ESTUDIO, ANÁLISIS Y MEJORA DE UN SISTEMA DE GESTIÓN DE EXPEDIENTES

Memoria del proyecto
de Ingeniería en Informática
realizado por
David Lozano Romera
y dirigido por
Carles Ferrer Ramis
y por
Jaume Miralles i Fàbregues

Bellaterra, 16 de Junio de 2008

El abajo firmante, Carles Ferrer Ramis

Profesor de la Escuela Técnica Superior de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al qué corresponde esta memoria ha sido realizado bajo su dirección por David Lozano Romera

Y para que quede constancia firma la presente.

.....

Firmado: CARLES FERRER RAMIS

Bellaterra, ...de.....de 200.....

El abajo firmante, Jaume Miralles i Fàbregues

de la empresa, Steria Ibérica S.A.U.

CERTIFICA:

Que el trabajo al qué corresponde esta memoria ha sido realizado en la empresa bajo su supervisión mediante convenio de colaboración firmado con la Universitat Autònoma de Barcelona.

Así mismo, la empresa tiene conocimiento y da el visto bueno al contenido que se detalla en esta memoria.

.....

Firmado: JAUME MIRALLES I FÀBREGUES

Bellaterra, ...de.....de 200.....

Me gustaría expresar mi agradecimiento a todas las personas que finalmente han hecho posible el desarrollo de esta memoria y el proyecto en el cual se enmarca. Muchas gracias.

En primer lugar a los que directamente han participado, a Jaume Miralles por el seguimiento y ayuda durante estos meses, a Carles Ferrer por los consejos para acabar esta memoria. También a mis compañeros en Steria: Gerard y Manel, así como a Felix, Bernabé y Xavier.

Por supuesto, también querría agradecer a los que me han ayudado no sólo estos últimos meses, sino durante todo el periodo de formación en la universidad y en el que desafortunadamente, he tenido que hacer ciertas concesiones. Gracias a la familia y amigos.

Especialmente dedicado a todos los que hace más de 6 meses con los que no he podido coincidir y de los cuales me acuerdo a menudo aunque no lo parezca.

Muchas gracias.

Firmado: David Lozano Romera

Índice

Índice de figuras

Índice de Tablas

1 Introducción

1.1 Objetivos

El principal objetivo es aumentar el grado de parametrización o abstracción de las diferentes partes de un sistema para poder adaptarlo con mayor facilidad a las necesidades de los clientes. Para conseguirlo, durante el desarrollo del proyecto se propondrán una serie de problemas para que el alumno los analice e intente resolverlos, para finalmente llevar a cabo la implementación de un conjunto de los mismos. Además de realizar tareas de parametrización, se podrán realizar mejoras sobre otras partes.

El último objetivo, y no por ello menos importante, será personal del alumno, y consistirá en adquirir conocimientos y experiencia suficiente que le ayude a inicial su carrera profesional.

Se parte del proyecto *SEDAS*, el cual ha sido personalizado para diferentes clientes. Este proyecto ha alcanzado cierto grado de parametrización pero es necesario aumentarlo de cara al futuro.

El sistema está diseñado como una aplicación web cliente-servidor con diferentes capas, en las que la lógica de control y las interfaces de usuario se generan sobre Java utilizando *Apache Tomcat y Struts, mientras que* la capa de modelo y lógica de negocio está implementada sobre *Oracle*.

Está arquitectura puede sufrir modificaciones, sobre todo en la capa de lógica de negocio, debido a restricciones del cliente ya que puede preferir conservar una o varias partes del sistema que disponía anteriormente, o simplemente por razones de coste, usabilidad, etc.

1.1 Marco de trabajo

1.1.1 Empresa

Líder en Europa de servicios globales de *TI* para organizaciones que consideran las nuevas tecnologías parte vital de su estrategia de negocio, Steria se focaliza en establecer con sus clientes relaciones de confianza en mercados claves como: sector público, finanzas, telecomunicaciones, energía, transportes e industria. Steria ofrece servicios integrados que incluyen consultoría de sus

procesos de negocio críticos y desarrollo y gestión de sus Sistemas de Información. El nuevo Grupo posee más de 18.000 empleados en 16 países, incluyendo 5.000 en la India. Los ingresos de Steria a 31 de diciembre de 2007 se cifran en 1.400 millones de euros. Con sede principal en París, Steria cotiza en la Bolsa de París (Eurolist).

Steria en España cuenta con una consolidada experiencia en Consultoría, Integración de Sistemas y Servicios de Outsourcing. Con más de 1.000 empleados y una amplia cobertura geográfica en todo el país gracias a sus 9 delegaciones comerciales y centros de servicio, Steria ofrece soluciones de negocio en sus ámbitos de especialización en el mercado español sector público, sanidad, banca y seguros, industria, energía y transporte.

1.1.2 Convenio

La Universitat Autònoma de Barcelona (UAB) realiza convenios de prácticas para sus alumnos con las empresas de su entorno. El objetivo de estos convenios es iniciar un marco de colaboración para desarrollar distintas iniciativas como el desarrollo de proyectos de final de carrera. La duración de los proyectos aproximada es de unas 450 horas cursadas dentro de las empresas con posibilidad de ampliarlas hasta un periodo máximo de 900 horas.

Cada proyecto tiene asignado un tutor, o varios según el proyecto, que actúa como jefe del proyecto. Gracias a estos convenios los alumnos pueden adquirir conocimientos y experiencia en empresas de desarrollo de software de gestión y tecnologías de la información.

Este proyecto se ha realizado en el marco de un convenio entre Steria Ibérica S.A.U. (Steria) perteneciente al grupo Steria y la Universitat Autònoma de Barcelona (UAB) y su objetivo principal es que los alumnos realicen el proyecto final de carrera.

1.1.3 Departamento de trabajo

El proyecto se desarrollaría de forma independiente, pero con posibilidades de contacto y consulta con la decena de personas que actualmente se encargarían de diversos desarrollos del proyecto *SEDAS*. El departamento pondría a disposición del alumno todo el software, servidores y documentación necesaria para poder terminar con éxito el proyecto.

Además, se trabajaría diariamente con otros estudiantes proyectistas encargados de otras tareas, por lo que se aprenderían temas de diversa temática no estrictamente relacionadas con el proyecto *SEDAS* o los gestores de expedientes.

1.2 Motivaciones personales

El principal motivo para la realización de este proyecto es el de tener una primera experiencia justo antes de empezar la vida laboral. Además, tener la posibilidad de realizar el proyecto en una gran empresa multinacional como *Steria* podría ser un importante valor añadido.

También se desea observar como se realiza el trabajo, como se organizan los grupos de trabajo y como es la arquitectura de una gran aplicación web, ya que se tiene interés en este tipo de desarrollos.

Además, trabajar en el ámbito empresarial, podría ayudar y ser una motivación extra de cara a realizar un buen proyecto y tener experiencia de cara a la incorporación a la misma empresa o otra del mismo sector.

1.3 Contenido de la memoria

La memoria se ha estructurado en 8 capítulos más un anexo en al cual, a modo de glosario, se definen o explican todos los términos no comunes utilizados en este documento. El contenido de cada uno de los apartados es el siguiente:

- Introducción

  El presente capítulo en el cual se da una explicación sobre los objetivos generales, el marco de trabajo y el contenido de esta memoria.

- Planteamiento

  El apartado esta dedicado a exponer las partes previas al análisis de la situación, haciendo una exposición de los recursos disponibles, la planificación inicial de la cual se parte así como los costes y riesgos de su realización.

- Análisis

  Se incluye una descripción del sistema a partir del cual se trabajará haciendo una explicación de su arquitectura y una serie de conceptos sobre los cuales se tuvo que documentar en un inicio. La parte central está dedicada al estudio de cada una de las propuestas que fueron surgiendo durante el proyecto, haciendo finalmente un estudio de los resultados para determinar cuales serian las que finalmente se implementarían en fases posteriores.

- Diseño

  Este apartado está dedicado a explicar detalles relativos al diseño de cada una de las mejoras seleccionadas en el capítulo anterior.

- Implementación y pruebas

  El capítulo se centra en exponer los detalles clave relativos a la codificación y pruebas de las mejoras diseñadas previamente, sin por ello entrar en detalles excesivos que redunden en aspectos poco importantes del mismo.

- Conclusiones

  En este caso se mostrará como la planificación se ha ido modificando viendo las desviaciones producidas y se hará un balance del desarrollo del proyecto así como se explicarán algunas de las líneas de continuación establecidas.

- Referencias

  Finalmente se mostrarán algunos documentos publicados que son fuente de información para temas tratados en el proyecto.

- Anexos

  Se incluye un anexo consistente en un conjunto reducido de términos, abreviaciones y productos que se nombran durante la memoria para ilustrar de forma breve sobre su significado, usos, etc.

2 Planteamiento

2.1 Modelo de desarrollo

El modelo de desarrollo seguirá una pauta similar a uno clásico de cascada, excepto que inicialmente se hará un análisis de la totalidad de los diferentes cambios y mejoras propuestas, para después evaluarlos e implementar los más aptos. Por lo tanto, según este modelo, se realizará un análisis inicial y se seguirá el modelo en cascada para cada desarrollo.

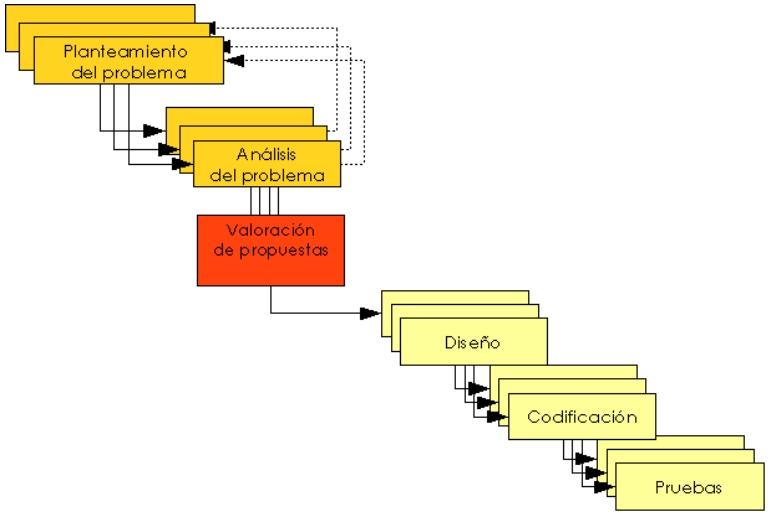


Figura 1: Modelo de desarrollo seguido

2.2 Recursos

2.2.1 Humanos

Los recursos humanos con los se contarán serán un becario / estudiante a tiempo parcial con una disposición temporal de 4 horas diarias realizando labores de analista/programador. Además se cuenta con la supervisión de un jefe de proyecto.

2.2.2 Materiales

En cuanto a los recursos informáticos y materiales se hará uso de los ya existentes en la empresa. Los posibles prototipos también se desarrollarán dentro de la plataforma de la empresa, por lo que no habrá necesidad de obtener otras

licencias tan solo para ello.

Para el desarrollo del proyecto se dispone de un *PC* con procesador *Intel Pentium 4* con 512 MB de *RAM* y *Windows XP*. También se tiene acceso a herramientas de ofimática, de mensajería y organización como *Lotus Notes*, de desarrollo como *Eclipse* o *TOAD* para interactuar con el SGBD, y todas las herramientas puramente de desarrollo mencionadas en la sección de documentación.

2.3 Planificación inicial

Es necesario presentar una planificación inicial ya que la duración del proyecto está muy bien delimitada por ser un proyecto de fin de carrera y tener unas fechas de finalización preestablecidas en la correspondiente convocatoria. Esta planificación sufrirá modificaciones durante el proyecto ya que será estrictamente necesario adaptarla al análisis realizado posteriormente, por lo que en las conclusiones de la presente memoria se detallarán los cambios principales y el motivo por el cual se tuvieron que hacer.

Antes de poder realizar una estimación previa del coste es conveniente presentar una planificación con los plazos de entrega, ya que están fijados por las convocatorias de entrega en la universidad.

Esta estimación puede variar sensiblemente dependiendo de los requisitos específicos, por los riesgos determinados posteriormente y sobre todo según que se decida analizar e implementar. Esta variación resultaría en una redistribución del tiempo dedicado a cada tarea.

Planificadas 468 horas realizando 4 horas de trabajo diario.

Fecha de inicio: Diciembre 2007

Fecha de finalización: Junio 2008

		Días
1	Planteamiento inicial	15
1.1	Planteamiento del problema a tratar y conceptos	
1.2	Objetivos	
1.3	Recursos	
1.4	Antecedentes y contexto	
2	Análisis y especificación de requisitos	25
2.1	Obtención de requisitos	
2.2	Análisis de requisitos	
3	Diseño	25
3.1	Diseño arquitectónico	
3.2	Diseño de datos	
3.3	Diseño de les interfaces	
3.4	Diseño detallado	
4	Implementación	40
4.1	Documentación sobre herramientas y plataformas a utilizar	
4.2	Codificación	
4.3	Pruebas finales	
5	Conclusiones y documentación	12
5.1	Conclusiones	
5.2	Generación de la memoria del proyecto	
	Total	117

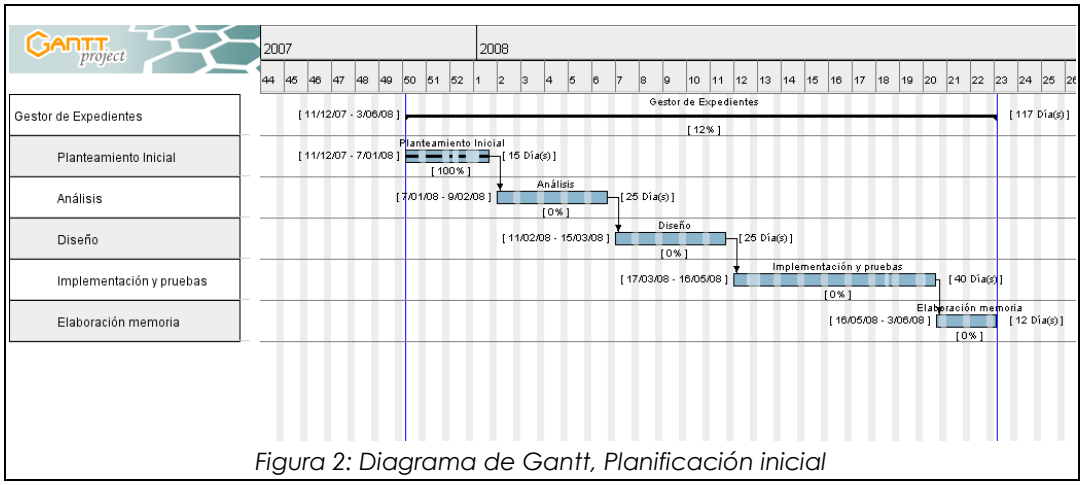


Figura 2: Diagrama de Gantt, Planificación inicial

2.4 Costes

Al ser un proyecto en parte académico, no se pueden tener en cuenta gran parte de los coste que supondría su desarrollo en otras circunstancias, pero siempre es necesario hacer una valoración para tener constancia de cuales serían en cualquier otro caso.

2.4.1 Mano de obra

Los costes se reducen a tan solo la beca del desarrollador ya que se utilizará infraestructura ya existente y no se tiene necesidad de ningún tipo de licencia de software adicional.

Para tener un coste orientativo sobre la mano de obra se calculará según el número de horas planificadas y el sueldo de un trabajador dedicado a cada tipo de tarea:

Puesto de trabajo	Coste/hora	Horas planificadas	Total
Analista	30€	65·4=260	7800€
Programador	20€	40·4=160	3200€
Documentalista	15€	12·4=48	720€
TOTAL			**11720€**

2.4.2 Coste de materiales y licencias

El resto de costes incluyen un ordenador con herramientas de ofimática, *TOAD* y software libre o gratuito. Se ha calculado un coste diario en el cual constaría el uso de toda la infraestructura que la empresa pone a disposición, además del ordenador también se incluye el coste de la conexión en red, los servicios de impresión, etc.

Se utilizará una licencia de *Oracle Enterprise* existente que tiene un coste importante, pero que está compartida entre aproximadamente 50 usuarios.

Concepto	Coste	Dias planificados	Total
Infraestructura de la empresa	6€/día	117	702€
Oracle Enterprise	40000€/50 usuarios	-	800€
TOTAL			**1502€**

2.4.3 Total

	Total
Mano de obra	11720€

Recursos materiales y licencias	1502€
TOTAL	**13222€**

2.5 Riesgos

Los riesgos existentes son muy limitados. Uno de ellos es el de no finalizar el proyecto en el límite de tiempo establecido, también se trabajará en todo momento de forma independiente, por lo que el riesgo de afectar a otros desarrollos es muy bajo.

El otro riesgo remarcable es no realizar el trabajo apropiado o que su resultado no sea útil ni para el alumno ni para la empresa.

2.6 Beneficios

Los beneficios académicos son muy numerosos. Se estaría en un entorno muy diferente al académico en que se ha desenvuelto el estudiante durante los años de formación, por lo podría adquirir ciertas destrezas y habilidades imprescindibles en el mercado laboral. Otros beneficios serían directamente derivados del aprendizaje y uso de un gran número de tecnologías con las cuales no se ha tenido la oportunidad de trabajar anteriormente.

Los beneficios para la empresa serán proporcionales a los resultados obtenidos en cada una de las fases, los cuales su vez dependerán en gran medida de las propuestas que se realicen.

3 Análisis

3.1 Descripción general del proyecto SEDAS

3.1.1 Perspectiva

El proyecto fue iniciado a finales de los 90 en forma de aplicación cliente-servidor. Durante la vida del proyecto se han realizados un gran número de cambios, desde pequeños hasta otros mayores, tales como el rediseño en forma de aplicación *web*. Desde 2003, cuando se realizó la primera implantación basada en aplicación *web* con *Struts* como marco de desarrollo, se ha seguido con el mismo modelo, mejorándolo progresivamente conforme surgían necesidades a los clientes o iniciativas internas de mejora de los procesos.

3.1.2 Funciones

SEDAS es un sistema que permite gestionar el proceso de recobro de deudas. La gran cantidad de funciones que realiza y como está estructurado, permitiría clasificarlo como un *ERP* financiero especializado en tareas de recobro.

Algunas de las características más destacadas son las siguientes:

- Esquema modular con un núcleo central complementado por diferentes partes según las necesidades específicas de cada entidad.
- Integración en el "*core business*" de la entidad dando servicio a todos los actores internos/externos del proceso (agencias bancarias, letrados, sociedades de cobro, ...).
- Herramienta abierta y modular que permite al usuario la definición del modelo de datos, las tablas de parámetros, los esquemas de actuación, ...
- Solución adecuada a los nuevos escenarios de mercado y a las nuevas normativas de medición global del riesgo.

Una de las partes más importantes es el motor de plantillas o flujos, hecho por *Steria* y personalizado para usos específicos financieros, que permite al cliente poder modelar cualquier requisito especifico que tenga en su proceso de negocio.

3.1.3 Usuarios

Los clientes principalmente son entidades bancarias, por lo que los usuarios son en prácticamente todos los casos, trabajadores con amplios conocimientos

financieros, así como gestores externos tales como abogados o entidades externas a la empresa que se dedican exclusivamente a la gestión de la morosidad.

Por lo tanto, en su mayoría, los usuarios no tendrán un perfil con grandes conocimientos sobre computación, pero si sobre el proceso de negocio que deben gestionar.

3.2 Arquitectura de SEDAS

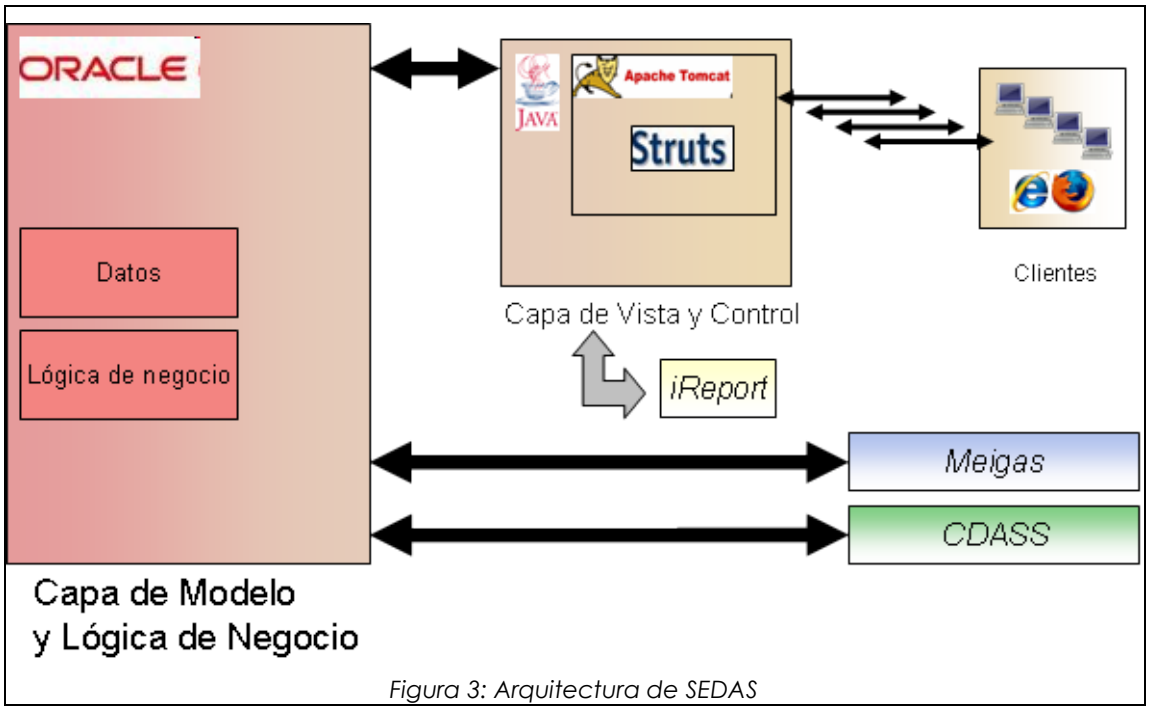


Figura 3: Arquitectura de SEDAS

El sistema está estructurado en tres partes, esto es debido a las diversas migraciones que se han ido haciendo, ya que inicialmente, todo el sistema estaba diseñado como aplicaciones de escritorio.

