashTable;*
    - *fastVector;*

- *Exception*: Clases con los distintos tipos de excepciones que puede manejar la aplicación:

    - *…;*

- *JMS*: Clases encargadas de gestionar el servicio de mensajes en colas.

- Paquete *jfactory.mcserver2.rc.util.sa* : Este paquete contiene clases e interfaces necesarios para almacenar información, y transportarla a lo largo del flujo de invocación entre las clases implicadas en éste. Ya explicado anteriormente en este mismo capítulo su funcionamiento dentro de esta arquitectura.

C.2 Implementación del sistema de inserción en la BD

Si había un aspecto trascendental en el diseño y la implementación de este proyecto, y cuyo buen fin podía ser fundamental para el correcto desarrollo de toda la aplicación, fue la decisión y la construcción del sistema para insertar los datos que el recolector va recogiendo de la banca. Era importante, ya que un mal sistema podría echar abajo varios de los aspectos más importantes que una banca electrónica tiene que ofrecer a sus clientes: fiabilidad en los datos de los clientes, disponibilidad y respuesta inmediata a los usuarios.
Respecto al primer aspecto, la fiabilidad de los datos, era importante que los datos almacenados en nuestra BD (para ser mostrados luego en forma de estadística) correspondieran con la realidad, sobre todo en lo relativo a datos básicos (como importes, datos bancarios), como a tiempos (fechas). De todas formas, no era éste el aspecto más problemático, ya que si esto se podía controlar fácilmente en la fase de pruebas, con un protocolo lo suficientemente completo y con su ejecución óptima, además de la utilización de un gestor de BD que de confianza (como se hacía en este caso).
Respecto al segundo aspecto, es muy importante que siempre que un usuario se conecta a una banca electrónica, ésta responda y no de un error, ya que da una imagen lamentable al cliente, ofreciendo poca seguridad al mismo. Y no olvidemos que lo que está haciendo dicho cliente tiene que ver con temas económicos, por lo tanto, en todo momento una banca tiene que dar imagen de seguridad. Así pues, tenemos que construir un sistemas, que por muchas inserciones que se quieran hacer en la BD relativas a la información de la utilización de la banca por parte de los clientes, esto haga que ni la BD en sí, ni la banca se puedan caer, aunque el número de inserciones sea abrumador (imaginemos las operaciones al minuto que se pueden hacer en una banca electrónica por muy pequeña que sea ésta).
Respecto al tercer aspecto, una respuesta rápida por parte de la banca electrónica puede ser también fundamental para la imagen del banco de cara al cliente. Si un usuario usa los métodos electrónicos (sobre todo al principio de la aparición de estos, quizá en la actualidad esta decisión no sea tan fundamental) es por ahorrarse las típicas colas que suele haber en las oficinas de los bancos, y la parsimonia de algunos de sus empleados. Por ello, es importante que la inserción de todos estos datos, no ralentice el resto de la ejecución de la aplicación, y sea para ésta totalmente transparente.

Para cumplir todas estas necesidades, se pensó en un sistema que no supusiera una carga para la ejecución de la banca, y que no interfiriera en el desarrollo normal de su normal realización.

Y teniendo en cuenta todo ello, se implementó un sistema de almacenamiento de datos en la BD, bajo los siguientes dos conceptos:

C.2.1 Singleton

Se utilizó un patrón Singleton para conseguir que hubiera una única instancia de clase que pudiera iniciar la inserción en la BD desde el módulo de recolección de datos (y por lo tanto, el único acceso posible a la BD de la banca electrónica desde MC-Spy). Este patrón parecía el más óptimo ya que ayudaba a conseguir el hecho de que no se saturara a la BD con múltiples inserciones simultáneas (además al ser conscientes de que, siendo realistas, estos datos que se insertan por parte de MC-Spy no son fundamentales para el normal desarrollo de la banca), ya que esta clase controlaría únicamente, gracias al Singleton, la forma de realizar dichas inserciones (esta forma la veremos en el siguiente apartado).
La clase que se definió como un Singleton fue la clase anteriormente mencionada LogEntryQueue.