- Aplicación principal

Es un modelo en tres capas que puede ser catalogado como aplicación web. Implementa un patrón derivado del *MVC* donde la mayor parte del código está desarrollado directamente en el *SGBD Oracle*. Esto supone ciertas ventajas debido a la seguridad y a la facilidad de implantación, ya que el uso de *Oracle* está ampliamente extendido en las entidades bancarias, y estas son los principales clientes.

El disponer de una capa de presentación independiente permite adaptarse con mayor facilidad a las necesidades del cliente y reducir el tiempo de desarrollo.

Se utiliza *iReport* únicamente como herramienta de desarrollo y no se suministra al cliente final.

Las versiones de las principales librerías utilizadas en la última versión corporativa estable son las siguientes:

- *Oracle 9i*
- *JDK 1.4.2*
- *Apache Tomcat 4.1*
- *Apache Struts 1.1*
- Navegadores certificados
  - *Internet Explorer 5, 6 y 7*
  - *Mozilla Firefox*
- *JasperReports 0.5.1*
- *iReport 0.2.2*

También se utiliza *Exadel Struts Studio 4.7* como *IDE*.

Se han realizado pruebas satisfactorias haciendo uso de las últimas versiones de *Oracle 10g*.

- *Meigas*

Es una aplicación de escritorio que permite diseñar las plantillas o flujos. Su funcionamiento está basado en formularios y asistentes.

Al igual que *CDASS,* está programada para plataforma *WIN32*. Su desarrollo y mantenimiento se realiza mediante *Visual Studio 5.0*.

- *CDASS*

Es una aplicación de escritorio que permite crear y modificar los escritos. Estos documentos son plantillas en formato *RTF*, que permiten generar un documento final en tiempo de ejecución, enlazando una serie de datos con cada uno de los campos del escrito. Todo el proceso de combinación de plantillas con parámetros lo realiza el *backend* de *SEDAS,* y se puede acceder al resultado a través de la interfaz *web*.

Funcionalmente es una aplicación con algunas similitudes con *iReport* pero mucho más simplificada, de forma que permite diseñar los escritos de forma más sencilla y rápida. Esta aplicación accede a *Oracle* para gestionar las plantillas de los escritos.

3.3 Documentación

3.3.1 Introducción

La documentación y la explicación de los fundamentos teóricos fueron siendo introducidas progresivamente conforme se iba necesitando dadas las tareas que se planteaban. A pesar de que inicialmente ya se tenían algunos conocimientos sobre ciertos apartados técnicos, en muchos momentos sería necesario acceder a documentación oficial de la empresa, del software o consultar con *staff* del proyecto, siempre, por supuesto, intentando solucionar la mayor cantidad posible de problemas de forma autónoma, tal y como se espera de una persona que está en disposición de finalizar sus estudios.

3.3.2 Documentación sobre Gestión de Expedientes

La gestión de expedientes es el proceso de seguimiento de un determinado conjunto de ficheros a lo largo del tiempo.

En esta gestión podríamos remarcar una serie de conceptos:

- **Expedientes**

  Se puede entender como un conjunto de ficheros o información que va cambiando a lo largo del tiempo. También puede entenderse como un contenedor de información.

  En SEDAS se suele considerar a un expediente como un pago no realizado por una persona física o jurídica. Cada expediente tiene por lo tanto:

  - Cuentas

    Cuentas bancarias que dispone el titular del expediente.

  - Bienes

    Muebles e inmuebles que dispone el titular del expediente.

  - Personas

    Personas que están implicadas en un expediente, tanto los titulares como los posibles avalistas.

  - Liquidaciones

    Medidas y planes de pago que se toman para llegar a la resolución de un expediente.

- **Histórico**

  Conjunto de modificaciones que se han realizado a un documento en las que se especifica el autor de la misma, fecha, motivo, observaciones, etc.

- **Flujo**

  Movimiento de información que puede provocar el cambio de estado o la información de un expediente.

- **Plantilla**

  Conjunto de flujos que representan un escenario entre el cual puede verse modificado un expediente.

  Las plantillas se diseñan específicamente para cada cliente según las requisitos recogidos por las personas encargadas del área de consultoría. El motor que ejecuta los procesos y los desplaza por las plantillas es también propio de SEDAS.

  El resultado es el diseño de una serie de procesos *BPM* de diferente complejidad, que representarán las diferentes etapas por las que podría pasar un expediente y que se podrían visualizar como un grafo direccional.

- **Ciclos en el flujo**

  Un flujo que termina en un punto o estado en el cual ya había estado un expediente.

- **Puntos de decisión**

  Sección de una plantilla que puede direccionar el procedimiento de un expediente a diferentes flujos según un hecho determinado.

  En SEDAS existen puntos de decisión automáticos, p.e. comprueban una condición, esperan hasta determinado día antes de avanzas, etc. Manuales, los cuales deben de ser gestionados por un empleado. Así como los manuales con fecha de finalización, que pueden producir un cambio de estado aún sin la intervención de un empleado si se llega a una fecha o circunstancia concreta.

- **Escritos**

  Se denominan así a los documentos que se le hace llegar a una persona y que contienen información sobre algún hecho en concreto relacionado con su expediente, como por ejemplo algún cambio o hecho reciente.

3.3.3 Documentación técnica

Lenguajes de programación

El lenguaje de programación que se utilizará mayoritariamente será *Java*, por ser el principal en el cual está implementado *SEDAS*. Además también se espera utilizar ciertos conocimientos sobre *SQL* y *PL/SQL*.

Antes de comenzar el proyecto ya se tenían ciertos conocimientos sobre *Java* y *SQL,* pero no se había trabajado con la profundidad que sería necesaria por no tener experiencia laboral en el sector.

- *Java*
- *SQL*
- *PL/SQL*
- *ECMAScript/Javascript*

Lenguajes de marcado

Será necesario utilizar conocimientos acerca de HTML para poder comprender y proponer soluciones a algunos problemas, pero la complejidad y la experiencia previa hace que no sea necesario mucha documentación al contrario que sobre lenguajes de programación o librerías.

- *XML + DTD*
- *HTML*
- *CSS*

Plataformas

Estas plataformas librerías estándar con un conjunto de agrupaciones de clases que hacen más sencillo el trabajo con Java, incrementando de esta forma la productividad.

- Java J2SE
- Java J2EE

Servidores

Los servidores *web* serán los contenedores que ejecuten las aplicaciones y las pongan a disposición de los usuarios. Su funcionamiento es genérico y apenas se tendrían que tener conocimientos sobre las funcionalidades que proporcionan.

- *Apache Tomcat*
- *JBoss*

Librerías

En este caso si será necesario una amplia formación ya que muchas de la mejoras que podrían caber están potenciadas o por limitadas por las características de las librerías utilizadas.

La más importante *Struts* ya que en base a ella se diseño todo el sistema de SEDAS. Estos también conlleva que tengan que tener conocimientos sobre un número considerable de patrones de diseño de software que son utilizados con gran asiduidad (La mayoría del *GoF*, *MVC*, *Dispacher*, *Front controller*, *Composite View*, *Delegates*, *DAO's*, etc.

- *JSP y taglibs*
- *Apache Jakarta Struts*
- *Apache Ant*
- *JasperReports*

Herramientas

Se tendría que tener alguna breve formación sobre estas herramientas para poder utilizarlas durante la implementación.

- *Exadel StrutsStudio*
- *iReport*

3.4 Estado Actual

3.4.1 Descripción

Como comentábamos anteriormente, *SEDAS* ha visto modificada su arquitectura en diversas ocasiones, se disponen de versiones en forma de aplicación de escritorio, otras basadas en *Oracle IAS 10g* u otras similares a la actual pero con *Bea Weblogic* como servidor web o de aplicaciones.

Como todos los sistemas que recaban cierto éxito, se han tenido que realizar diversas implementaciones con cambios y mejoras sucesivas, pero desafortunadamente, esto ha conllevado que se tuviesen que conservar diferentes partes y que otras equivalentes se desarrollasen en paralelo.

Actualmente la empresa está abierta a estudiar todos los cambios que puedan suponer una mejora cualitativa al sistema, especialmente aquellas que supongan reducir el tiempo necesario para la adaptación a requisitos específicos.

A pesar de ello, como en cualquier proyecto que alcanza cierta madurez, se ha de realizar un amplio estudio de cuales serán las consecuencias de implementar un determinado cambio. Solo se tendría que actuar cuando los beneficios obtenidos de su implementación superen a los costes de desarrollarlo.

3.4.2 Diagnóstico

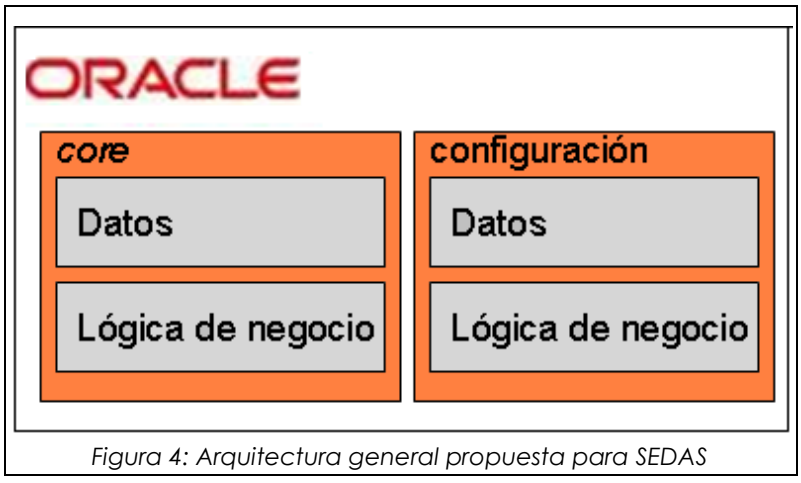


Figura 4: Arquitectura general propuesta para SEDAS

La aplicación tiene una buena arquitectura base que se ha visto beneficiada por el éxito y la solidez de las tecnologías utilizadas. No obstante, siguen quedando partes que podrían actualizarse en un momento dado por otras más actuales.

El reto más importante que resta por conseguir es separar el sistema en dos secciones. De esta forma sería posible disponer de un *core* o núcleo de la aplicación, al que mediante una sección de configuración sería posible configurar todos los aspectos que pudiesen variar según el cliente.

Esta separación también ayudaría a posibilitar los cambios de versión. Actualmente se debe mantener una rama de desarrollo independiente para cada cliente. Cuando una mejora concreta se desarrolla para un cliente puede necesitar muchos cambios antes de poder ser incorporada a otro. Esta propuesta puede facilitar en gran medida el mantenimiento ya que todas las mejoras pueden ser desarrolladas en una rama centralizada y tan solo configurar algunos aspectos puntuales según las necesidades específicas.

3.4.3 Alternativas

Dos ejemplos de cambios posibles en la arquitectura están representados en las aplicaciones *Meigas* y *CDASS*. Estas podrían adaptarse al uso en entornos web para reducir al máximo los requisitos de los clientes así como el tiempo de mantenimiento de las mismas.

Otro posible cambio importante sería el del marco de trabajo de la interfaz. Desde hace un tiempo han aparecido soluciones que mejoran y/o simplifican sustancialmente aspectos de las aplicaciones tales como la forma de organizar y mostrar las pantallas, la gestión de permisos, el test de cada uno de los elementos, la comunicación con el *backend*, etc. Entre los marcos de desarrollo que han extendido más últimamente se pueden resaltar *Spring/Spring MVC*, *Struts*[2], *JSF* o *Tapestry*.

Los cambios concretos se detallarán en el siguiente apartado.

3.5 Cambios en el sistema propuestos

3.5.1 Introducción

Las mejoras se pueden agrupar según el origen de la propuesta. Inicialmente el responsable en la empresa sugirió un conjunto de interesantes líneas de actuación para su estudio y análisis. Junto con estas, conforme se iba conociendo en mayor profundidad el sistema, fueron surgiendo otras alternativas o que completaban a las otras.

Entre estos dos conjuntos se seleccionó algunas de las más apropiadas para realizar su implementación dada la solución propuesta, los recursos del proyecto y la doble vertiente, tanto académica como empresarial, del presente proyecto.

3.5.2 Cambios

3.5.2.1 Cambios en el modelo de datos

- Introducción

El modelo de datos puede necesitar ser modificado frecuentemente ya que casi todos los clientes exigen algún cambio. Estos cambios pueden ser respecto al modelo de datos existente (p.e. necesidad de guardar más información de una entidad en concreto) o en la lógica de negocio (p.e. coeficientes y constantes a utilizar en las formulas) o conceptos generales (p.e. podemos entender a un expediente como una deuda que afecta a una persona o una deuda de una cuenta bancaria)

En primera instancia el análisis se centrará en encontrar formas de soportar cambios en el modelo de datos.

Estas aproximaciones no tienen en cuenta las repercusiones que tendrían en las capas externas a *Oracle*, sino que se centran en el *SGBD* por ser la más importante y la que más implicaciones tiene sobre el funcionamiento y mantenimiento del sistema dado su arquitectura.

- Objetivo

Poder incorporar campos extra a las tablas, es decir, facilitar el proceso de adaptación a las necesidades del cliente según sus requisitos específicos en cuanto a los modelos de datos.

- Restricciones

Se debe hacer de la forma más transparente posible debido al número de dependencias. Se han de considerar los cambios que seria necesario realizar a las funciones *PL/SQL*.

- Métodos / Herramientas
    - Añadir una tabla extra por cada una de datos

Separar un conjunto limitado de campos que serán los comunes a todos los clientes. Se tendrá una tabla con los datos estándar y otra con los personalizados para el cliente concreto.

Ventajas

- Portable entre bases de datos
- Esquema muy simple
- Clara separación entre esquema 'core' y de configuración
- Se puede modificar el sistema de forma gradual

Desventajas

- Prácticamente se duplica el número de tablas a gestionar
- Se debe modificar gran parte del código existente para adaptar completamente el sistema.

    - Herencia de tablas 'directa'

Esta propuesta utilizará el soporte a la orientación a objeto que *Oracle* ha ido introduciendo en su *SGBD* desde la versión 8. Se crearán tipos de datos con relación de herencia entre sí y tablas que contengan registros de estos tipos.

Un ejemplo sería tener una tabla 'expediente_personalizado', en relación de herencia con 'expediente', y que contendrá los campos extra específicos.

```
create table expediente of Expediente
create table expediente_empresaXX of Expediente_empresaXX under expediente
create table expediente_empresaYY of Expediente_empresaYY under expediente
```

Ventajas

- El esquema es transparente, se puede seguir utilizando la tabla base conservando prácticamente todo el código *PL/SQL* desarrollado e ir adaptándolo progresivamente sacando provecho de la información extra almacenada.

Desventajas

- No se tienen datos fiables acerca de la perdida de rendimiento que podría suponer respecto al modelo actual.
- Como en la propuesta anterior, se tendrán muchas más tablas a gestionar aunque por el contrario, se podrá hacer de forma transparente. Un ejemplo de ello:

Se desea obtener un listado con todos los datos de los expediente. En el esquema anterior habría que hacer un *join* entre la tabla 'core' y la de configuración, mientras que en este caso tan solo sería necesario hacer una consulta simple en la tabla de configuración.

- Herencia de tablas 'indirecta'

Mismas consideraciones que en el apartado anterior, solo que en este caso no se utilizaría la gestión de objetos que *Oracle* permite, sino simplemente modelos relacionales. Existen tres implementaciones principales:

- Tabla por línea hereditaria

  En una misma tabla se guardan todos los atributos de todas las subclases posibles. Estos podrán ser *NULL* o no según la clase y sus requisitos. Un campo identificará de qué tipo es exactamente.

  Se utilizaría código *PL/SQL* para realizar la discriminación de los campos según el tipo.

- Tabla por subclase

  Se tendrá una tabla por cada subclase posible.

  Esta implementación supone muchas dificultades de uso debido a la cantidad de *joins* necesarios en las consultas. Perderíamos la compatibilidad con el código actual ya que habría que cambiar prácticamente la totalidad del código *SQL* existente.

- Tabla por clase concreta

  Se tendrá una tabla por cada clase concreta (no abstracta). Cada tabla será completa, es decir, se especificarán todos los campos.

  Esta implementación tiene el problema de la cantidad de modificaciones que se deben aplicar a las tablas cuando se quiere añadir un campo generar a un conjunto de descendientes.

Las tres posibilidades comportan mayores problemas de mantenimiento y eso que la anterior propuesta de herencia 'directa'.

En nuestro caso no nos sirve ninguna de las tres estructuras que se proponen, ya que el sistema acabaría en un estado similar al actual donde hay una versión de cada tabla por cliente.

En general no interesan las posibilidades que la herencia permite en cuanto al número de descendientes sino la facilidad de generalización y especificación.

- Conclusiones

La forma más óptima de conseguir el objetivo sería utilizar el modelo objeto-relacional haciendo uso de la herencia por ser el que mejor relación entre ventajas e inconvenientes tiene.

Además de las ventajas comentadas anteriormente, que nos ayudarían a conseguir el objetivo propuesto, esta implementación permitiría empezar a aprovechar todas las funcionalidades objeto-relacional que *Oracle* proporciona. Aprovechando más la herencia y el polimorfismo, podría parametrizarse el sistema de procedimientos incorporando las funciones como métodos de los objetos.

A pesar de poder incorporarse gradualmente, el propuesto es un cambio muy importante respecto al modelo actual. La problemática radica en que difícilmente compensará modificar gran parte de la aplicación, aunque fuese de forma gradual, para conseguir los objetivos propuestos. Sin embargo, para nuevos proyectos seria muy interesante y una propuesta a considerar.

3.5.2.2 Mejora del sistema de listados

- Consultas

Un listado se generará con una consulta de complejidad variable. La información que contenga puede variar según las necesidades del cliente, por lo que una posibilidad es almacenar la consulta como datos dentro de una tabla y facilitar una herramienta externa para la personalización de la misma. De este modo no se necesita manipular código fuente en cada ocasión, por lo que un trabajador con algunos conocimientos sobre *SQL* y la estructura de datos, podría realizar los cambios mediante un asistente.

Para la mejora se propone un esquema con dos tablas de datos, en la principal se guardará el listado estándar que formará parte del *core* del *SEDAS*, mientras

que en la otra se guardará el listado modificado que necesita el cliente, siendo este una modificación de uno de los registros estándar u otro totalmente nuevo.

De esta forma se podrían realizar cambios de versión en esta parte *core* sin modificar la funcionalidad que obtiene el cliente, para ello simplemente sería necesario establecer como listados personalizados los que ya tuviese el cliente y moviendo los estándar que se utilizasen al esquema de configuración.

Habría que aclarar que el esquema indicado a continuación utiliza algunas propiedades del modelo objeto-relacional de *Oracle* en cuanto a la definición de tipos y métodos. Si fuese necesario podría realizarse un modelo relacional equivalente.

Tipos

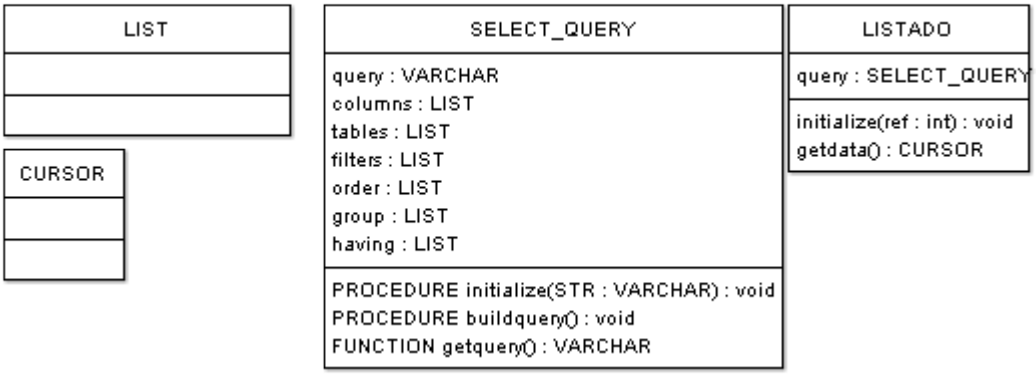


Figura 5: Clases propuestas para parametrización de consultas

Detalles

- **SELECT_QUERY.initialize(STR: VARCHAR) :** Interpreta (*parsing*) del parámetro para establecer los valores de las listas.
- **SELECT_QUERY.buildquery()** : Genera la consulta SQL utilizando las listas.
- **SELECT_QUERY.getquery()** : Retorna el SQL generado de la consulta (podría unificarse con la función anterior.

- **LISTADO.initialize(ref: int)** : Realiza la búsqueda del *SQL* del listado correspondiente al parámetro y construye el objeto QUERY.
- **LISTADO.getdata()** : Ejecuta la consulta del listado y devuelve el resultado (Podría retornar un cursos para su uso posterior o directamente los datos.

Tablas

LISTADO_ESTANDAR		
ID_LISTADO	DESCRIPCION	C_CONFIG

LISTADO_PERSONALIZADO		
ID_LISTADO (FK_LISTADO_ESTANDAR)	DESCRIPCION	C_CONFIG

- Formato / Vista

Arquitectura actual

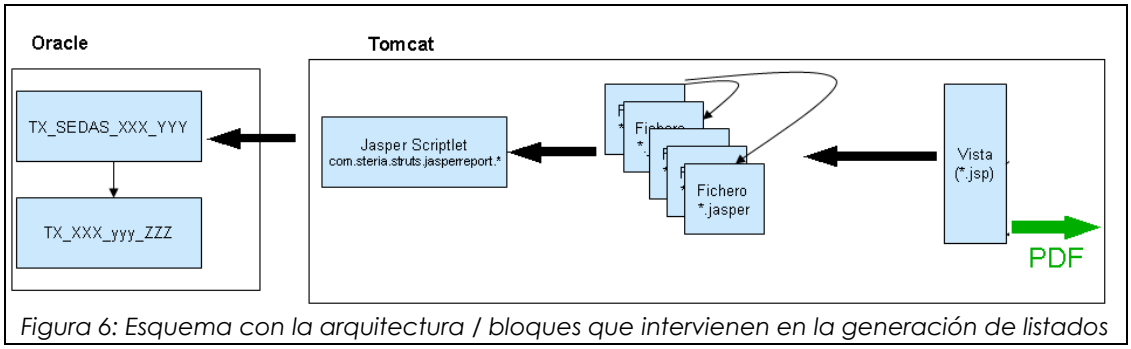


Figura 6: Esquema con la arquitectura / bloques que intervienen en la generación de listados

Se dispone de *n* ficheros *jasper*. Cada uno representa a un listado, que a su vez puede contener *n* sublistados. Todos ellos son ficheros *XML* que siguen el esquema de la librería *JasperReports* y se crearon mediante el diseñador *iReport*.

En la parte de *Oracle*, se dispone de 2 métodos por cada listado o sublistado, que implementan una consulta *SQL* en código *PL/SQL*. Este código también se utiliza en las vistas para visualizar los listados en *HTML* que permiten interactuar con el sistema (p.e. Ofreciendo la posibilidad de fijar un expediente o ver más información de cada resultado).

Estructura de los listados

Para poder pensar en como solucionar el problema, es recomendable saber

como están organizados los listados para saber exactamente que se puede segmentar y dividir. Por ello se han analizado los listados que puede generar la aplicación y llegado a una conclusión al respecto sobre su organización que puede verse en la figura siguiente:

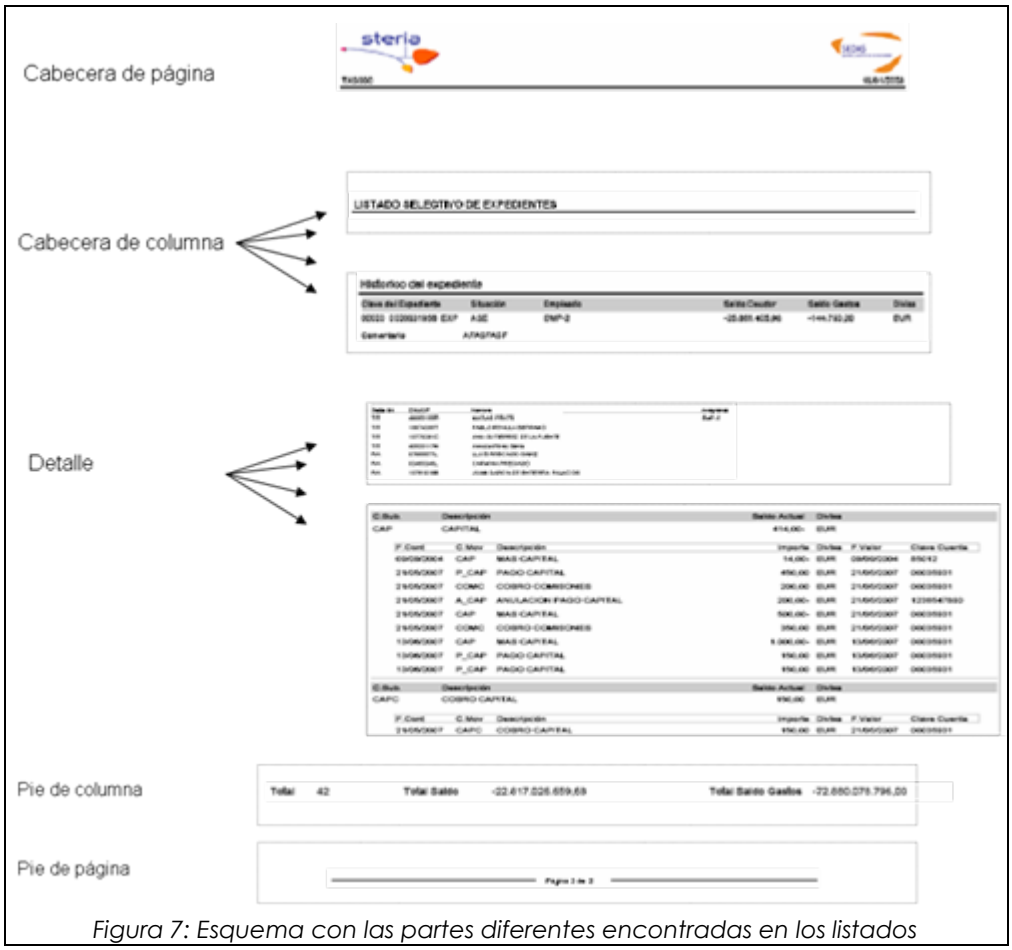


Figura 7: Esquema con las partes diferentes encontradas en los listados

- Problemas detectados

  - Cuando se modifica un listado para un cliente en particular se debe cambiar el código *XML*, esto dificulta el mantenimiento del sistema y la posibilidad de instalar una versión posterior del sistema.
  - Añadir información al informe (p.e. añadir una columna a un listado) puede implicar modificar tanto el *XML* como el código *SQL* y por lo tanto tiene un coste muy superior a cambiar el formato o eliminar una columna.
  - Cada listado se tiene que desarrollar prácticamente desde cero. Cada tipo de listado se diseñó 'manualmente'.

- No se aprovechan todas las posibilidades de las bandas que permite *JasperReports* (p.e. la cabecera, título o el pie de página), están codificadas en cada uno de los informes, por lo que hay ligeros cambios en algunos estilos según el listado.

• Solución propuesta

Vistas las similitudes entre todos los listados y que estas se reducen a variaciones en el número de columnas o filas, se llegó a la conclusión de que se pueden generalizar los XML.

Una de las librerías propuestas para la parametrización sería *Apache Velocity*. Esta biblioteca permite crear plantillas que mediante unas directivas determinadas y unos parámetros. También nos permite modificar un documento con independencia de su formato (fichero de texto, *XML*, binario, etc.). Está librería proporciona funcionalidades similares a las que se pueden conseguir con JSP, pero su sintaxis es ligeramente más sencilla lo que puede facilitar las modificaciones manuales de las plantillas.

Para almacenar la configuración concreta de cada listado y de todos sus elementos se utilizarían una serie de tablas en el *SGBD*. La separación entre el contenido de los listados por defecto y la configuración específica del cliente se conseguiría mediante la creación de un conjunto de tablas en las que se guardaría su apariencia.

El almacenamiento de la configuración de cada elemento se podría solucionar guardando cada atributo de cada elemento como una columna en una tabla de la base de datos.

• Esquema

El resultado final está formado por cinco partes principales:

1. Plantillas JSP o *Velocity*
2. Estructura de tablas en *SGBD* con la configuración de cada listado
3. Paquete de clases Java que permitan el acceso a los datos mediante *VTL* (lenguaje *script* de *Velocity*) o *JSTL*
4. Compilador de *JasperReports* existente
5. Procedimientos y funciones *PL/SQL* existentes

Además se crearía una interfaz en forma de aplicación de escritorio o web para

poder realizar las modificaciones que los clientes soliciten para sus listados. Esta interfaz permitirá modificar de forma más rápida y sencilla que anteriormente, obteniendo una funcionalidad similar a la de otros generadores de informes del mercado.

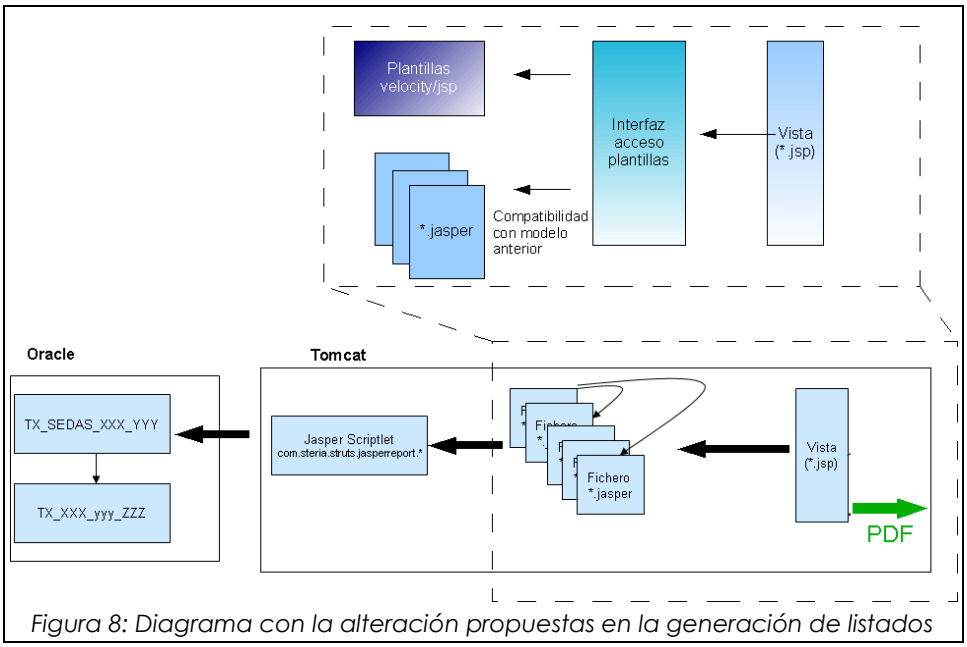


Figura 8: Diagrama con la alteración propuestas en la generación de listados

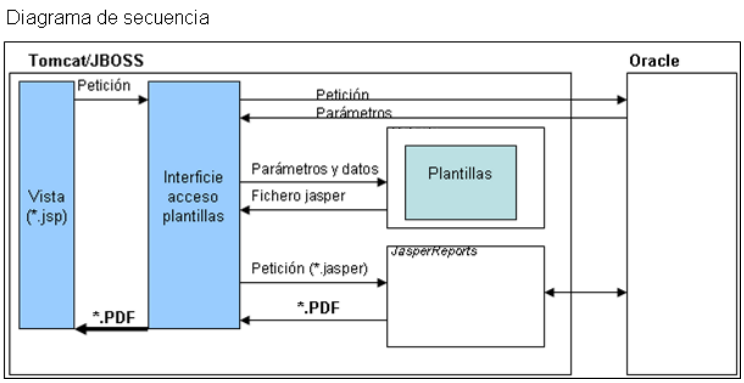


Figura 9: Diagrama de secuencia sobre el proceso propuesto de generación de listados

- Balance

Ventajas

- Unificación del formato. Todos los listados serán idénticos en las partes comunes. Si se desea, por ejemplo, modificar el formato del pie de página, solo es necesario realizar el cambio en un archivo. Este es el contrapunto respecto al modelo anterior en el cual se deberían cambiar prácticamente la totalidad.
- Se dispone de un número reducido de ficheros de plantillas lo cual supone una mejora respecto al modelo anterior en el que se almacena un centenar de ficheros para definir los listados. En este caso se ha generalizado, especificando tan solo los apartados diferenciadores, y almacenando la diferencia en el *SGBD*.
- Se establecen las bases para disponer de un diseñador de informes muy simplificado respecto al utilizado actualmente (*iReport*), que podría tanto utilizarse de forma interna para reducir el tiempo de desarrollo, como externa ofreciéndolo como extra al cliente.
- Se promueve el *refractoring* de los listados por lo que se pueden añadir nuevas funcionalidades y corregir errores.

Inconvenientes

- No se podría utilizar directamente y de forma automática *iReport* para modificar el aspecto de los listados. Esto puede suponer algunos problemas para el personal encargado de su preparación que ya estaba habituado al otro sistema.
  Por ello es obligatorio proporcionar una interfaz para modificar la presentación de cada elemento del listado (alineación, tamaño, etc.) general del *SEDAS* o personalizado de cada cliente de forma sencilla. También sería necesario que la implementación contemplase el uso de los ficheros *Jasper* que ahora se disponen, con el objetivo de facilitar la transición, y de esta forma permitir la convivencia de los dos sistemas.
  - Se añade alguna carga extra al *SGBD* al tener que gestionar el conjunto de tablas. A pesar de ello, no se considera significativo respecto al resto de módulos del sistema, su capacidad y la frecuencia de uso.

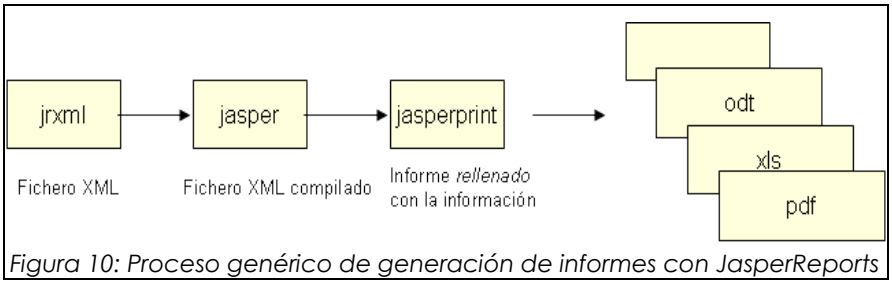


Figura 10: Proceso genérico de generación de informes con JasperReports

- Se aumentaría ligeramente el tiempo necesario para generar el informe, ya que será necesario compilar cada uno para cada solicitud (primer paso de la figura). Actualmente este proceso lo realizan los

propios trabajadores desde *iReport*, por lo que el usuario final utiliza el fichero precompilado eliminando una de las etapas.

Una solución para ello, es una vez se dispusiese de todos los listado se activase un modo *release*, en el cual se generase el fichero *jasper* según el contenido de los registros de las tablas relacionadas. De esta forma, en producción se utilizarían los *jasper* guardados en el *SGBD* y solo se cambiarían las tablas cuando se necesitase algún cambio.

3.5.2.3 Parametrización de los procesos de carga de datos (*batch*)

- Introducción

Los procesos *batch* en este contexto se utilizan para transferir información desde las bases de datos de la entidad bancaria cliente hacia *SEDAS*. Este proceso es necesario ya que parte de la información que gestiona *SEDAS* está guardada en servidores independientes dedicados a otras tareas, y que por razones de rendimiento, seguridad, o compatibilidad, no pueden verse alterados.

La información se exporta periódicamente en un fichero secuencial que se interpreta para extraer su información y actualizar los datos de *SEDAS*. Todo este proceso se gestiona desde un *package* de código *PL/SQL* guardado en el *SGBD*.

La organización de la información varía según el cliente por lo que se pretende añadir una capa de abstracción/adaptación para poder cargar los datos aprovechando el código de entrada a *SEDAS*.

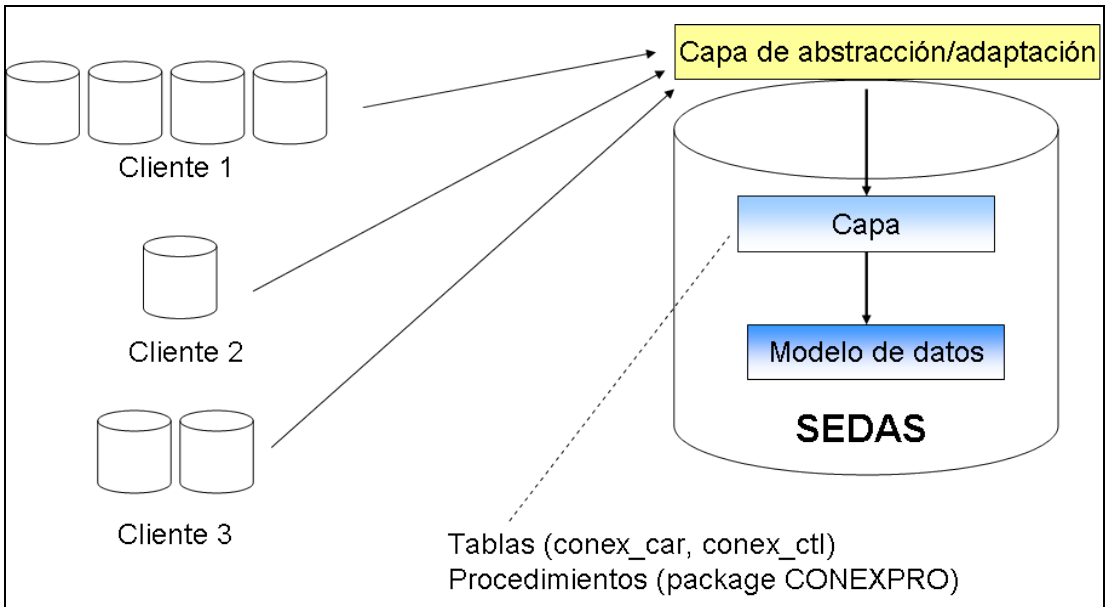


Figura 11: Esquema propuestos para la parametrización de los procesos de conexión

- Implementación

La capa de abstracción constará de diversas tablas de datos y un conjunto reducido de funciones que se encargarán de mover la información a la capa inferior.

El comportamiento global equivaldría al de una translación desde un formato a otro.

La tabla de datos se podría separar según el tipo de entradas a gestionar. En la implementación actual solo hay tres tipos de entradas: cuentas, bienes y personas.

- **Datos**

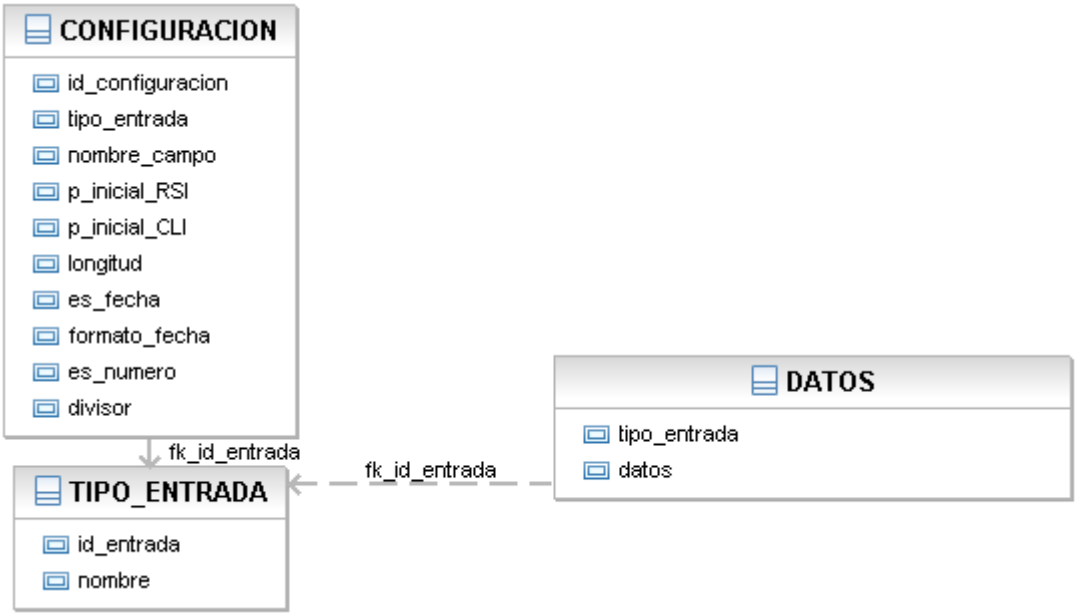


Figura 12: Modelo de datos necesario para implementar la parametrización de los procesos de conexión

- **DATOS:** Guardará los datos '*raw*' en una única columna en el formato propio que maneje el cliente.
- **TIPO_ENTRADA:** Guardará los diferentes tipos de entidades a cargar. Por ejemplo: cuentas, personas, bienes, etc.
- **CONFIGURACIÓN:** Almacenará la configuración concreta de cada propiedad de cada tipo de entrada.

  Ejemplo:

id_configuracion	tipo_entrada	nombre_campo	p_inicial_RSI	p_inicial_CLI	...
1	CUENTA	NOMBRE_TITULAR	5	25	
2	PERSONA	NOMBRE	40	10	
...					
27	BIEN	CLASE	1	44	
...					