Pasemos a ver un poco más teóricamente como funciona un Singleton.
Un Singleton en una de las técnicas existentes para asegurar que nunca se crea más de una única instancia de una clase. En esencia, la técnica tiene el siguiente enfoque: no dejar que nadie fuera de la clase cree instancias del objeto y proporcionar un punto de acceso global a ella. Cuando se invoca a dicha clase, se (auto)comprueba si ya existe una instancia, y en caso de que no sea así, se crea. En caso contrario se utiliza la instancia ya creada. Con este tipo de patrones, también se ahorra memoria. Por lo tanto, el Singleton siempre nos proveerá una única instancia de esta clase, basándose en 3 aspectos:

- La misma clase es la que se encarga de crear la única instancia.
- El patrón Singleton permitirá el acceso a la única instancia a través de un método de la misma clase.
- El patrón Singleton obliga a declarar el constructor con privado para que no se pueda instanciar fuera de él.

Generalmente, la implementación de una clase Singleton se realiza de la siguiente forma:

- Se crea una variable privada con el mismo nombre de la clase.
- Se privatiza el constructor.
- Se define un método público (generalmente llamado getInstance) que devuelve la instancia de la clase. Dentro de éste se comprobará si ya existe dicha instancia o hay que crearla.

C.2.2 JMS

Aunque consigamos que una única instancia de una clase se encargue de realizar la gestión de las inserciones en la BD, siempre se puede seguir teniendo el problema de que una mala implementación de dicha gestión, pueda seguir echando abajo la BD o ralentizando los procesos. Es por lo que se implementó un sistema de colas mediante

JMS para terminar de conseguir ese óptimo funcionamiento que se requería anteriormente.
En este caso se utilizó un modelo JMS punto a punto, ya que únicamente teníamos un cliente que publicaba información en las colas (se podría decir que es MC-Server), y por otro lado otro cliente consumidor de dicha información (igualmente se podría decir que es MC-Spy).
Este modelo garantiza que la información siempre va a llegar a su destino, ya que en caso de que el proceso destino no esté disponible (recordemos que en este caso, es muy complicado que el cliente proveedor de información no esté disponible, ya que se trata de la misma banca electrónica, lo cual es de esperar que nunca pase. Sin embargo, si MC-Spy estuviera en otro servidor u otra instancia, si que podría ser que no estuviera disponible temporalmente, y no sería una tragedia para el funcionamiento del proyecto global), se almacenarían los datos en una cola FIFO, y cuando estuviera disponible los iría consumiendo según hubieran entrado.
Realmente, JMS no es más que un API que permite a las aplicaciones crear, enviar, recibir y leer mensajes, de forma síncrona o asíncrona y de forma totalmente segura. Son imprescindibles tres requisitos para considerar un modelo como JMS:

- El proceso proveedor de información no quiere que los interfaces de la información dependan del proceso destino(o consumidor).
- El proceso proveedor de información no quiere depender de si los procesos consumidores están ejecutándose o no a la vez que él publica información.
- El proceso proveedor de información quiere publicar datos sin tener que esperar a una respuesta por parte del proceso consumidor.