- **Pseucódigo de las funciones**

```
func mueve_datos()
  para_cada_fila // tabla DATOS
    si fila.tipo_entrada == 1
      mueve_cuenta(fila.datos);
    fsi
    si fila.tipo_entrada == 2
      mueve_persona(fila.datos);
    fsi
    si fila.tipo_entrada == 3
      mueve_bien(fila.datos);
```

```
    fsi
  fpara
ffunc

funct mueve_cuenta(datos)
    nueva_fila = varchar(500)
    para_cada_fila // tabla configuracion, tipo_entrada = cuenta
    si(!datos.esFecha)
      nueva_fila[datos.p_inicial_RSL, datos.p_inicial_RSL + datos.long] =
                  datos[datos.p_inicial_CLI, datos.p_inicial_CLI + datos.long]
    fsi
    si(datos.esFecha)
      nueva_fila[datos.p_inicial_RSL, datos.p_inicial_RSL + datos.long] =
                  cambia_fecha(datos[datos.p_inicial_CLI,
                                     datos.p_inicial_CLI + datos.long],
                               datos.formato_fecha);
    fsi
    si(datos.esNumero)
      nueva_fila[datos.p_inicial_RSL, datos.p_inicial_RSL + datos.long] =
                  cambia_numero(datos[datos.p_inicial_CLI,
                                     datos.p_inicial_CLI + datos.long],
                               datos.divisor);
    fsi
    guardar_fila(conex_car, datos)
  fpara
ffunc
```

- Conclusión

La implementación no supone ninguna dificultad significativa y la propuesta resuelve el problema de tener que modificar el código fuente para adaptarse a los sistemas del cliente. En vez de cambiar el código se rellenará o modificará la tabla de parámetros según la información que especifique el cliente sobre el formato en que exporta la información.

- Cuestiones pendientes

  - Longitud de los campos

Existe la posibilidad de que la longitud de los campos variase en la información aportada por el cliente y el modelo de datos del sistema, para lo cual se tendrían que añadir alguna cosas más para controlarlo, p.e.

	Nombre titular de cuenta	
	FORMATO CLIENTE	FORMATO RSL
Longitud	90	255

En este caso no habría problemas si la longitud hace referencia a la del formato del cliente, ya que el resto quedaría con el valor por defecto, en caso contrario sería incorrecto, porque se copiaría información de otros campos (mas allá de la

posición inicial + 90)

Si el campo es más corto en el estándar RSL, habría que segmentar según un criterio predefinido.

Diferencia de formatos
Se han considerado estas diferencias en cuanto a la fecha y unidad (*formato_fecha* y *divisor*) pero no otras como por ejemplo el punto decimal (vital por las diferencias con el sistema anglosajón), el formato de expresar los nombres ('apellido apellido, nombre', 'nombre apellido apellido'), etc.

- Gestión de errores

Se ha de desarrollar un sistema de gestión de errores acorde.

3.5.2.4 Parametrización de la interfaz de usuario

- Introducción

Formularios dinámicos según lo que quiera el cliente (implica cambio modelo de datos y en la vista)

Objetivo: Añadir información extra demandada por un nuevo cliente a las pantallas

Implica: Cambiar datos que se muestran en formularios y en tablas

Tres cosas a cambiar:

Consulta y modelo de datos (trabajado en mejoras anteriormente)

Beans (definidos en *XML*, ~200 ficheros)

Se definen que campos se podrían mostrar

JSP's (~600 ficheros)

Se definen los campos que finalmente se muestran y su configuración (aspecto)

- Análisis

El sistema está desarrollado con un marco de trabajo *Struts* 1.1

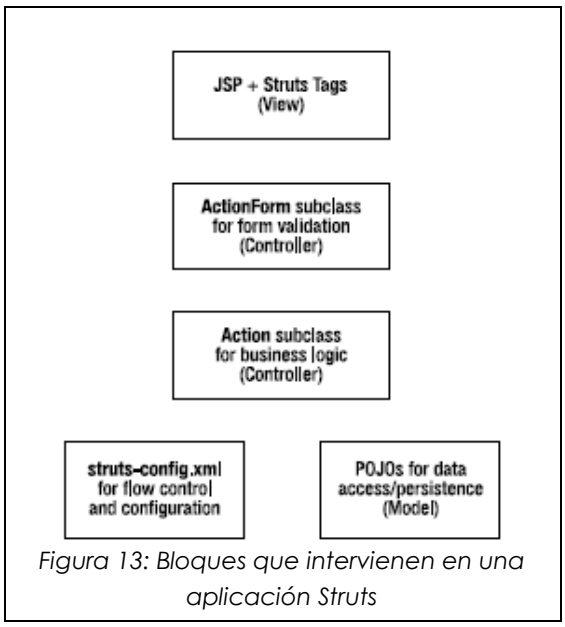


Figura 13: Bloques que intervienen en una aplicación Struts

Para lograr el objetivo es necesario hacer cambios en los *JSP* y los *XML*.

Observada la estructura de **EXP_datosGrales_sit_pro_solo_lec.jsp**

Este fichero genera la vista correspondiente a la información general de expediente, contiene su información de 'cabecera' y una lista con las situaciones por las que ha pasado el expediente:

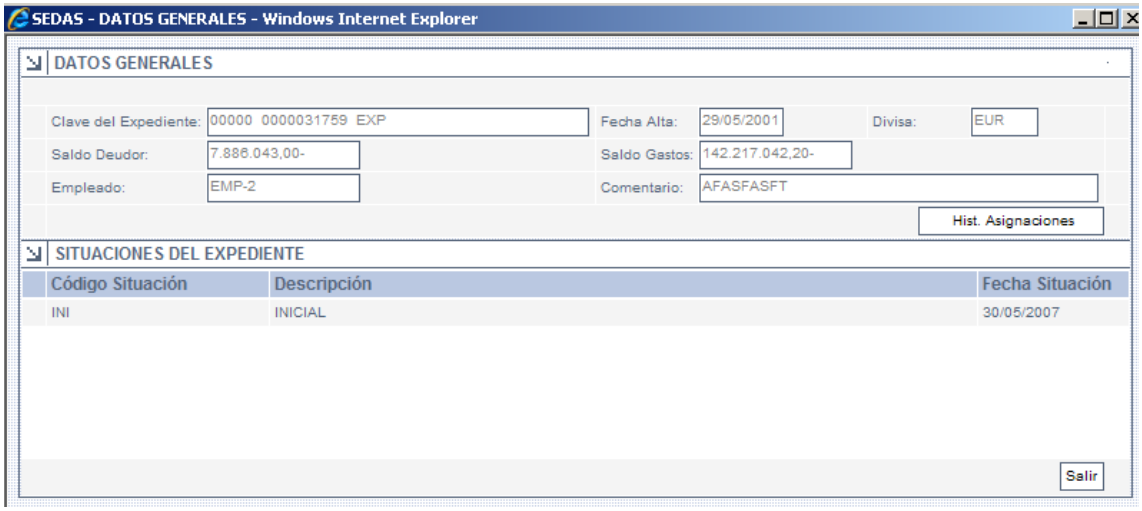


Figura 14: Ejemplo de ventana típica de SEDAS para parametrizar

El objetivo sería poder parametrizar esta parte de la interfaz. Para ello es imprescindible poder poner un número determinado de columnas en la tabla y un formulario con tantos campos como fuese necesario.

Se han pensado dos posibilidades dando por hecho que la configuración del modelo de datos está resuelta y las consultas retornarán todos los datos necesarios:

1. La información mostrada es dinámica, es decir, en tiempo de ejecución, y según la información que se lea del SGBD se crearán las pantallas.

   Pseudocódigo para generar una fila del formulario:

```
Mientras haya columnas en la consulta
{
  // etiqueta
  print("<jsp:printLabel name=elemento[1].nombre estilo=elemento[1].estilo>");
  // input
  print("<jsp:printInput name=elemento[2].nombre estilo=elemento[2].estilo>");
}
```

   Generar una fila de una tabla seria muy similar.

   Principales problemas:

   - Validaciones
   - Estilo de los elementos en los formularios (tamaño, posición, etc.)

2. Se tendrá un diseño patrón de cada pantalla en el que se especificaran todos los campos posibles. A partir de él, se creará la versión específica ocultando los no necesitados.
   Para ello se tendrá un conjunto de tablas en las que guardar qué se quiere ver y qué no, y su configuración especifica (p.e. nombre de las etiquetas).

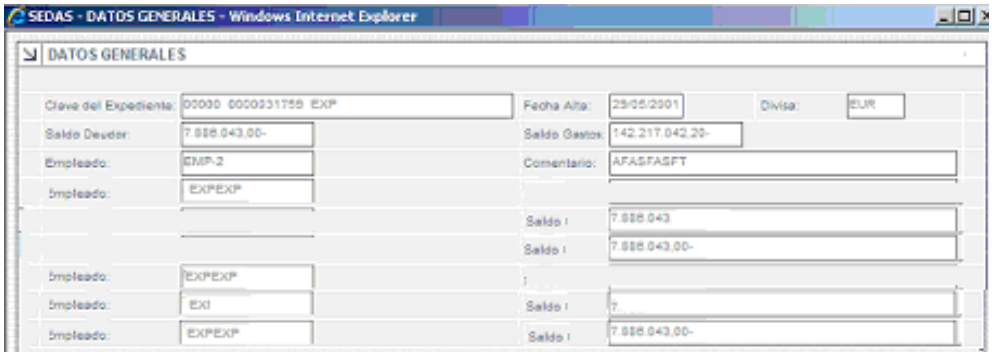


Figura 15: Esbozo sobre una posible organización de los formularios

Problemas:

- Que hacer cuando se necesiten más campos que los establecidos en el modelo patrón.
- Nombre, configuración y etiquetas en los formularios.
- Que hacer con los espacios en blanco que dejan los elementos no seleccionados.

3. Combinación de las dos anteriores para parametrizar específicamente los formularios. Se tendrá un modelo o patrón con una disposición estándar (p.e. filas con 2 parejas etiqueta-*input box*) similar a la posibilidad 2 pero sin diferenciar en el aspecto de los *input-box*, todos tendrán un tamaño y formato idéntico.
4. Teniendo este modelo, la generación del código *HTML* en los *JSP* sería similar al de la posibilidad 1:
5.

```
Mientras haya columnas en la consulta
{
      // etiqueta
     print("<jsp:printLabel name=registro[1].nombre style="predefinedLabel" >");
      // input
     print("<jsp:printInput name=registro[2].nombre style="predefinedInput">");
}
```

Ventajas:

- Escalabilidad muy buena, el formulario puede constar de tantos campos como sea necesario.
- El código se simplifica notablemente.

Problemas:

- Mismo estilo para todos los elementos. Problema por ejemplo en los campos obligatorios de los formularios de entrada de datos, que actualmente se remarcan con la fuente en color rojo.
- ¿Como manejar los validaciones en los formularios de entrada/modificación?

6. Modificación del esquema anterior creando un *core* donde se tendrán los campos que coincidirán en todos los clientes. Estos campos se implementarán con el código existente actualmente.
   Los campos declarados como opcionales se implementarán como en la propuesta anterior, se tendrá un *bean* que guardará el resultado de un *procedure* y a partir de él se crearán tantas filas en las tablas como fuese necesario.

   Esta implementación implicaría contar con dos procedimientos para cada tabla, uno que retornará los datos *core* y otro los datos

opcionales.

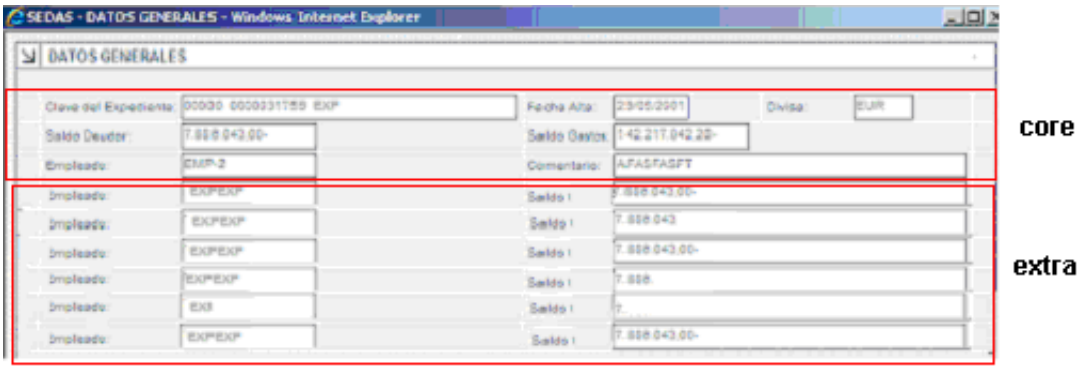


Figura 16: Esbozo con parametrización de formularios

Ventajas:

- Escalabilidad buena, el formulario puede constar de tantos campos como sea necesario.

Problemas:

- Mismo estilo para todos los elementos opcionales. Por ejemplo, habría problema en los campos obligatorios de los formularios de entrada de datos, que actualmente se remarcan con la fuente en color rojo, en esta implementación los campos *no-core* no podrían mostrarse directamente como obligatorios.

- Habría que crear y mantener un procedimiento extra de consulta y cabecera por cada entidad

- **Conclusiones**

Ninguna de las cinco posibilidades es perfecta, las tres suponen romper de forma total o parcial con el modelo que propone *Struts* de modelo-vista-controlador, dejando toda o parte de la vista a especificar en tiempo de ejecución.

De todas la que menos problemas supone es la última.

3.5.2.5 Cambios en el interfaz web

- Introducción

SEDAS dispone de una interfaz web bastante cuidada. A pesar de ello, se pueden realizar algunas mejoras para aumentar su usabilidad.

Los cambios propuestos se pueden clasificar en dos tipos, mejoras del interfaz, en las que se incluirán las que sean un cambio en la representación de elementos ya existentes, y nuevas funcionalidades, las que añadan una nueva funcionalidad que no esté presente en la versión actual.

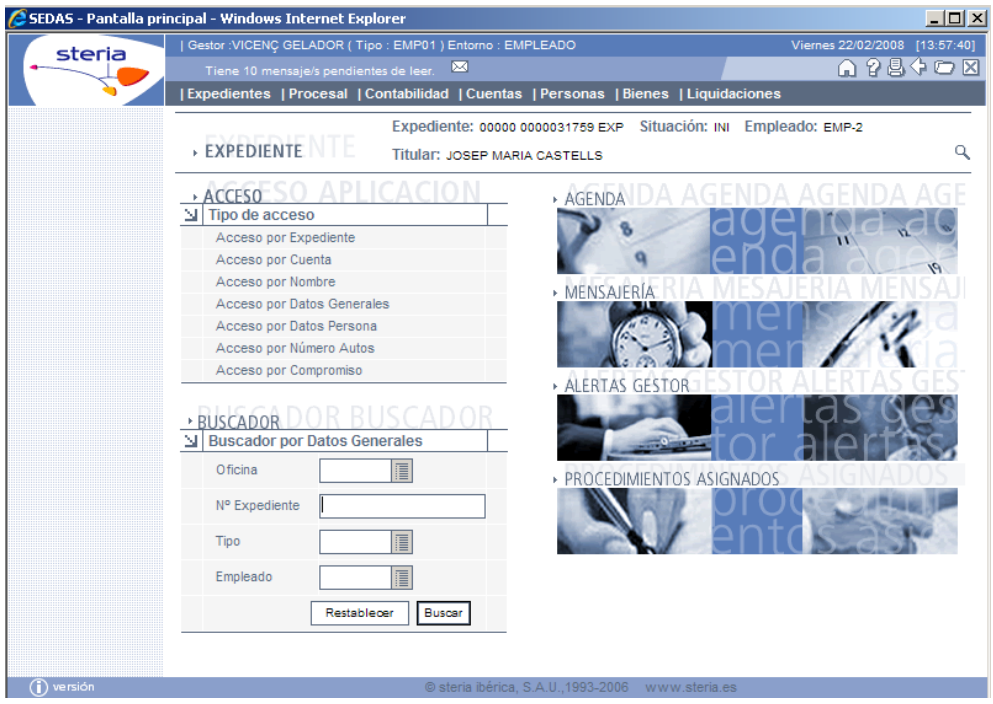


Figura 17: Página inicial o home de SEDAS

- Propuestas

  - **Cuadro / Marco con datos del expediente fijado**

Actualmente esta información se muestra tan solo en el 'home' del empleado y en las páginas que muestran información ligada al expediente fijado. Otra posible opción sería disponer de esta información en todo momento para tener una referencia cuando se realicen acciones de búsqueda.

Como en las siguientes propuestas, la idea sería dar la posibilidad al usuario de poder ocultar esta información si no la considera necesaria o si simplemente requiere más espacio útil para visualizar datos. Para ello se podría añadir un menú de configuración personal, similar al de administración de la aplicación, o una serie de opciones en el menú principal.

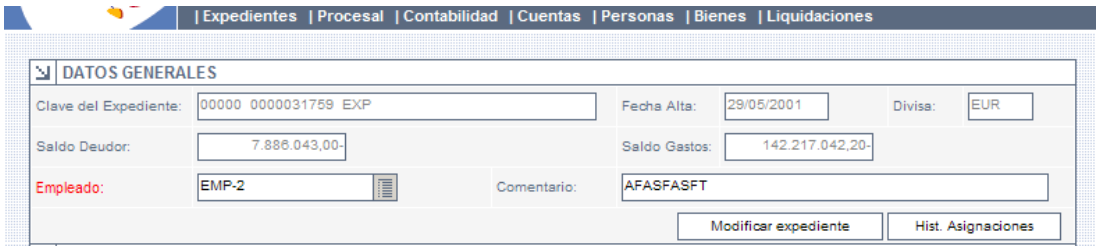


Figura 18: Datos de el expediente en ventana de datos generales

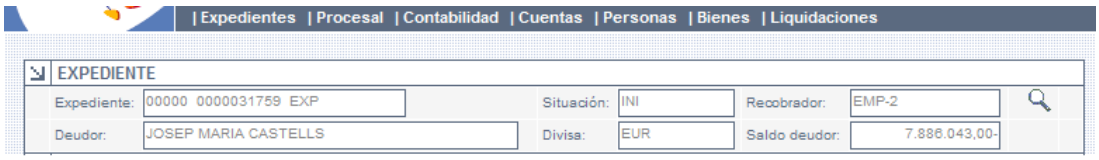


Figura 19: Datos de expediente en las ventanas referidas a uno concreto

La idea seria tomar uno de los dos modelos como patrón y a partir de ahí crear uno genérico visible en todo momento. Se muestra una maqueta en la figura 4.

- **Cuadro / Marco adicional con últimos expedientes fijados**

El modelo de aplicación permite muchas facilidades para la búsqueda de expedientes. Sin embargo, según el número de casos a los que tenga que enfrentarse un usuario, darle la posibilidad de cambiar rápidamente entre los *n* últimos expedientes podría suponer un ahorro de tiempo.

Una funcionalidad similar se puede conseguir con el listado de procedimientos asignados al trabajador, pero en este caso el campo seria visible desde cualquier pantalla, se podría hacer con tan solo una acción y sin verse limitado a un conjunto tan reducido de expedientes.

Al igual que en la propuesta anterior, se debería dar la posibilidad al usuario de desactivarlo si lo considera pertinente.

Figura 20: Maqueta con cuadros permanentes de expediente fijado y últimos abiertos

Cabe señalar que se ha aprovechado más el espacio eliminando la separación entre elementos.

El cuadro de los últimos expedientes fijados podría especificar el nombre del titular si fuese más descriptivo para el usuario en vez de la referencia del expediente. La funcionalidad también se podría conseguir con un menú desplegable si es necesario más espacio útil.

- **Mejora del apartado de notificación de mensajes**

El espacio dedicado a este menester tan solo informa sobre los mensajes pendientes. Se podría aprovechar mucho mejor este espacio añadiendo más información, como por ejemplo, el número de alertas o el total de procedimientos asignados. Además del número total podría especificarse la fecha del último acontecimiento de cada tipo (último mensaje, alerta o asignación)

Figura 21: Añadida indicación sobre el número de mensajes, alertas y procedimientos asignados del usuario

- **Anotaciones**

Modificar o añadir un sistema de anotaciones paralelo en el cual poder gestionar otras propias del usuario, obteniendo una funcionalidad similar al de una agenda

personal. Estas anotaciones podrían ser introducidas de forma manual y revisadas en formato de listado o calendario.

3.5.2.6 Actualización de la librería de reporting

- Introducción

La versión de la librería de *reporting* utilizada, *JasperReports*, no cuenta con algunas mejoras que se han ido añadiendo durante los últimos años.

Entre ellas, y no siendo propiamente una funcionalidad de la biblioteca, es la posibilidad de utilizar las últimas versiones de *iReport*. Esto es debido a que desde que apareció la versión que se utiliza actualmente en el proyecto SEDAS, la cual se podría prácticamente catalogar como *alpha*, han aparecido dos líneas de desarrollo basadas en *eclipse RCP* y en *Netbeans Platform*. Ambas versiones son muy superiores tanto en las funcionalidades que ofrecen, como en la estabilidad de la herramienta, en especial la versión basada en eclipse, ya que se han desarrollado versiones estables desde finales de 2006.

- Análisis

En un principio, no se espera tener demasiadas dificultades para implementarlo, los informes son relativamente simples y la librería se ha desarrollado con la retrocompatibilidad en mente. Se espera que haya ciertos elementos utilizados que ahora estén considerados obsoletos o depreciados, por lo que si se encuentra una cifra manejable se intentarán adaptar. Si se encontrase una cifra excesiva, considerando las limitaciones del proyecto, se solucionarían solo las cuestiones imprescindibles, ya que teniendo en cuenta alguna de las otras mejoras propuestas de este apartado del proyecto, existe la posibilidad de que se vuelvan a diseñar desde cero todos los informes.

3.5.2.7 Exportación de listados

- Introducción

Actualmente el sistema genera ficheros PDF como medio genérico para obtener un fichero final, que no se ha de modificar ni distribuir, ya que tan solo se ha creado para su impresión directa en el mismo momento en que se genera.

Recientemente algunos clientes se han interesado en poder distribuir listados para facilitar la comunicación entre empleados, y en tenerlos disponibles en formatos fácilmente editables con programas como hojas de cálculo. De este modo se

puede añadir información de forma 'limpia', realizar modificaciones puntuales o fusionar listados para generar informes más complejos sin que por ello se necesite modificar la propia aplicación.

- Análisis

La funcionalidad consistiría en añadir alguna opción en forma de menú o listado para que todos y cada uno de los listados de los que ahora se disponen puedan estar disponibles en otros formato y que aprovechando las amplias posibilidades de exportación que *JasperReport* dispone, el sistema escale de forma adecuada y no sea excesivamente complejo añadir otros formatos.

El software más utilizado en los clientes para este tipo de tareas de edición son las hojas de cálculo, y entre estas, *Microsoft Excel* es la más extendida con diferencia. Por ello, se podría considerar tres formatos distintos: *XLS*, *CSV* y *XML*.

El formato *XLS*, además de ser propietario y tener una compatibilidad limitada con el resto de hojas de cálculo del mercado, no añade prácticamente ninguna funcionalidad necesaria en los listados, ya que los clientes tan solo desean disponer de la información en forma tabular con una cabecera especifica.

El formato *XML* podría ser el más adecuado junto con una hoja de estilo apropiada, pero el formato *CSV* permite toda la funcionalidad requerida y tiene una mejor compatibilidad con las últimas versiones de *Excel*.

Por ello, inicialmente se dará soporte para *CSV* dejando las otras alternativas como posibles ampliaciones.

Esta funcionalidad se vería muy beneficiada si se implementasen anteriormente las otras dos mejoras relacionadas con los listados, debido a que los exportadores de la librería trabajan intensivamente intentando generar la información en forma tabular o distribuida en forma de tabla, mientras que el diseño 'manual' realizado a través de *iReport* puede conducir fácilmente a desajustes de las posiciones de los elementos.

3.6 Cambios a implementar

3.6.1 Introducción

Tal y como se explicaba en la introducción, debido a las limitaciones temporales del proyecto será imposible implementar todos los apartados, por lo que se hace necesario realizar una selección de las más idóneas dados los objetivos y las soluciones propuestas.

3.6.2 Resumen de mejoras

3.6.2.1 Cambios en el modelo de datos

Se abordó el problema intentando solucionarlo utilizando conceptos de herencia para solucionar así tanto los cambios en el modelo de datos, como los de la lógica de negocio. La conclusión final fue que la que mejor podría adaptarse a este tipo de aplicación es el desarrollado a partir del modelo objeto-relacional sobre el *SGBD*.

3.6.2.2 Mejora del sistema de listados

En este apartado tratábamos de simplificar el sistema de creación, manipulación y gestión de los listados. Se vio la posibilidad de generalizarlos y simplificar su formato de representación para así reducir el tiempo de desarrollo y mantenimiento.

El resultado obtenido es el diseño de una capa intermedia entre la aplicación y la librería de generación de informes, que también sería un buen punto de partida para disponer finalmente de una aplicación de generación de informes muy simplificada similar a la que disponen productos de la competencia.

3.6.2.3 Parametrización de los procesos de conexión

En esta parte se trató de generalizar el modo en el cual el sistema se conecta con su proveedor de datos para reducir al máximo el tiempo de adaptación de esta parte a cada uno de los clientes. El resultado final fue la especificación de una capa de interconexión que aprovecha el código estándar que actualmente analiza e introduce información en *SEDAS*.

3.6.2.4 Parametrización de la interfaz de usuario

En este apartado se trató de generalizar y parametrizar los formularios y tablas que se muestran en la aplicación. Estos cambios serian los complementos imprescindibles para aprovechar al máximo las mejoras 1 y 2.

Se llegó a la conclusión de que el modelo *MVC* utilizado limita en gran medida los cambios que se pueden hacer, por lo que todas y cada una de las opciones propuestas se ven limitadas en mayor o menor medida.

3.6.2.5 Cambios en el interfaz web

Este capítulo se trabajó desde la perspectiva de un ingeniero de software especializado en tareas de usabilidad. De esta forma se revisó la interfaz gráfica de *SEDAS* pensando en formas de enriquecerla y mejorarla.

Los cambios son de diferente complejidad, desde los simples que apenas requieren modificar una función específica, a otros de mayor complejidad que requieren una mayor dedicación y para lo cuales es necesario añadir funcionalidades a la capa de lógica *PL/SQL* de *Oracle*.

3.6.2.6 Actualización de la librería de reporting

Este propuesta abordaba la posibilidad de actualizar *JasperReport* a alguna de las últimas versiones estables. Se concluyó que el principal beneficio sería la posibilidad de actualizar al mismo tiempo *iReport*, ya que este incorpora una gran cantidad mejoras.

3.6.2.7 Exportación de listados

En este caso se trató de analizar porque y como se podrían añadir opciones de exportación de listados al proyecto. Se concluyo que aprovechando las posibilidades de la librería se podría hacer y de forma no excesivamente compleja, haciendo que los formatos soportados inicialmente pueden ser el *PDF* actualmente utilizado y el *CSV*.

3.6.3 Comparativa

En todos los casos se ha ponderado de 0 a 10, siendo 10 el máximo posible y el más positivo.

Mejora / Críterio	Cambio en el modelo de datos	Mejora del sistema de listados	Parametrización de los procesos de conexión	Cambios en el interfaz web	Parametrización de la interfaz de usuario	Actualización de la librería de reporting	Exportación de listados
Facilidad implementación	3	7	8	9	5	10	10
Calidad de la solución	6	8	9	7	1	10	10
Necesidades de la empresa	8	6	7	5	8	8	9
Independencia respecto al sistema	2	10	10	9	6	10	9
Posibilidades de ampliación	10	7	6	5	10	5	6
Beneficio académico	9	7	6	6	6	5	6
Total	**38**	**45**	**46**	**41**	**36**	**48**	**50**

3.6.4 Conclusiones

La menos valorada es la primera alternativa ya que solo destaca ligeramente en un apartado. En gran medida, esto ha sido porque se ha considerado que su implantación final requeriría muchos más recursos y modificar muchas partes del sistema. Esta solución se podría limitar a partes del sistema más dadas a sufrir modificaciones, reduciendo de este modo los recursos necesarios y las partes afectadas. Además, representa una opción interesante para estudiar de cara al futuro, ya que se trabajan aspectos relativamente modernos y poco explotados.

La segunda peor debe en gran medida la baja valoración a ser una solución 'pobre' o incompleta.

La tercera tiene otro problema, y es la dificultad de disponer de una base de datos y un entorno a partir del cual se pueda trabajar de forma totalmente independiente por lo que su implementación se complica.

De las tres siguientes, podrían descartarse la cuarta o quedarse como una

funcionalidad extra o complemento a implementar, debido a que, al menos inicialmente, requiere muchos menos recursos que el resto.

Las otras soluciones están muy igualadas en todos los apartados y podrían llegar a escogerse la mayoría. Un apartado importante sobre el que destacan las dos últimas es la mayor necesidad de la empresa de disponer de ellas, por lo que este criterio puede ser decisivo para su elección final.

Una vez comentado con la empresa, se ha determinado que una de las mejoras de la interfaz de usuario de las propuestas, la de mostrar y permitir realizar cambios de expedientes es una buena candidata para incorporarla al proyecto si se implementase de forma adecuada, por lo que será una de las primeras en realizarse.

Además quedará pendiente trabajar algo en la tercera si se pudiese tener el entorno necesario para ello disponible.

Por todo ello se ha determinado que las que finalmente se realizarán serán:

- Actualización de la librería de *reporting*
- Exportación de listados
- Menú de accesos a expedientes
- Mejora del sistema de listados

4 Diseño

4.1 Introducción

Debido a que este proyecto no sigue el patrón típico de desarrollo de una única aplicación o sistema, se ha optado incluir en esta sección el diseño de cada una de las partes, y en la siguiente su correspondiente codificación o implementación a pesar de que el desarrollo consistió en diseñar y justo después implementar cada mejora.

4.2 Actualización de librería de reporting

El diseño inicial de este cambio fue muy reducido ya que apenas se podría hacer nada hasta finalmente empezar a implementarlo. El diseño consistió más que nada en analizar las clases que se utilizaban y la pila de llamadas que se producía al acceder a la impresión de los listados.

El paquete de clases que realiza las tareas de interfaz con *JasperReports* está contenido en uno mayor con todas las clases Java creadas para el proyecto *SEDAS*. Entre otros, incluye paquetes para tareas como la ejecución de procedimientos *Oracle*, transacciones, soporte de fuentes de datos personalizadas, librerías de marcas *JSP* (*taglibs*), etc. El nombre de esta librería es **com.steria.struts**.

El primer problema encontrado fue que *JasperReports* cambió la URI del paquete y las clases, que pasó de ser **dori.jasper** a **net.sf.jasperreports**. Para solucionarlo se creó un nuevo paquete: com.steria.struts.jasperreports2 y se adaptó cada una de las clases necesarias. La decisión de crear un nuevo paquete y no modificar el existente, fue debida a que esta librería contiene una gran cantidad de código y que cambiarla implicaría realizar un complejo proceso de pruebas. Además, para implementar alguna de las otras mejoras sería estrictamente necesario añadir más clases y ampliar las existentes, por lo que tener un paquete independiente ayudaría en su desarrollo y pruebas.

Tener un nuevo paquete interfaz hace que se tenga que se aborden dos partes de este sistema:

- En tiempo de diseño: Los ficheros XML tienen dependencias con el interfaz a *JasperReports*, por lo es necesario modificarlos para poder compilarlos y generar los *JASPER*.
- En tiempo de ejecución: Los *JSP* de impresión de listados hacen llamadas a

la clase **STSPrinter**, por lo que se tendrán que modificar todos.

El flujo que se seguía para obtener finalmente el listado en formato imprimible seguía un esquema como el del siguiente ejemplo:

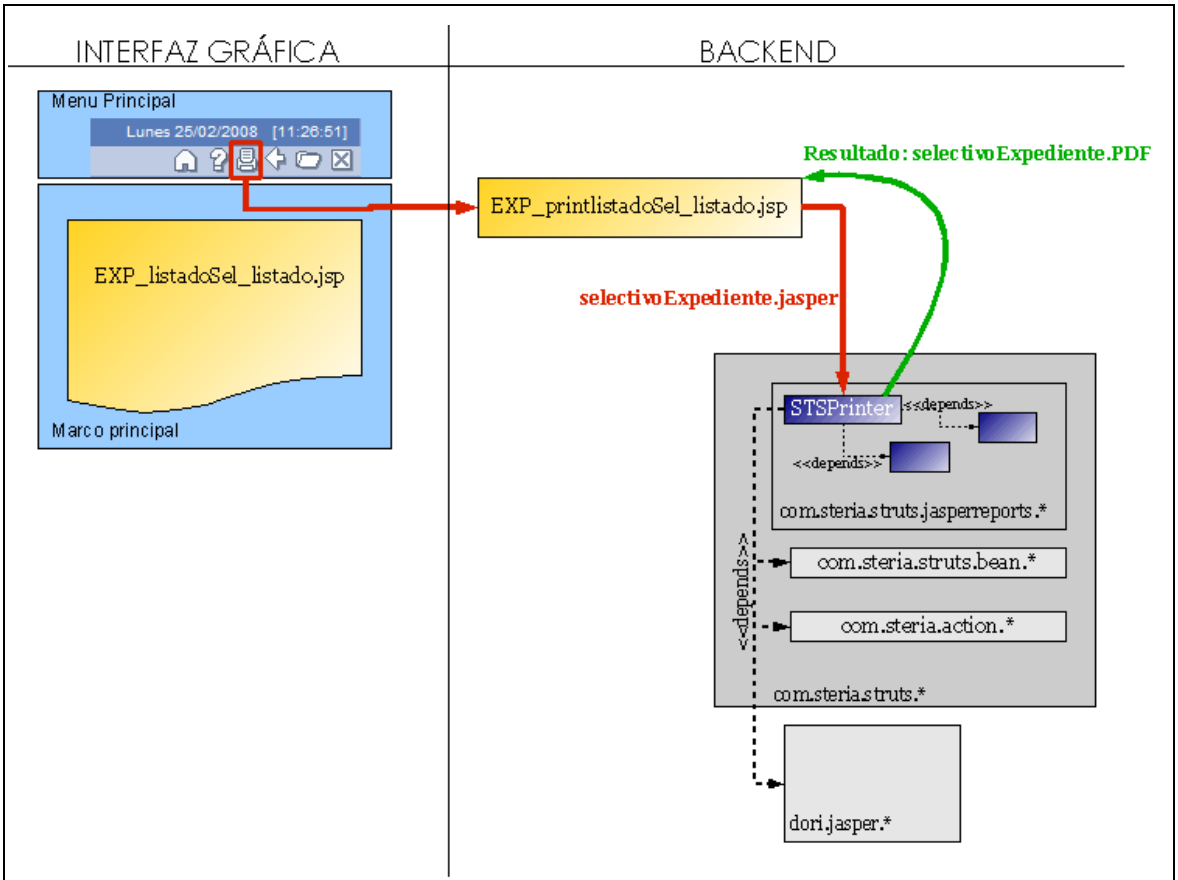


Figura 22: Diagrama de bloques de la implementación del sistema de listados

En el diagrama se puede observar la interfaz gráfica, la cual muestra un menú y un listado. Al activar el botón de impresión, se ejecuta una acción que termina en el fichero *EXP_printlistadoSel_listado*. Este, utilizando el nombre del listado, hace una llamada a la clase *STSPrinter*, la cual busca el fichero en el sistema de archivos y una vez realizadas una serie de llamadas, genera un fichero PDF que finalmente se retorna al usuario.

En este caso la clave está en la clase ***STSPrinter***, ya que tiene diversas dependencias que hay que satisfacer para su correcto funcionamiento.

Una vez estudiado se vio que habían dos cambios que se tendrán que realizar: por un lado cambiar la dependencia hacia *dori.jasper* a la nueva *URI* creando un nuevo paquete sustituto, y por el otro modificar los ficheros *XML* que contienen los informes, ya que estos utilizan las clases del paquete que se va a sustituir.

Por lo tanto, se finalizó este fase creando un esquema como el siguiente:

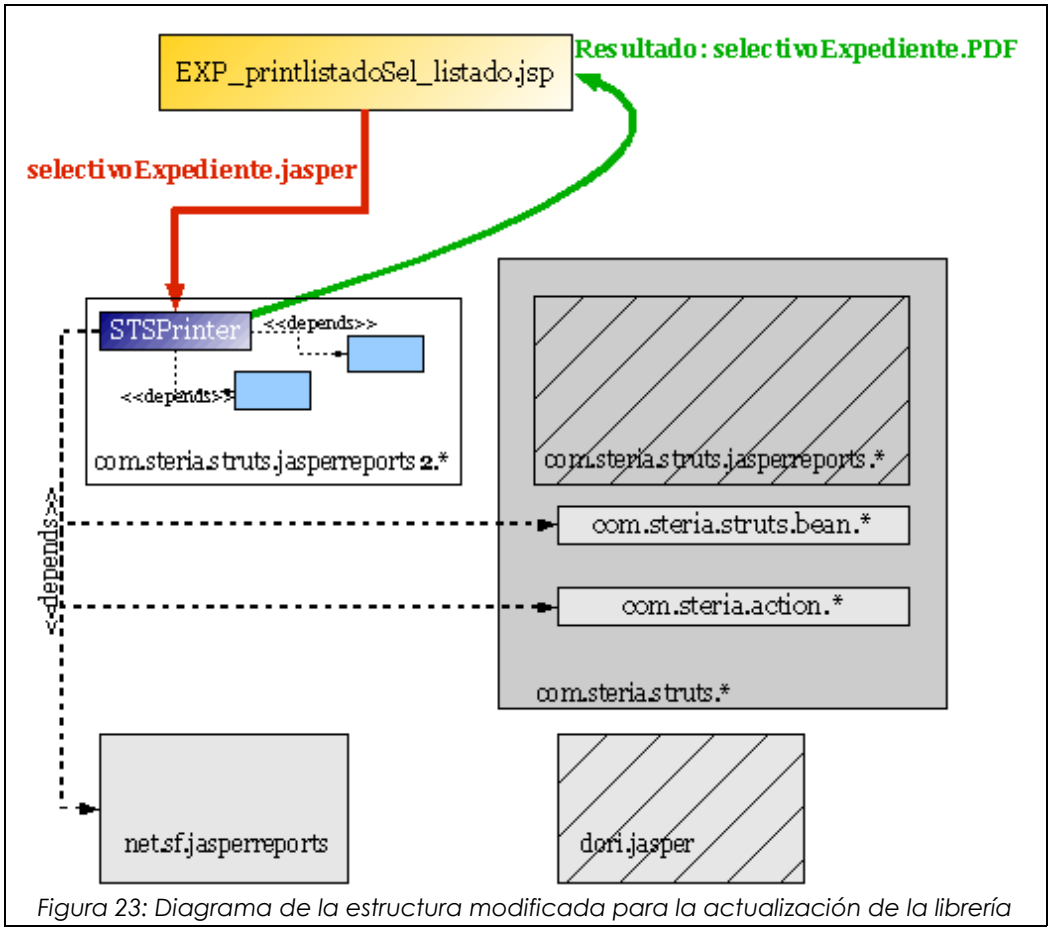


Figura 23: Diagrama de la estructura modificada para la actualización de la librería

4.3 Exportación de listados

Una vez comprendida como es la traza de ejecución de obtención de un listado, y estudiando las clases de la biblioteca que permiten generarlos en cada uno de los formatos, se determinó que había que contemplar la posibilidad de que el número de formatos pudiese ser escalable, a pesar de inicialmente solo se necesitasen dos concretos. Por ello, se decidió ampliar la clase *STSPrinter* añadiendo un método para permitir configurar en cual de los formatos soportados se desea obtener el informe:

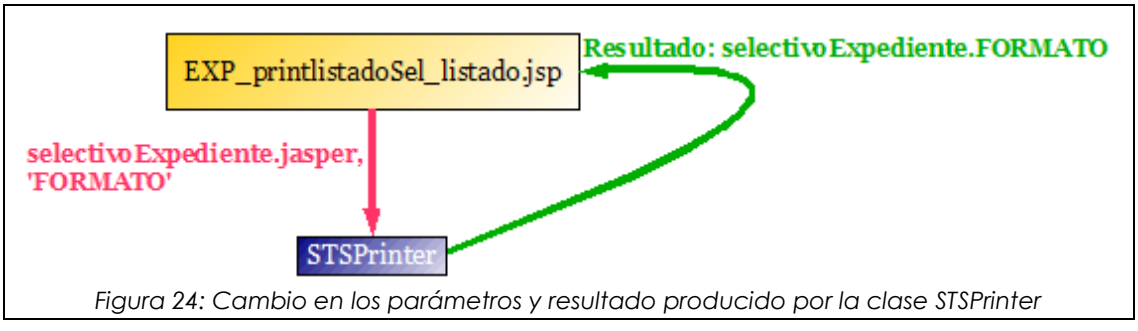


Figura 24: Cambio en los parámetros y resultado producido por la clase STSPrinter

En este caso, la sección del diagrama muestra como aparte del nombre del listado, se le pasa como parámetro el formato en que se desea obtener.

Para recoger el formato exacto en que el usuario quiere el listado, se diseño un menú que mostraría un icono por cada tipo. Con lo cual, la traza completa sería como la siguiente:

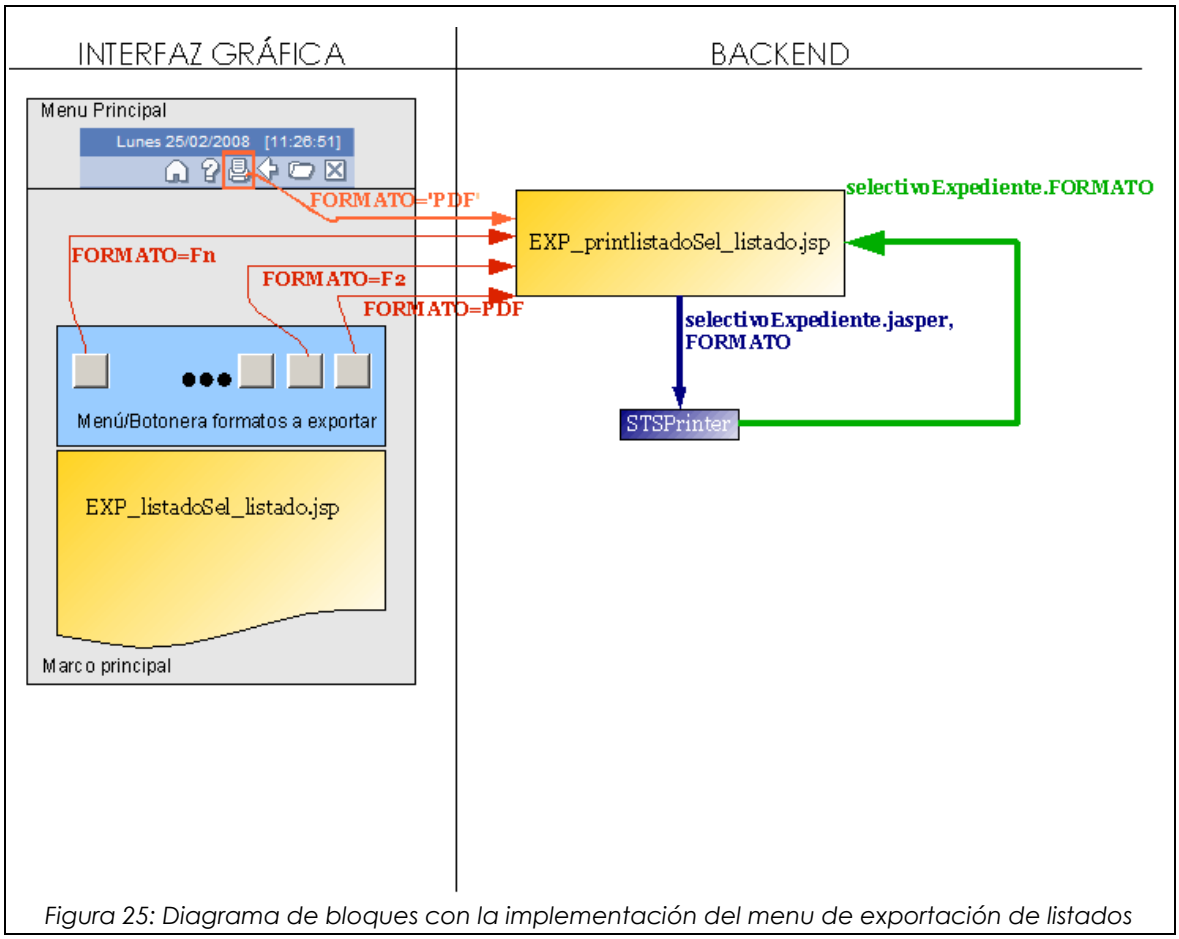


Figura 25: Diagrama de bloques con la implementación del menu de exportación de listados

Los cuadros en azul serán botones que a través de Javascript ejecutarán la acción pasando el formato oportuno como parámetro.