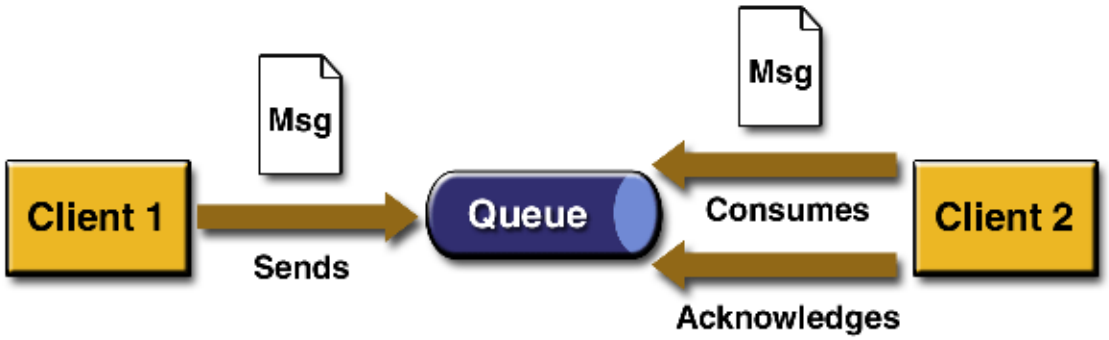


Figura 31- Esquema funcioamiento JMS

Para construir este modelo JMS, es necesario en primer lugar configurar la queue, a través del API de JNDI (Java Naming and Directory Interface). Una vez creada la queue, hay que crear tanto los clientes proveedores como los clientes consumidores. Ambos se tendrán que crear la conexión a la queue y una sesión, y en el caso del proveedor, tendrá que conectarse a ella y enviar el/los mensaje(s), indicándole cuando termina el final de esos mensajes. Esos mensajes permanecerán en la queue hasta que un

cliente consumidor se conecte a ella, y obtenga los mensajes que ha dejado el otro proceso.
Luego ya, será este mismo proceso, u otro al que se lo encargue, el que decida que hacer con dichos mensajes.

C.3 Implementación y algoritmia para las previsiones

C.3.1 Detección del tipo de serie de datos

ACLARACIÓN: Antes de nada, el autor de este proyecto quiere aclarar que no ha sido su intención el realizar un profundo estudio de las previsiones de datos económicos, ni quiere hacer parecer que el presente proyecto y la presente memoria es un máster en predicción de datos. Simplemente, viendo el posible interés que podría tener en el mercado para el que iba a ser utilizado MC-Spy,y partiendo de los conocimientos adquiridos en las asignaturas de Estadística durante la carrera, ha querido hacer una breve iniciación en este tema. Por lo tanto, seguramente haya métodos mejores y mas exactos para conseguir las predicciones deseadas. Lo que ha hecho el autor de este proyecto es escoger los métodos y los algoritmos que, dentro del alcance de su conocimiento y del tiempo material del que disponía para realizar este proyecto, creía que iban a darle un mejor resultado, siempre dentro de los modestos objetivos de MC-Spy.
También se sobreentiende que las series que se utilizan, son "normales", es decir no son series muy complicadas que no siguen ninguna norma lógica. Este tipo de series no se podrían estudiar con los métodos aquí utilizados, y serían objeto de una ampliación de este proyecto.

Lo primero de todo, establezcamos lo que entendemos por Serie de datos: Conjunto ordenado de observaciones de una variable, comúnmente registradas a intervalos de tiempo constantes.

Si queremos realizar una previsión partiendo de una serie de datos, para poder predecir los valores de los siguientes datos a los ya aportados, necesitamos en primer lugar conocer que tipo de serie es. Para ello, se pueden usar diversas formas y métodos, pero una de la mas común es clasificarla según dos características que puede tener la serie en cuestión:

- Tendencia
- Estacionalidad.

Entendemos por tendencia el comportamiento que la serie puede tener a largo plazo. Un ejemplo básico de una serie con tendencia sería la de la población mundial. Si tomamos datos anuales de la población mundial desde el año 1 DC, por ejemplo, tendremos una serie con una clara tendencia creciente. Eso no implica que año a año haya tenido que aumentar obligatoriamente (por ejemplo, años de terribles pestes o de guerras mundiales, la población habrá bajado), pero la tendencia de la población durante este periodo de tiempo ha sido siempre creciente.
Estacionalidad, es el comportamiento cíclico dentro de un periodo de tiempo. Un ejemplo común y recurrente puede ser el número de parados en España. En verano y en otros periodos, como Navidades o Semana Santa, el desempleo suele bajar, pero no así durante el resto del año. Esto hace que esta serie de datos a lo largo del año, tenga una clara estacionalidad en esos periodos.

Para detectar ambas características en una serie de datos, se escogieron dos contrastes distintos:

- Contraste de Daniel(o de correlación de rangos de Spearman): Desarrollado por Charles Spearman en 1904, y que se utiliza, entre otras cosas, para contrastar la existencia de tendencia en una serie temporal. Se basa en una doble ordenación de los datos de la serie: por un lado la ordenación natural, y por otra la ordenación de la serie de menor a mayor. Así cada dato tendría 2 rangos. Basándose en el sumatorio del cuadrado de la resta entre ambos rangos, y comparando el valor final con una distribución normal N(0,1), se obtendrá si la serie tiene o no tendencia, siempre aplicando a dicha distribución normal un nivel de significación determinado, que hará afinar mas o menos la afirmación de la existencia de tendencia.