Un consideración es que por razones de compatibilidad, se ha mantenido el botón de impresión del menú superior, para de este modo no alterar la forma en que los usuarios realizan las tareas de impresión hasta el momento.

4.4 Listado de accesos a expedientes

En el capítulo de análisis se hacia una breve propuesta acerca de poner a disposición de los usuarios un pequeño menú o listado disponibles en todas las páginas que permitiese tener un conjunto de expedientes con los que poder trabajar de forma rápida.

Esta propuesta obtuvo el visto bueno rápidamente por la dirección del proyecto, por lo cual se diseñará e implementará un prototipo con ella. Se han incluido también el conjunto de requisitos y casos de uso derivados, ya que se dispuso de esta información después de realizar el análisis previo.

4.4.1 Requisitos

Los requisitos que debe cumplir han de ser:

- Accesibles desde todas las pantallas
- Poder ver una conjunto de los últimos expedientes con los cuales trabajó el usuario (expedientes que fijó)
- Poder fijar cada uno de los expedientes desde cualquier página sin que por ello cambié la sección en la cual el usuario se encuentra, para de esta forma poder realizar comparaciones.
- Gestión del listado para que esté siempre actualizado.
- El listado no ha de guardar solo los accesos que se han realizado durante la sesión en curso, sino que se han de guardar para otras posteriores.

4.4.2 Casos de uso

Se han añadido tres nuevos casos de uso **Cargar listado**, **Cambio rápido expediente** y **Actualizar listado** tal y como se muestra en el siguiente diagrama:

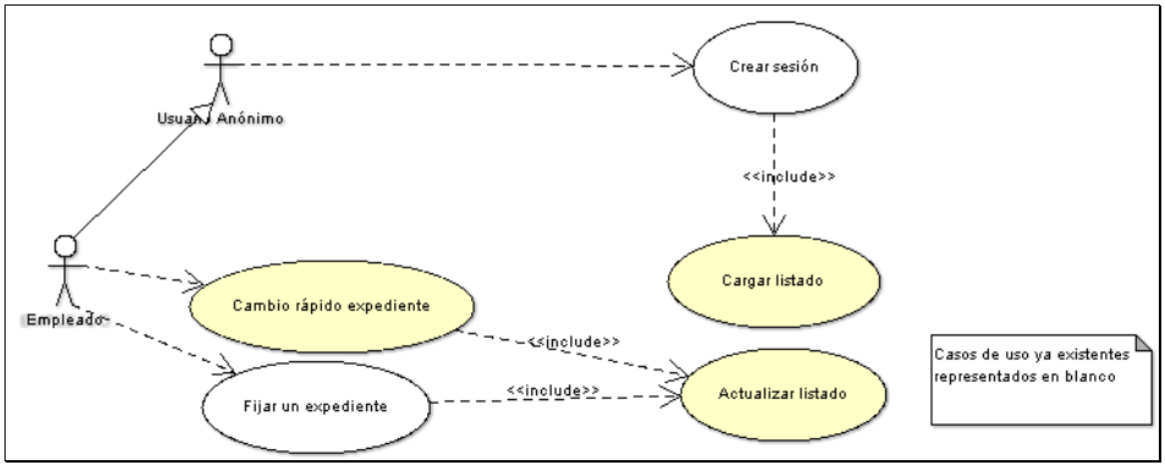


Figura 26: Diagrama de los casos de uso implicados en el menú de accesos

4.4.3 Modelo de datos

Para no correr riesgos y no afectar a otras ramas de desarrollo del proyecto, se añadió una entrada de datos (*datasource*) al proyecto *Struts* y un esquema a la base de datos.

El primer problema encontrado es que no todos los expedientes están vinculados con una persona o titular en la base de datos de desarrollo, por lo que se necesita otro elemento que sí esté presente en todos los expedientes para poder mostrarlo en el menú. Se escogió para ello el identificador del expediente.

Otro problema encontrado es que la aplicación no carga en la sesión el identificador del empleado, sino el identificador del gestor asociado, que puede ser un alias. Por ello se decidió que para modificar la mínima cantidad de código posible, se guardase en la tabla creada el nombre del empleado y no su identificador. Esto implica que para el correcto funcionamiento del prototipo no puede haber más de un empleado con el mismo nombre.

Dentro del esquema añadido se creó únicamente una tabla:

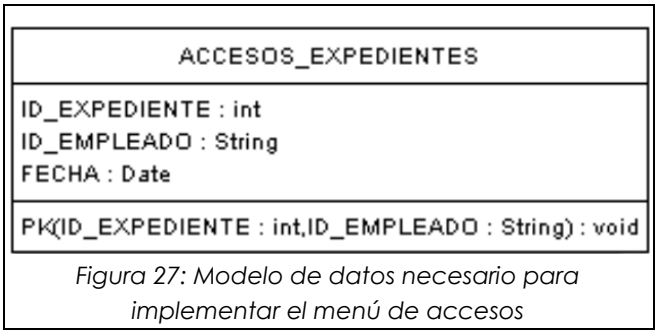


Figura 27: Modelo de datos necesario para implementar el menú de accesos

```
CREATE TABLE ACCESOS_EXPEDIENTES
(
 ID_EXPEDIENTE NUMBER,
 ID_EMPLEADO   VARCHAR2(15 BYTE),
 FECHA         DATE NOT NULL,
 CONSTRAINT PK_ACCESOS_EXPEDIENTES PRIMARY KEY (ID_EXPEDIENTE, ID_EMPLEADO)
)
```

4.5 Gestor de informes

Esta parte corresponde a la mejora que se pretendía realizar sobre el sistema de listados actual. Como se concluyó en el análisis, es necesario tener un gestor o aplicación para poder soportar el esquema propuesto en el cual los informes estuvieran almacenados en el *SGBD*.

En este apartado se creará un diseño que permita la funcionalidad planteada en el análisis. Realizar el trabajo requerido para disponer de una versión totalmente funcional en cuanto a las tareas de gestión y de diseño de informes sobrepasaría los límites temporal que se establecieron en el proyecto. Por lo tanto, del prototipo se implementarán las partes necesarias para verlo en funcionamiento, dándole importancia al proceso de gestión de informes y menos al de diseño, ya de esta forma se podría dedicar este tiempo a conseguir integrar el gestor con *SEDAS*.

Una de las partes que se podrían evitarse en el prototipo es el soporte a la de creación de nuevos tipos de plantillas, ya que según como se deseen, se deberían hacer cambios funcionales en diseñador.

4.5.1 Casos de uso

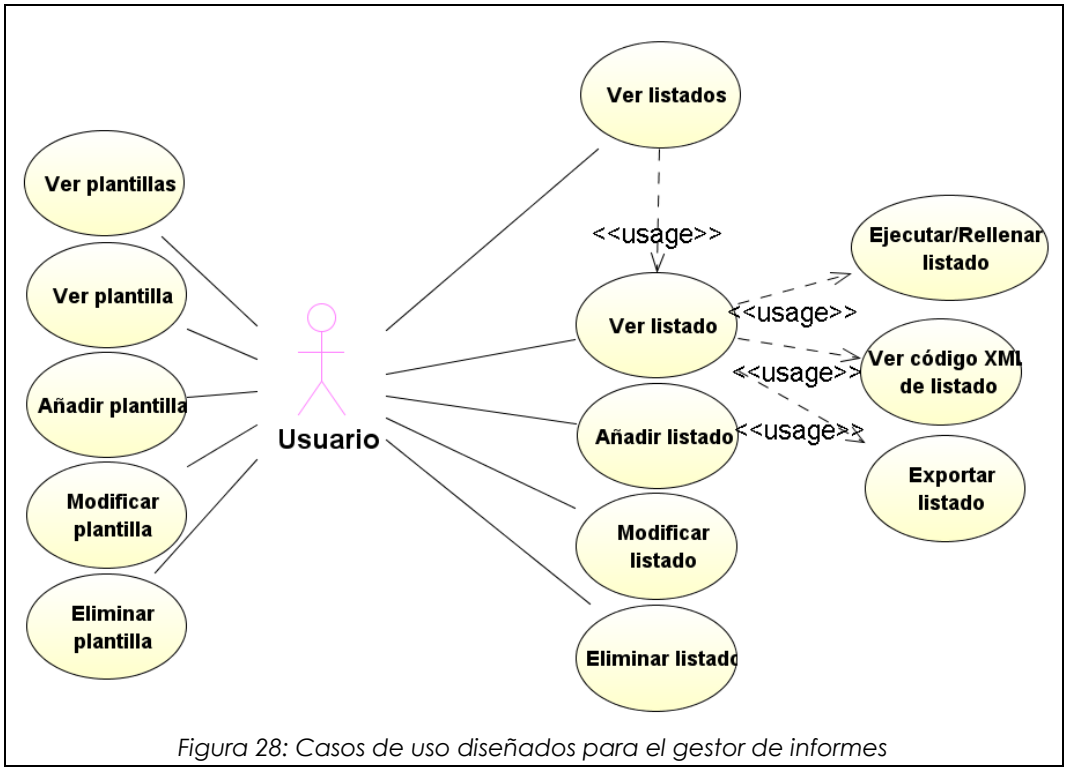


Figura 28: Casos de uso diseñados para el gestor de informes

Se tienen 12 casos usos principales que se pueden agrupar en los dedicados a las plantillas, y en los dedicados a los listados.

4.5.2 Arquitectura

Se pretende, aunque no sea estrictamente necesario, utilizar un entorno, herramientas y librerías similares a las de *SEDAS* para poder siempre utilizarlo en el mismo servidor y no aumentar los requisitos.

Por lo tanto, la arquitectura tendrá una capa de presentación similar a *SEDAS*, y se diferenciará por como se implementará la lógica de negocio, ya que se implementará en Java.

Los motivos para el cambio en la implementación de la lógica de negocio son la escalabilidad, la facilidad tanto el desarrollo como el mantenimiento y que además permite hacer el desarrollo de forma totalmente local, dejando la posibilidad de utilizar posteriormente cualquier *SGBD*.

Se añadirá una capa de persistencia de datos para separar y aislar el acceso a estos del resto de capas. Para ello se utilizará *Hibernate*, que es la librería libre más extendida y una de las más potentes al ser prácticamente un estándar en cuanto a persistencia de datos. A pesar de existir alternativas como el estándar *JPA* de *Sun*, o implementaciones como *Oracle Toplink*, *Hibernate* destaca por su soporte de la JDK versión 1.4 y ser aparentemente una plataforma más madura.

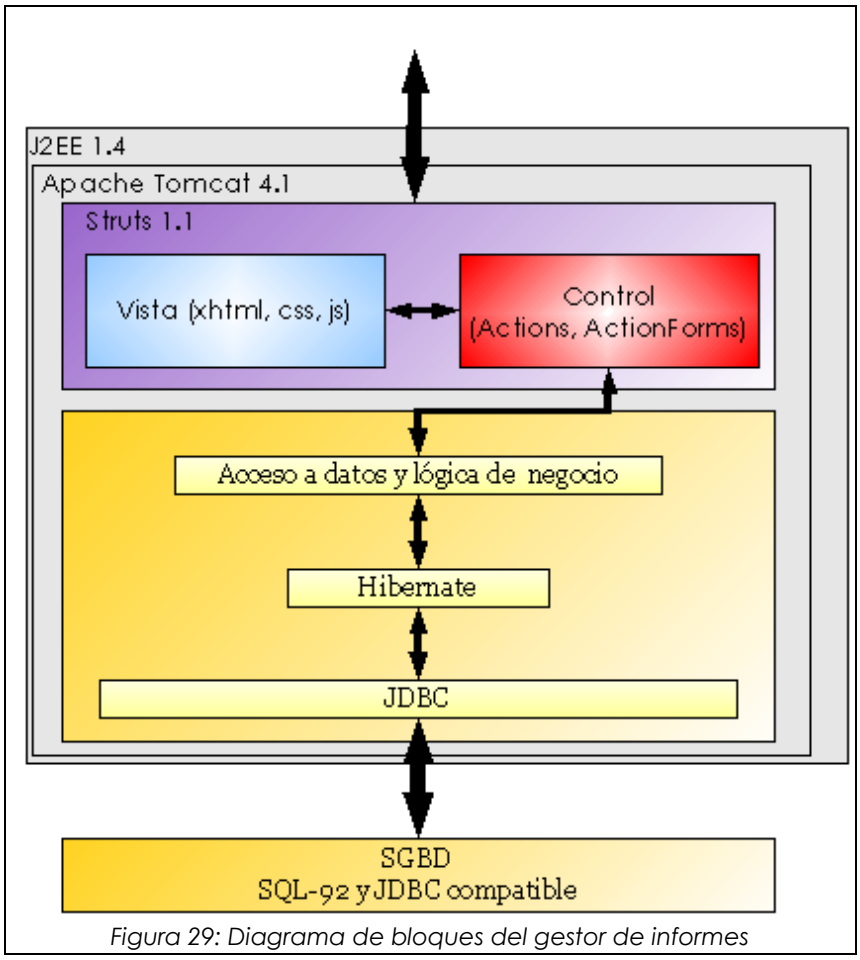


Figura 29: Diagrama de bloques del gestor de informes

4.5.3 Modelo de datos

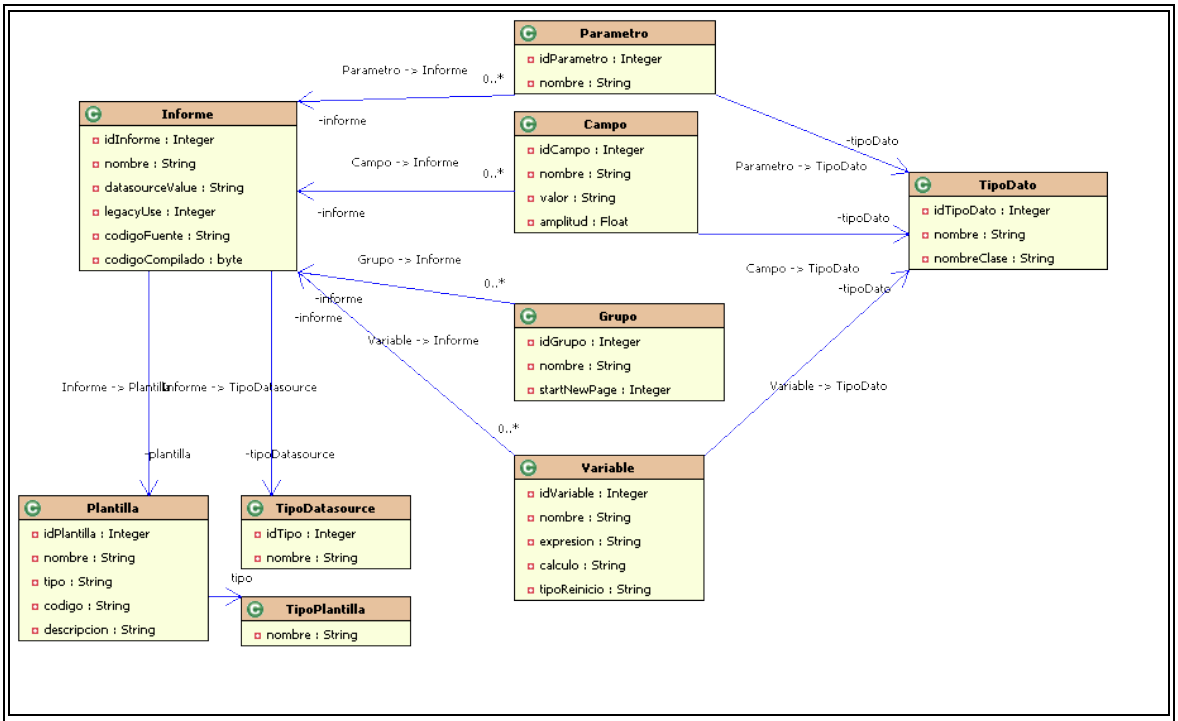


Figura 30: Modelo de datos del gestor de informes

Informe

Considérese a los informes como la generación de los listados que queremos modelar. Estos incluirán una plantilla en la cual se basa, un tipo de ***datasource*** (p.e. Consulta *SQL*, procedimiento almacenado, fuente datos *XML*, etc.) y un conjunto de colecciones de **Parámetros**, **Campos**, **Grupos** y **Variables** que acabarán de configurar el contenido del listado.

Plantilla

Serán los patrones o diferentes plantillas generales que dispondremos en los proyecto. Serán también informes pero formados por información fija especificada en *XML*. Un ejemplo de plantillas sería el siguiente:

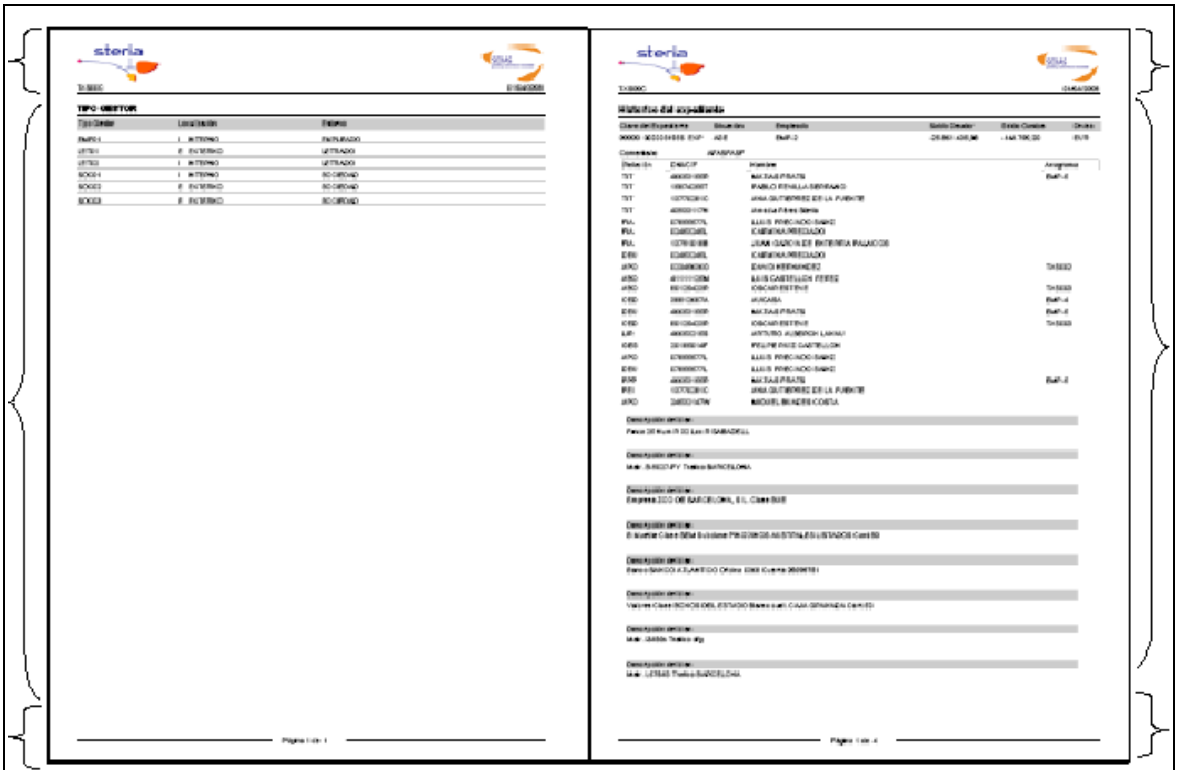


Figura 31: Esquema con dos tipos de plantillas y las partes que las componen

Ambas comparten una cabecera y pie de página idénticos, pero en cambio, el cuerpo de la primera es un listado sencilla mientras que el de la segunda es una más complejos con subinformes anidados.

En general, las plantillas guardan la cabecera, el pie de página y la organización del detalle que contendrá, mientras que el listado configurará cuales son sus parámetros específicos, cuantas columnas hay en el cuerpo, cual es la fuente de datos, etc.

TipoDataSource

Serán los tipos de fuentes de datos disponibles. En la creación del informe se tendrá en cuenta esto para configurar el listado. En el caso de ser una consulta *SQL*, tan solo se establecerá el atributo con el valor correspondiente, mientras que por ejemplo, en el caso de procedimientos almacenados como los usados en *SEDAS*, se tendrá que especificar el nombre del procedimiento y el *scriptlet* (la clase que ejecuta el procedimiento).