- Contraste de Kruskal-Wallis: Desarrollado por W. H. Kruskal y W. A. Wallis en 1952, y que se puede usar, entre otras aplicaciones, para contrastar la existencia de estacionalidad en una serie temporal. Basándose también en una doble ordenación de los datos, al igual que el anterior Contraste, realiza en este caso una compleja operación para comparar su valor el de una distribución chi-cuadrado, obteniendo la afirmación o negación de la existencia de estacionalidad en la serie.

C.3.2 Algoritmos para cada uno de los tipos de series

Una vez que ya conocemos la existencia o inexistencia de tendencia y estacionalidad en una serie, ésta se podría clasificar siempre dentro de una de las siguientes opciones posibles:

- Serie sin tendencia ni estacionalidad.
- Serie sin tendencia con estacionalidad.
- Serie con tendencia sin estacionalidad.
- Serie con tendencia y con estacionalidad.

C.3.2.1 Serie sin tendencia ni estacionalidad

Para obtener la predicción de los futuros datos de una serie de este tipo, utilizaremos el **método del Alisado exponencial**. Dicho método define la predicción mediante una suma ponderada de todos los valores previos de la serie al periodo para el que se formula la predicción. Esta ponderación se basa en que se va haciendo menor cuanto más alejada está la utilización de un dato, es decir, a la hora de realizar los cálculos para predecir un valor determinado, se le da más importancia a los últimos valores, que a los anteriores. Tiene el inconveniente de que da un único valor para todas las predicciones futuras (algo razonable si pensamos que es una serie que no tiene ningún tipo de

característica lógica que nos haga predecir cómo va a ser esa serie en el futuro). Por otro lado, de los métodos existentes para este tipo de series, es de los que mejores resultados aporta.

C.3.2.2 Serie sin tendencia con estacionalidad

Para las series de este tipo se ha utilizado el **método de Medias estacionales**. Es un método de estructura fija que define la predicción para cada periodo a partir de la media muestral de los períodos con idéntico componente estacional. Es decir, si por ejemplo tenemos datos trimestrales durante varios años, hacemos la media de todos los datos de los primeros trimestres, y eso será lo que utilicemos como predicción para los datos futuros de esos periodos. Igual se hará con el resto de periodos (o estaciones).
Puede parecer un método bastante sencillo y no tener demasiada precisión, pero dentro de los métodos para este tipo y que sean implementables, era el que mejor resultados daba.

C.3.2.3 Serie con tendencia sin estacionalidad

En este caso, usaremos el **método de Alisado exponencial lineal de Holt**, para obtener las predicciones de las series de tipo 3. Es un método de estructura variable que asume que la tendencia es localmente lineal. Por lo tanto, la predicción está basada en una actualización de la estimación de la tendencia y la pendiente para cada periodo muestral, conforme se incorpora nueva información. Para ello, se utilizan todos los valores previos de la serie y no solo un número reducido de ellos, como se hace en otros métodos.
Con la estimación de la tendencia, veremos si la serie de datos tiende a ser lineal, o por el contrario por ejemplo, sigue una tendencia cuadrática. Con esta estimación, junto con la pendiente (como de rápido va esa tendencia), podremos obtener una previsión de los valores bastante fiable.

C.3.2.4 Serie con tendencia y con estacionalidad

Para este último tipo de series, utilizaremos el **método de Alisado exponencial de Holt-Winters**. Éste es un método de estructura variable basado en la reestimación a lo largo de todos los períodos muestrales de la tendencia, la pendiente y el componente estacional mediante unas ecuaciones de actualización. La predicción en un periodo se formula en términos de los valores estimados de estos tres términos en el periodo en que se formula la predicción. Como se puede suponer viendo el nombre del método, está

íntimamente relacionado con el método del anterior tipo de series. Por lo tanto, además de la tendencia y de la pendiente, se añade ahora el cálculo de otra componente que es el componente estacional (éste último sería algo así como el valor de lo que se separa cada uno de esos periodos de la media de la serie).
Este método, comparándolo con otros similares y también utilizados para este tipo de series (como por ejemplo el método de Tendencia lineal con variables ficticias o el método de Descomposición), obtiene unos resultados similares a los demás, pero se adapta mucho mejor a los algoritmos programables que aquí podríamos utilizar e implementar.