Parámetro

Guardará cuales serán los parámetros a tener en cuenta para configurar la fuente de datos. Por ejemplo, en una consulta donde se quisiera obtener todos los informes creados entre dos fechas determinadas, se deberían tener dos

parámetros, la fecha inicial y la final.

Campo

Los campos conformarán el cuerpo del listado. De su nombre se obtendrá la etiqueta de la columna, de su valor, el origen dentro de la fuente de datos que determinará el contenido.

Por ejemplo, si tuviésemos la fuente de datos resultante de la consulta del apartado anterior, el nombre de un campo podría ser "Expediente", mientras que el valor correspondería al nombre de la consulta en la consulta: "ID_EXPEDIENTE".

Grupo

Los grupos sirven para agrupar información utilizando un criterio determinado.

Siguiendo con el ejemplo anterior, un campo podría ser el mes en que se crea el expediente, de esta forma podríamos tener agrupados y separados en cada página los expedientes según la fecha.

Variable

En este caso, el uso principal será el de tener sumatorios al final de los listados, aunque las variables pueden contabilizar hechos utilizando un gran número de funciones matemáticas y estadísticas.

Por ejemplo, podríamos tener al final del listado la deuda total que tenían los expedientes en el momento de crearlos y la mediana de deuda de todo el conjunto.

TipoDato

En este caso servirá para saber en cual de los tipos básicos se utilizará y se mostrará en el listado.

TipoPlantilla

Este tipo servirá para saber que tipo de plantilla, podría ser que cada tipo exigiese cambiar añadir alguna funcionalidad especifica a las clases que se encarguen de construir los informes, por lo que inicialmente solo se crearán unos específicos.

4.5.4 Interfaz gráfica

La parte más importante, la de creación de los listados, estará organizada como un asistente en el cual se irá especificando cada uno de los elementos

necesarios. La interfaz será muy sencilla, por lo que se creará posteriormente directamente en HTML.

5 Implementación y pruebas

5.1 Introducción

Al igual que en el capítulo anterior, en este se detallarán aspectos relativos a la implementación y pruebas de cada uno de los cambios.

No se hará una explicación exhaustiva y se tratará de sintetizar en la medida de lo posible para no extenderse demasiado en detalles irrelevantes.

5.2 Actualización de librería de reporting

En el diseño ya se vio como se deberían implementar los cambios entre tres elementos: en los ficheros *XML* con los listados, en las clases *java* y en los *JSP*.

5.2.1 Clases Java

Lo que se modificó fueron las inclusiones para que apuntasen a la nueva librería (**net.sf.jasperreports**) y el nombre del paquete en que se encuentra cada clase que pasó a ser **com.steria.struts.jasperreport2**.

5.2.2 Ficheros XML

Estos tienen inclusiones e importaciones del paquete que hemos sustituido, por lo que lo primero es cambiar estas por la nueva, tal y como se muestra en el ejemplo:

```
scriptletClass="com.steria.struts.jasperreport.STSJRScriptlet"
scriptletClass="com.steria.struts.jasperreport2.STSJRScriptlet"
```

Una vez hechos estos cambios en todos los ficheros se procedió a compilar los informes. Su compilación dio buen resultado, excepto en algunos casos en los que el tamaño de los elementos no correspondía y fallaba la compilación. El motivo es que en las últimas versiones de la librería se han endurecido los requisitos necesarios. Para su resolución hubo que modificar ligeramente (del orden de milímetros) algunos listados.

5.2.3 Ficheros JSP

En este caso también se cambiaron las inclusiones, de nuevo se muestra un ejemplo de ello:

```
<%@ page language="java" import="com.steria.struts.jasperreport.STSPrinter,
                          com.steria.struts.action.STSTransactionActionForm,
                          com.steria.struts.action.STSMultiTransactionActionForm,
                          com.steria.struts.beans.STSResult,
```

```
                            com.steria.struts.beans.STSMultiResult,
                            com.steria.struts.beans.STSComplexResult"%>
```

```
<%@ page language="java" import="com.steria.struts.jasperreport2.STSPrinter,
                            com.steria.struts.action.STSTransactionActionForm,
                            com.steria.struts.action.STSMultiTransactionActionForm,
                            com.steria.struts.beans.STSResult,
                            com.steria.struts.beans.STSMultiResult,
                            com.steria.struts.beans.STSComplexResult"%>
```

5.2.4 Soporte iReport 2.0.4 y compilación de informes

Por último se añadió la posibilidad de abrir directamente y poder compilar tanto con la última versión estable de *iReport*, como desde el entorno de desarrollo.

Para ello se amplió el *script ant* (*build.xml*) con las siguientes reglas:

```
  <taskdef name="jrc" classname="net.sf.jasperreports.ant.JRAntCompileTask">
       <classpath refid="jaspercompile.classpath"/>
  </taskdef>

  <target name="compile_reports" description="Compiles report designs specified
using a fileset in the src tag. Generated .jasper files lose their relative
location and are placed in the same output directory.">
       <jrc destdir="${project.path}/WEB-INF/src/listados"
                      tempdir="${project.path}/WEB-INF/src/listados"
                      keepjava="false"
                      xmlvalidation="true">
              <src>
                     <fileset dir="${project.path}/WEB-INF/src/listados">
                            <include name="**/*.jrxml"/>
                     </fileset>
              </src>
              <classpath refid="jaspercompile.classpath"/>
       </jrc>
  </target>

  <target name="open_designer" >
      <copy todir="classes" includeEmptyDirs="no">
          <fileset dir="../WEB-INF/src/listados">
          <patternset>
              <include name="**/*.jasper"/>
          </patternset>
          </fileset>
      </copy>
      <exec dir="${ireport.path}" executable="cmd">
           <arg value="/c"/>
           <arg value="ireport.bat"/>
           <arg value="${project.path"/>
         </exec>
  </target>
```

Además se modificó el *script* de ejecución (*ireport.bat*) y se añadió una carpeta con la configuración de *iReport* al proyecto. El resultado obtenido es que a partir del entorno de desarrollo *StrutsStudio* se podía abrir el diseñador o compilar la totalidad de los informes con una sola pulsación de ratón.

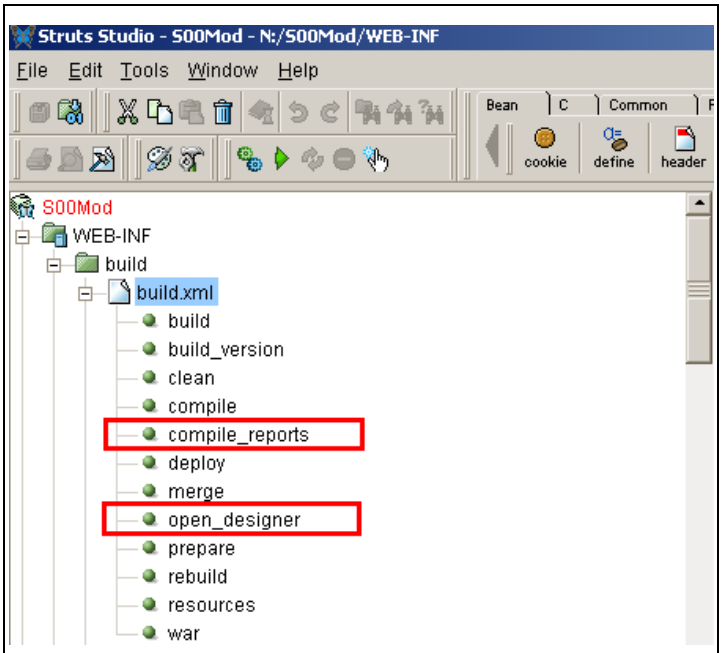


Figura 32: Resultado de la modificación del script de generación de SEDAS en Struts Studio

5.3 Exportación de listados

La implementación se realizó de acuerdo al diseño planteado.

5.3.1 Cambios en clases Java

Se modificó la clase **STPrinter**, el resultado de sus métodos es el siguiente:

```
/** Configura la impresora para generar ficheros con el formato del parámetro
especificado
*/
public void prepareDesign(String reportType)


/** Genera el fichero resultado dada la configuración de la impresora y el
informe pasado como parámetro
*/
public void generateOutput(PageContext pageContext, JasperPrint p, String
filename)


/** Configura y utiliza una instancia de la clase JRCsvExporter para generar un
fichero dado el parámetro especificado
*/
public void doCSV(PageContext pageContext, JasperPrint p, String filename)
```

La función **doPDF** se reaprovechó ya que no necesitaba prácticamente ningún cambio.

5.3.2 Código Javascript

En este caso se creó una nueva función que recibía como parámetro el nombre del formato deseado y hacia la llamada al *JSP* específico.

Para mostrar directamente el fichero CSV en el navegador, se optó por utilizar un ActiveX que forma parte de Microsoft Office Web compon*ents*, el cual permite además de mostrar el documento, poder realizar algunas modificaciones, guardarlo o abrirlo en *Microsoft Excel* directamente. Para utilizar tanto este componente como el de *PDF* en *Internet Explorer,* se utilizó el *CLSID* o identificador de cada uno de ellos.

5.3.3 Cambios en ficheros JSP

Por último, fue necesario cambiar cada uno de los *JSP* que generaban *PDF* obteniendo el parámetro enviado desde *Javascript* y haciendo las llamadas

oportunas a la clase *STPrinter* modificada anteriormente.

No se adaptaron todos los ficheros ya que la gran cantidad (del orden de centenares) hace que se necesite mucho tiempo, lo cual teniendo en cuenta los objetivos del proyecto y que tan solo era un prototipo era mejor no realizar.

5.3.4 Resultado

El resultado de la interfaz gráfica es el siguiente:

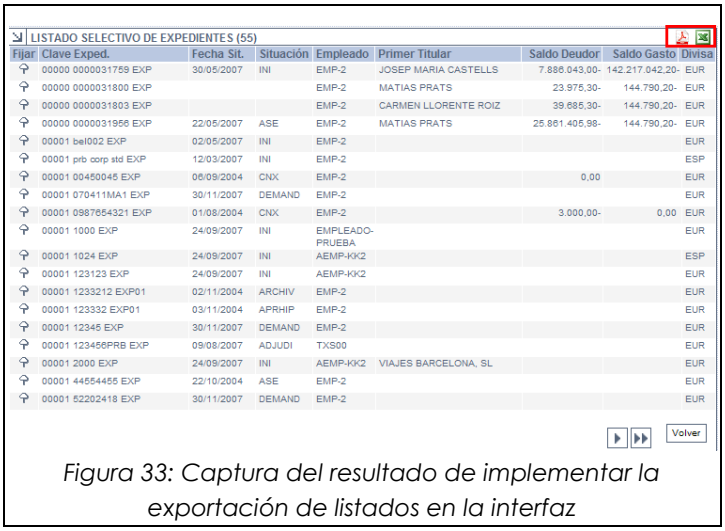


Figura 33: Captura del resultado de implementar la exportación de listados en la interfaz

Y utilizando cada una de las opciones del menú se obtiene:

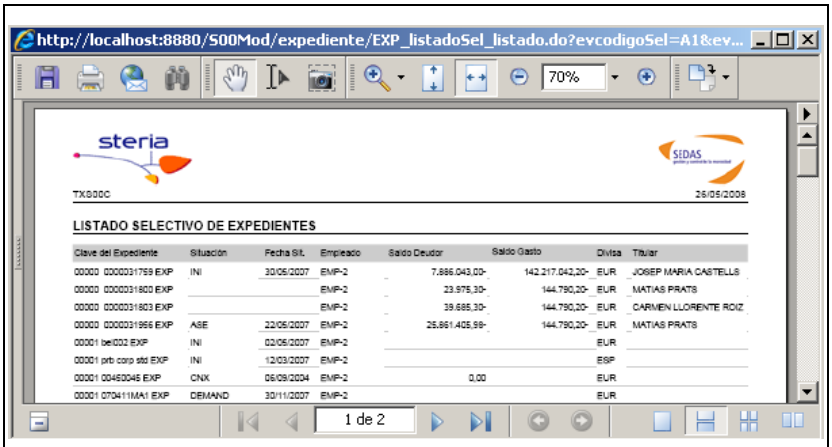
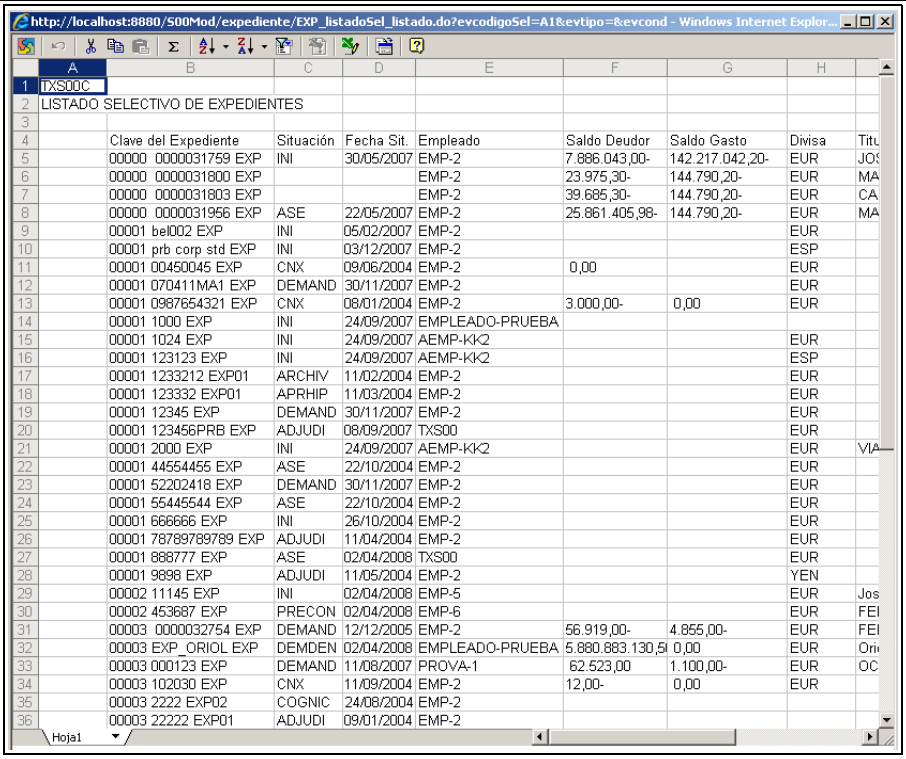


Figura 34: Listado en formato PDF generado a través del botón de exportación añadido



http://localhost:8880/S00Mod/expediente/EXP_listadoSel_listado.do?evcodigoSel=A1&evtipo=&evcond - Windows Internet Explor...

	A	B	C	D	E	F	G	H	
1	TXS00C								
2	LISTADO SELECTIVO DE EXPEDIENTES								
3									
4		Clave del Expediente	Situación	Fecha Sit.	Empleado	Saldo Deudor	Saldo Gasto	Divisa	Titu
5		00000 0000031759 EXP	INI	30/05/2007	EMP-2	7.886.043,00-	142.217.042,20-	EUR	JOS
6		00000 0000031800 EXP			EMP-2	23.975,30-	144.790,20-	EUR	MA
7		00000 0000031803 EXP			EMP-2	39.685,30-	144.790,20-	EUR	CA
8		00000 0000031956 EXP	ASE	22/05/2007	EMP-2	25.861.405,98-	144.790,20-	EUR	MA
9		00001 bel002 EXP	INI	05/02/2007	EMP-2			EUR	
10		00001 prb corp std EXP	INI	03/12/2007	EMP-2			ESP	
11		00001 00450045 EXP	CNX	09/06/2004	EMP-2	0,00		EUR	
12		00001 070411MA1 EXP	DEMAND	30/11/2007	EMP-2			EUR	
13		00001 0987654321 EXP	CNX	08/01/2004	EMP-2	3.000,00-	0,00	EUR	
14		00001 1000 EXP	INI	24/09/2007	EMPLEADO-PRUEBA				
15		00001 1024 EXP	INI	24/09/2007	AEMP-KK2			EUR	
16		00001 123123 EXP	INI	24/09/2007	AEMP-KK2			ESP	
17		00001 1233212 EXP01	ARCHIV	11/02/2004	EMP-2			EUR	
18		00001 123332 EXP01	APRHIP	11/03/2004	EMP-2			EUR	
19		00001 12345 EXP	DEMAND	30/11/2007	EMP-2			EUR	
20		00001 123456PRB EXP	ADJUDI	08/09/2007	TXS00			EUR	
21		00001 2000 EXP	INI	24/09/2007	AEMP-KK2			EUR	VIA
22		00001 44554455 EXP	ASE	22/10/2004	EMP-2			EUR	
23		00001 52202418 EXP	DEMAND	30/11/2007	EMP-2			EUR	
24		00001 55445544 EXP	ASE	22/10/2004	EMP-2			EUR	
25		00001 666666 EXP	INI	26/10/2004	EMP-2			EUR	
26		00001 78789789789 EXP	ADJUDI	11/04/2004	EMP-2			EUR	
27		00001 888777 EXP	ASE	02/04/2008	TXS00			EUR	
28		00001 9898 EXP	ADJUDI	11/05/2004	EMP-2			YEN	
29		00002 11145 EXP	INI	02/04/2008	EMP-5			EUR	Jos
30		00002 453687 EXP	PRECON	02/04/2008	EMP-6			EUR	FEI
31		00003 0000032754 EXP	DEMAND	12/12/2005	EMP-2	56.919,00-	4.855,00-	EUR	FEI
32		00003 EXP_ORIOL EXP	DEMDEN	02/04/2008	EMPLEADO-PRUEBA	5.880.883.130,5(	0,00	EUR	Ori
33		00003 000123 EXP	DEMAND	11/08/2007	PROVA-1	62.523,00	1.100,00-	EUR	OC
34		00003 102030 EXP	CNX	11/09/2004	EMP-2	12,00-	0,00	EUR	
35		00003 2222 EXP02	COGNIC	24/08/2004	EMP-2				
36		00003 22222 EXP01	ADJUDI	09/01/2004	EMP-2				

Hoja1

Figura 35: Listado en formato CSV generado a través del botón de exportación añadido

5.3.5 Pruebas

Se probó una decena de listados en ambos formatos con resultados positivos en la mayoría de casos.

En ocasiones se produjeron algunos desajustes en los ficheros *CSV* ya que el exportador analiza los informes como si de una tabla se tratase, y con tan solo detectar un milímetro de diferencia entre una etiqueta de la *cabecera* y un campo del listado, añadía una columna más desplazando todo el contenido del listado una posición hacia la derecha. Se puede comprobar en el siguiente ejemplo:

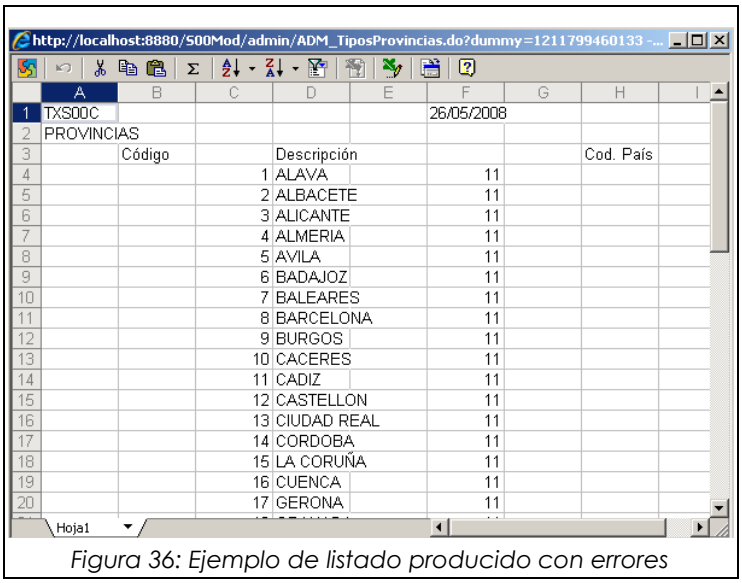


Figura 36: Ejemplo de listado producido con errores

Se puede observar como las tres columnas no coinciden. El motivo es que en el fichero *XML* su posición en el eje de las abscisas esta desplazada una unidad. Solucionarlo no supone mucha dificultad en caso de incorporarlo en las ramas no experimentales de desarrollo.

5.4 Listado de accesos a expedientes

5.4.1 Clases Java

Una vez creada la fuente de datos y el esquema en la base de datos mencionada en el diseño, se procedió a crear una serie de acciones de *Struts*.

Se han creado dos nuevas acciones, una que gestiona la carga del listado de accesos en la sesión, y otra que gestiona su actualización. Estas acciones se han introducido justo al final de flujo existente, redirigiendo en caso que no se produzca ningún error a la misma página que ya se utilizaba anteriormente.

A diferencia del resto de acciones de la aplicación, estas están implementadas en *Java* y no en procedimientos almacenados en *Oracle PL/SQL* ya que de esta forma se puede hacer sin afectar a la lógica de negocio existente.

- Flujo de abrir nueva sesión

  Al abrir la sesión se ejecuta una acción que consulta información en la base de datos y crea un listado que se va modificando en tiempo de ejecución según las acciones del empleado.

  Cabría remarcar, que para no reducir el *throughtput* del *SGBD* la información para confeccionar el listado se consulta únicamente al iniciar sesión y al fijar un expediente. Por lo que se pueden producir incoherencias, aunque no errores, en el orden del listado si un empleado utiliza simultáneamente dos sesiones de la aplicación.