C.4 Implementación de aspectos gráficos

C.4.1 Árbol de sesiones

Para mostrar debidamente el árbol de sesiones necesitamos dotarle de dinámica, como en todos los elementos gráficos en forma de árbol que pueden aparecer en una página web.
Para ello se utilizó HTML dinámico, modificando el aspecto de la página de forma dinámica en base a eventos javascript previamente definidos (abrir una rama, cerrar una rama, etc…).

Figura 32- Árbol de sesiones

C.4.2 Estadísticas gráficas

Para dibujar las gráficas (tanto en formato de tarta como de barras), se implementó un sencillo applet que se integraba dentro de la página jsp, y que dibujaba la estadística correspondiente. Para ello se le pasaban como parámetro el tipo de gráfica que se trataba, y todos los valores que se debían emplear en la gráfica. Así obtendríamos gráficas como la que aquí se muestra a continuación.

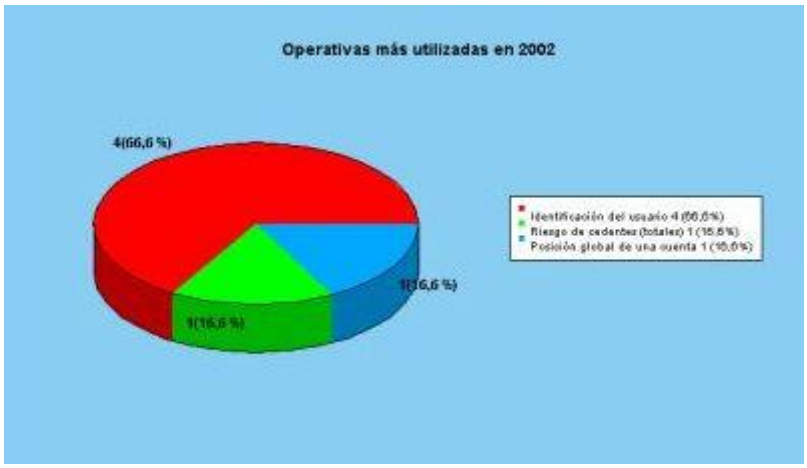


Figura 33- Gráfica de una estadística

Anexo D. Pruebas

La fase de pruebas nos permite comprobar el buen funcionamiento de los algoritmos utilizados durante la implementación, y detectar si cumplen con los requisitos inicialmente planteados.
Se realizaron durante el ciclo de vida del proyecto dos tipos de pruebas: pruebas unitarias (probando cada uno de los módulos durante y tras su implementación), y pruebas de integración (cuando dichos módulos se integraban unos con otros para formar la aplicación).
Como se ha explicado anteriormente en la memoria, para este proyecto se utilizó un modelo incremental, por lo que para cada módulo se realizaron pruebas unitarias que certificaban su completitud, y una vez que se finalizaron todos los módulos habiendo pasado dichas pruebas, se realizaron las pruebas de integración, ya dentro de la fase de Modo Operacional.

D.1 Pruebas unitarias

Se realizaron pruebas unitarias en la finalización de la implementación de cada uno de los módulos para el aseguramiento de la calidad y de la afinidad con los requisitos iniciales.
En este proyecto, como se ha comentado en los apartados del Análisis y el Diseño, se identifican dos grandes módulos, a los cuales se les aplicaron etas pruebas.
Se realizaron pues pruebas unitarias en el Módulo de Recolección de Datos, asegurándose que los datos insertados en la BD, eran exactamente iguales a los que el usuario había insertado en la web de la banca electrónica, y que los datos que se generaban automáticamente (por ejemplo fechas y tiempos), coincidían con la realidad de la acción. Para ello se realizaban consultas a la BD, comprobando la verisimilitud de esta información. Usando pues un entorno de pruebas de la banca electrónica, se generaban datos, que luego eran consultados (mediante SELECTS directas a la BD) para su comprobación.
Respecto al otro módulo, el de Consultas, para la fase de pruebas se identificaron tres submódulos sobre los que ejecutar las pruebas unitarias. A saber:

- Árbol de sesiones.
- Estadísticas
- Previsiones.

Dada la criticidad de cada uno de ellos, se prefirió dividir estas pruebas para cada uno de los submódulos, para así cerciorarnos de manera más definitiva de su correcto funcionamiento.
Para el árbol de sesiones, se preparó una batería de pruebas de usuarios que entraban en la banca electrónica, y que realizaban diversas operativas (también incluyendo usuarios que no hacían nada, por ejemplo, o usuarios que repetían constantemente una operativa, es decir, casos raros para ver si el árbol de sesiones funcionaba correctamente en todo tipo de situaciones y bajo todas las posibles casuísticas), y se comprobaba que la

información que se mostraba en las ramas del árbol correspondía con lo ejecutado anteriormente en la banca.
En el caso de las estadísticas, se probaron una a una todas las estadísticas disponibles, y para comprobar que los datos eran ciertos, se cargaron previamente en la BD datos con varias posibilidades variadas (por ejemplo que no hubiera datos, que solo hubiera uno, que hubiera múltiples), para ver cómo se comportaba la estadística, ya fuera gráfica o en texto. A destacar en este punto que debido a la integración con el componente que dibujaba los gráficos (de tarta, de barras, etc…), hubo varios casos de errores en la muestra en pantalla de las gráficas, sobre todo a la hora de tratar datos especialmente singulares (como por ejemplo y como ya se ha comentado, el que no hubiera ningún dato para mostrar o bien que sólo hubiera uno y no se mostrara correctamente en el gráfico de tarta). En el apartado XXXX, se explica la respuesta ante este tipo de situaciones.
En el caso de las previsiones se trata de un apartado especial, ya que no solo influyen los datos que hay almacenados en la BD, y que son con lo que trabajarán para obtener las previsiones, sino que también influye (y mucho en este caso) los algoritmos que construyen las series de tiempo previstas. Ante la dificultad de poder comprobar o bien visualmente o bien mediante consultas a BD la verisimilitud de dichas previsiones, lo que se hizo fue preparar varias series de datos, y construir "a mano" el resultado con la serie de tiempo que tendría que obtenerse. Gracias a este método se obtuvieron diversos errores en la implementación de los algoritmos.

D.2 Pruebas de integración

Para hacer pruebas de integración, y dado que los dos módulos están separados pero íntimamente relacionados, se decide que la mejor forma de planificarlas y ejecutarlas es crear un usuario totalmente nuevo en la banca electrónica, y con él, iniciar una serie de consultas en el módulo de Consultas. Posteriormente, y habiendo dicho usuario entrado ya a la banca y habiendo ejecutado varias operativas, se comprobaban que los datos que se mostraban en las Consultas correspondían con lo realizado en la banca (y por lo tanto con lo insertado a través del módulo de Recolección de Datos); con lo cual se podía comprobar conjuntamente el correcto funcionamiento de ambas partes de la aplicación MC-Spy.
Durante varias jornadas se estuvieron realizando dichas pruebas para comprobar y asegurar el correcto funcionamiento de ambos módulos integrados en una sola aplicación.

D.3 Respuesta ante un error detectado

Si a lo largo de las pruebas unitarias se encontraba un error en la implementación de alguno de los módulos de la aplicación, la forma de actuar era simple. Tal y como muestra el apartado 5 de la memoria de este proyecto, y siguiendo el modelo incremental, cuando se encontraba este error, se intentaba corregir en el código, y se comprobaba si era debido a un error de implementación o de diseño.

En caso de que fuera de éste último, se revisaba el diseño para corregir lo que fuera necesario, y se volvían a ejecutar las pruebas unitarias de ese módulo.
En caso de que fuera únicamente de implementación, se corregía y se pasaban de nuevo las pruebas.

Durante las pruebas de integración, la respuesta era similar, pero en este caso, incluyendo todos los módulos en la ejecución de las pruebas tras la búsqueda y la solución del error.

Índice de figuras