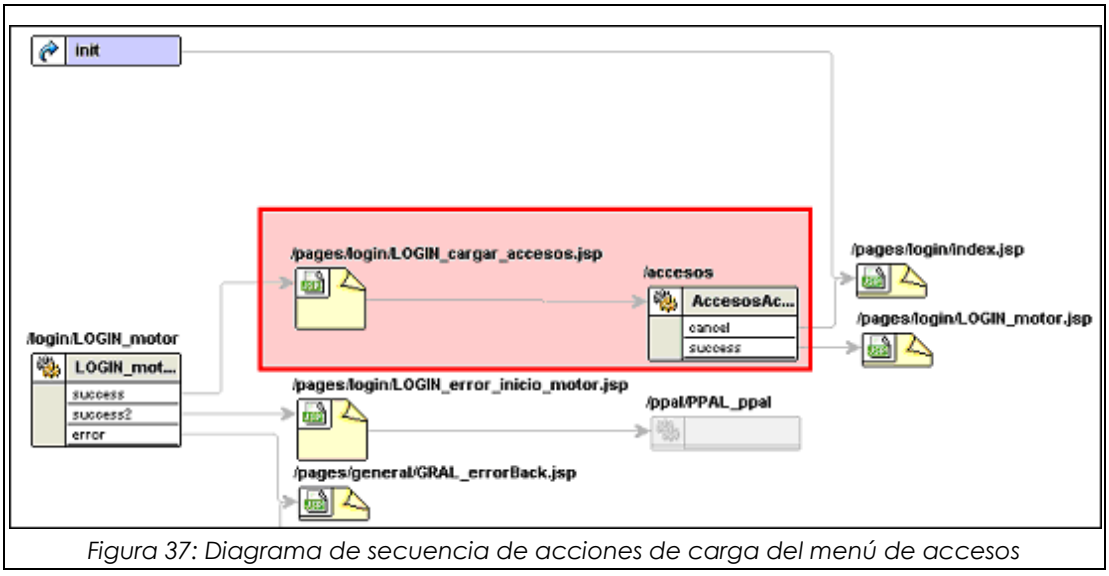


Figura 37: Diagrama de secuencia de acciones de carga del menú de accesos

- Flujo de actualización del listado

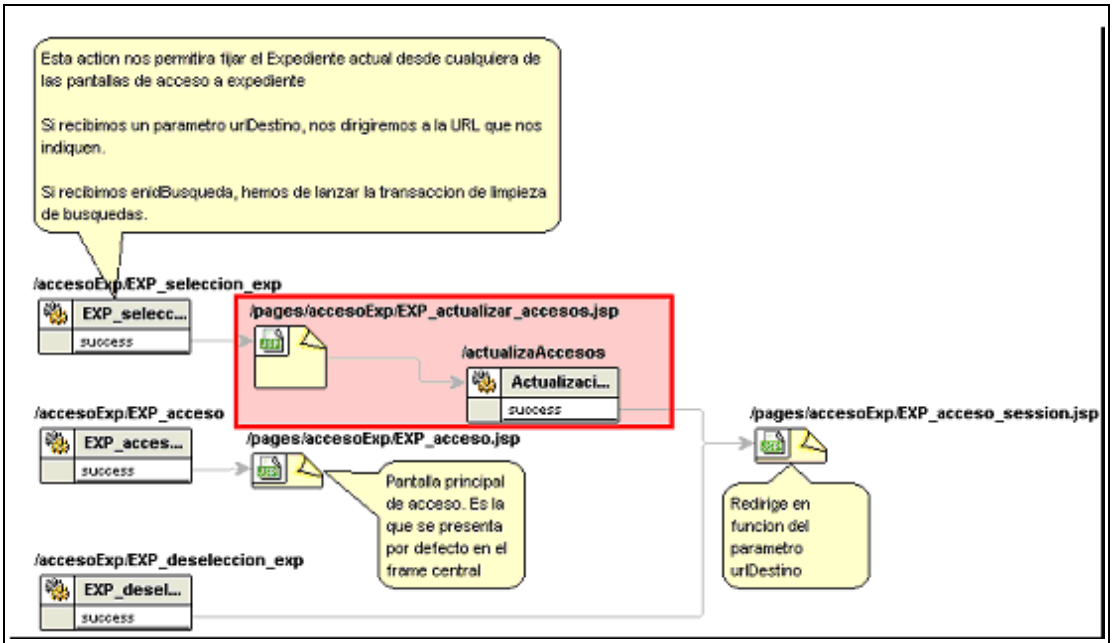


Figura 38: Diagrama de secuencia de acciones de actualización de menú de accesos

En total se han añadido dos acciones en forma de clases Java:

- CargaUltimosExpedientesAction
- ActualizaListaAccesosAction

Y un *formbean*:

- AccesosExpedientes

Que en cada instancia contiene los datos correspondientes a un registro de la base de datos.

5.4.2 Interfaz gráfica

El objetivo inicial era crear un menú desplegable similar a los que ya se tienen. Durante el desarrollo han aparecido algunas dificultades:

- El menú desplegable disponible se genera con *Javascript* y debido a la parametrización y la gestión de permisos de los usuario, no se pueden añadir opciones directamente, sino que hay que modificar la base de datos añadiendo entradas en diferentes tablas. Esto último no puede hacerse por compartir esta estructura de datos con otras ramas de desarrollo del proyecto *SEDAS*.

  A causa de lo anterior, se decidió crear un menú independiente utilizando

tan solo *CSS*.

- Se hicieron gran cantidad de pruebas que en su mayoría acabaron siendo satisfactorias en el caso del navegador *Mozilla Firefox* y que en ningún caso funcionaron en *Internet Explorer*.

  Después de muchos intentos se dio con el origen de todos los problemas surgidos para su creación: La aplicación no especifica el *DOCTYPE* por lo que los navegadores hacen una suposición de la versión de *HTML* en que se creó (*HTML 3.0, 3.2, 4.01, XHTML*, etc.). Al hacerlo, *Internet Explorer* carga la aplicación con una especificación que no contempla muchas de las funcionalidades de las hojas de estilo, haciendo imposible desplegar y ocultar el menú.

  Añadiendo el *DOCTYPE* se consigue hacer funcionar el menú pero se producen una serie de daños colaterales, ya que muchos elementos de la interfaz no se muestran en la posición que deberían.

Debido a todo lo anterior, teniendo en cuenta el tipo y las restricciones del proyecto, se decidió que al menos inicialmente, mientras que no se dispusiera de más libertad para terminar la implementación, se crearía todas las funcionalidades y se mostraría el menú simplemente como un formulario con un *input* de tipo desplegable.

5.4.3 Resultados

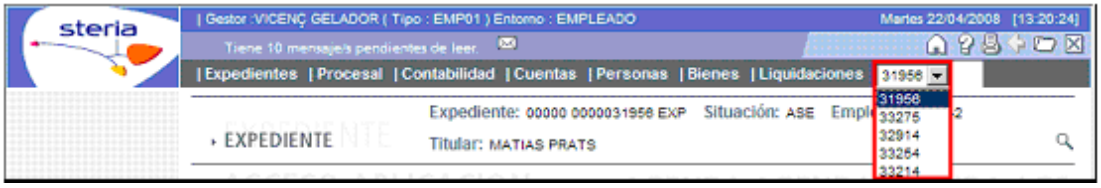


Figura 39: Captura del resultado obtenido de la implemetación del menú de accesos

Se ha creado un menú que proporciona las funcionalidades propuestas:

- Visualización de los últimos expedientes fijados por el empleado.
- Cambio rápido del expediente fijado.
- Listado de *n* elementos donde la cantidad de ellos se puede cambiar de forma sencilla.
- Listado de tipo *LRU* ordenada por fecha descendente.
- Gestión automática del listado a medida que el empleado fija expedientes se utilice o no la opción de cambio rápido.

Para su implementación real en el entorno de producción habría que considerar ciertos aspectos:

- Se debe mostrar algo más identificativo del expediente, como por ejemplo su titular si en todos los casos los clientes lo especifican. Incluso podría mostrarse el titular en todos los casos en los que se disponga, y en el resto la referencia.
- La implementación de las acciones en *Java*, a pesar de poder mantenerse, sería recomendable implementarla en *PL/SQL* para estar en concordancia con el resto del sistema. Su codificación seria relativamente sencilla de realizar en *PL/SQL*.
- A pesar de que sea aceptable el utilizar un formulario con una entrada desplegable, en entorno de producción deberían solucionarse los problemas de hacerlo con *CSS* o crear directamente un menú completamente en *JavaScript*.

5.5 Gestor de informes

5.5.1 Introducción

Como se especificaba en el diseño, se crearía una aplicación web utilizando *Struts* y *Java*.

Como *SGBD* se escogió *HSQL* ya que cumplía los requisitos que se establecieron:

- Se puede trabajar en local

  Perfecto para entornos de desarrollo, donde existe la posibilidad de provocar caídas del sistema.

- Compatible con *SQL-92*

  El sistema puede ser migrado fácilmente a la inmensa mayoría de *SGBD* como *Oracle*, que es el que finalmente se utilizaría.

- Fácilmente integrable en aplicaciones *Java*

  Tan solo es necesario distribuir una librería con la aplicaciones sin necesidad de realizar ninguna instalación ni proceso adicional. Con la misma librería se distribuye el controlador *JDBC*.

- *Free / Opensource*

  Es gratuita, de código abierto y libre distribución.

- *Footprint* reducido

  Ocupa muy poca memoria y espacio en disco.

- Buena integración con *Hibernate*

  Requisito imprescindible ya que es la librería de persistencia escogida.

El primer paso fue crear el modelo de datos diseñado

5.5.2 Resumen de librerías y recursos

- Servidor
  - Lenguaje de programación y plataforma: J2SE 1.4.12
  - Servidor web / aplicaciones: Apache Tomcat 4.1
  - Marco de desarrollo: Struts 1.2
  - Librería de persistencia: Hibernate 3.2.5

- SGDB: HSQL 1.8.0.10
- Librería de *reporting* a parametrizar: JasperReports 2.0.4
- Clientes el test
  - Navegador: Mozilla Firefox
  - XHTML 1.0
  - JavaScript
  - CSS

5.5.3 Paquetes

Para disponer de una buena base y así facilitar la mejora y mantenimiento se separaron a conciencia las clases en paquetes bien diferenciados en los cuales poder localizar fácilmente todos los recursos. En los casos en los cuales se da al usuario la posibilidad de introducir información, se han creados dos acciones diferentes, una con un sufijo **Prepare**, que carga la información necesaria para mostrar el formulario en la sesión o en la petición, y otra con el sufijo **Save** que comprueba la información introducida y la almacena o ejecuta según el caso.

Con esta separación también se consigue aislar las dependencias con librerías externas. De este modo, se tiene unos *POJO* genéricos sin ninguna dependencia (***businessobjects***), una serie de clase con dependencias hacia Hibernate (***daos***) , otras con dependencias con JasperReports (***service***) y finalmente otras dependientes de Struts (***actions*** y ***forms***). Por lo tanto, se ha conseguido dar la posibilidad de cambiar cada uno de ella produciendo una mínima afectación al resto de elementos.

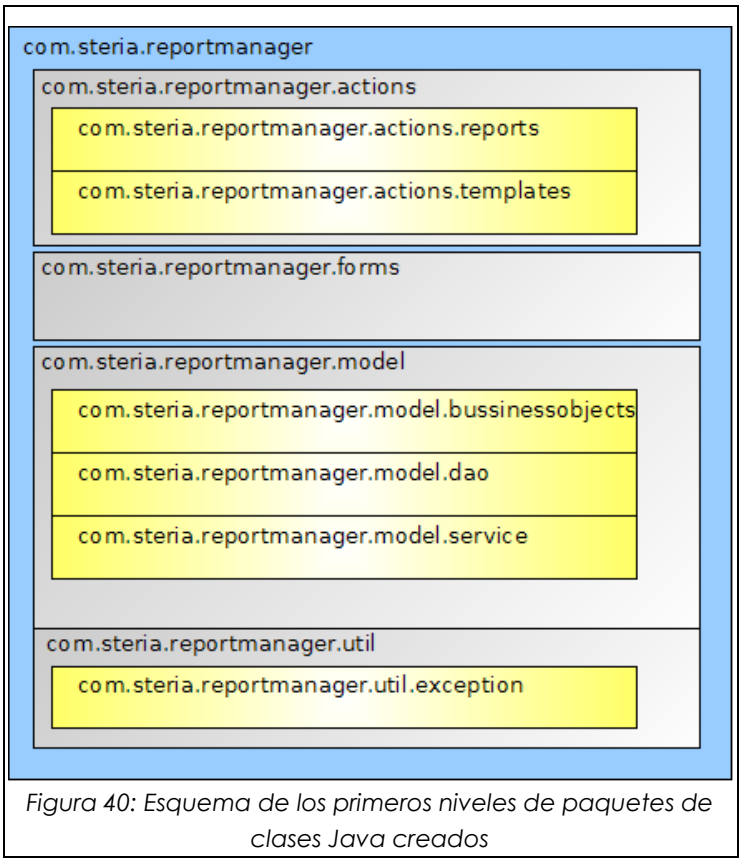


Figura 40: Esquema de los primeros niveles de paquetes de clases Java creados

- ***com.steria.reportmanager.actions***

  Este paquete contiene todas las acciones *Struts* creadas. En todos los casos son clases que extienden a *org.apache.struts.action.Action*.

- *com.steria.reportmanager.actions.**reports***

  En este caso se incluye un paquete por cada tipo de acción.

- *com.steria.reportmanager.actions.reports.**view***

  - ***ViewReportAction**.java*

    Carga la información de un formulario definido.

  - *ViewReport**Jasper**Action.java*

    Carga la información necesaria para exportar el código *JASPER* un listado concreto.

  - *ViewReport**Xml**Action.java*

    Carga la información necesaria para exportar el código *XML* un listado

concreto.

- ***ListReportsAction**.java*

  Carga el listado con todos los informes disponibles.

- *com.steria.reportmanager.actions.reports.**edit***
  - *EditReport**BasicPrepare**Action.java, EditReport**BasicSave**Action.java*

    Define la información necesaria para modificar la información básica de un listado: su nombre y tipo de plantilla.

  - *EditReport**DatasourcePrepare**Action.java,*
    *EditReport**DatasourceSave**Action.java*

    Define la información necesaria para modificar el tipo de origen de datos y su valor.

  - *EditReport**ParametersPrepare**Action.java,*
    *EditReport**ParametersSave**Action.java*

    Define la información necesaria para modificar los parametros del informe.

  - *EditReport**FieldsPrepare**Action.java, EditReport**FieldsSave**Action.java*

    Define la información necesaria para modificar los campos del informe.

  - *EditReport**GroupsPrepare**Action.java, EditReport**GroupsSave**Action.java*

    Define la información necesaria para modificar los grupos del informe.

  - *EditReport**VariablesPrepare**Action.java,*
    *EditReport**VariablesSave**Action.java*

    Define la información necesaria para modificar las variables del informe.

  - EditReport**ChangeLegacy**Action.java

    Esta acción cambia el estado del informe para cambiarlo entre el modo normal, en el cual se utiliza toda la parametrización definida en el asistente de creación o de edición de informe, o el modo *legacy*/compatibilidad/producción, en el cual solo se tiene en cuenta el código *XML* y el *JASPER*.

- EditReport**Menu**Action.java

  En este caso se carga el informe que se desea modificar en la sesión para poder mostrar la información que contiene y los enlaces a diferentes acciones con cada uno de los formularios.

- *EditReport**MenuSave**Action.java*

  Esta acción es la última en la cual una vez modificado un informe se almacena en la base de datos

- *EditReport**UpdateSaveJasper**Action.java*

  Esta acción actualiza el código *XML* a través del *Jasper* existente. Solo se puede acceder si el informe está en modo producción.

- *EditReport**UpdateSaveXml**Action.java*

  Esta acción actualiza el código *Jasper* a través del *XML* existente. Solo se puede acceder si el informe está en modo producción.

- *EditReport**UploadSaveJasper**Action.java*

  Esta acción actualiza el código *Jasper* directamente a través del fichero proporcionado.

- *EditReport**UploadSaveXml**Action.java*

  Esta acción actualiza el código XML directamente a través del fichero proporcionado.

- *com.steria.reportmanager.actions.reports.**delete***
  - *DeleteReport**PrepareAction**.java, DeleteReport**SaveAction**.java*

    Muestra la información del listado seleccionado y lo elimina según la opción escogida.

- *com.steria.reportmanager.actions.reports.**new***
  - *NewReport**BasicPrepare**Action.java, NewReport**BasicSave**Action.java*

    Define la información necesaria para modificar la información básica de un listado: su nombre y tipo de plantilla.

  - *NewReport**DatasourcePrepare**Action.java,*
    *NewReport**DatasourceSave**Action.java*

Define la información necesaria para especificar el tipo de origen de datos y su valor.

- *NewReport**ParametersPrepare**Action.java, NewReport**ParametersSave**Action.java*

  Define la información necesaria para especificar los parametros del informe.

- *NewReport**FieldsPrepare**Action.java, NewReport**FieldsSave**Action.java*

  Define la información necesaria para especificar los campos del informe.

- *NewReport**GroupsPrepare**Action.java, NewReport**GroupsSave**Action.java*

  Define la información necesaria para especificar los grupos del informe.

- *NewReport**FinishPrepare**Action.java, NewReport**FinishSave**Action.java*

  Compila toda la información introducida por el usuario y da la opción de guardarla en la base de datos.

- *com.steria.reportmanager.actions.**templates***
- *com.steria.reportmanager.actions.templates.**view***
  - ***ListReportsAction.**java*

    Carga el listado con todas las plantillas disponibles.

  - ***ViewTemplateXmlAction**.java*

    Exporta el código XML de una plantilla determinada.
- *com.steria.reportmanager.**forms***

  Este paquete contiene todos los *ActionForm* utilizados.

  - ***InformeActionForm.**java*

    Este formulario contiene un informe concreto.

  - ***UploadForm.**java*

    Es un formulario especifico que contiene un fichero subido por un usuario.

- *com.steria.reportmanager.**model***

  Este paquete contiene toda las clases encargadas de representar el modelo de datos, de su modificación y de ejecutar toda la lógica de negocio asociada.

- *com.steria.reportmanager.model.**businessobject***

  El paquete contiene todas las clases del modelo de datos creadas siguiendo el patrón definido en el apartado de diseño

  - *Informe.java*
  - *TipoDatasource.java*
  - *Plantilla.java*
  - *TipoPlantilla.java*
  - *Parametro.java*
  - *Campo.java*
  - *Grupo.java*
  - *Variable.java*
  - *TipoDato.java*

- *com.steria.reportmanager.model.**dao***

  En este caso se incluye una serie de clases adaptadores que interaccionan con *Hibernate* y por lo tanto con el *SGBD* que esta librería gestiona.

  Además se incluye una clase *singleton* que aglutina el resto de *DAOs* y permite disponer de un objeto común de cada tipo para toda la aplicación, para de esta forma evitar instanciarlo en cada ocasión.

- *com.steria.reportmanager.model.**service***

  El paquete contiene clases con la lógica de negocio de nada objeto.

- *com.steria.reportmanager.**util***

  Contiene clases secundarias de apoyo.

- *com.steria.reportmanager.util.**exception***

  Este paquete contiene un conjunto de clases de excepciones personalizadas.

5.5.4 Base de datos

- **Código SQL**

```
CREATE TABLE TIPO_PLANTILLA
(
  NOMBRE VARCHAR(32)
)
CREATE TABLE PLANTILLA

  ID_PLANTILLA INTEGER NOT NULL,
  NOMBRE VARCHAR(32),
  TIPO VARCHAR(32),
  CODIGO LONGVARCHAR,
  DESCRIPCION LONGVARCHAR,
  CONSTRAINT PLANTILLA_PK PRIMARY KEY(ID_PLANTILLA)
)
CREATE TABLE TIPO_DATASOURCE(
  ID_TIPO INTEGER NOT NULL,
  NOMBRE VARCHAR(32),
  CONSTRAINT PK_TIPO_DATASOURCE PRIMARY KEY(ID_TIPO),
  CONSTRAINT UNIQ_NOMBRE UNIQUE(NOMBRE)
)
CREATE TABLE INFORME
(
  ID_INFORME INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  ID_PLANTILLA INTEGER,
  ID_TIPO_DATASOURCE INTEGER,
  DATASOURCE_VALUE LONGVARCHAR,
  LEGACY_USE INTEGER,
  CODIGO_FUENTE LONGVARCHAR,
  FECHA_MODIFICACION DATE,
  CODIGO_COMPILADO LONGVARBINARY,
  FECHA_COMPILACION DATE,
  CONSTRAINT SYS_CT_71 UNIQUE(NOMBRE)
)
CREATE TABLE TIPO_DATO
(
  ID_TIPO_DATO INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  NOMBRE_CLASE LONGVARCHAR,
  CONSTRAINT UNIQ_CLASE UNIQUE(NOMBRE_CLASE)
)
CREATE TABLE CAMPO
(
  ID_CAMPO INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  ID_TIPO_DATO INTEGER,
  VALOR VARCHAR(128),
  AMPLITUD FLOAT,
  ID_INFORME INTEGER,
  CONSTRAINT SYS_FK_83 FOREIGN KEY(ID_TIPO_DATO) REFERENCES
  TIPO_DATO(ID_TIPO_DATO),
  CONSTRAINT FK_ID_INFORME FOREIGN KEY(ID_INFORME) REFERENCES INFORME(ID_INFORME)
)
CREATE TABLE GRUPO
(
  ID_GRUPO INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  ID_INFORME INTEGER,
  START_NEW_PAGE INTEGER,
```

```
  CONSTRAINT SYS_FK_95 FOREIGN KEY(ID_INFORME) REFERENCES INFORME(ID_INFORME)
)
CREATE TABLE PARAMETRO
(
  ID_PARAMETRO INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  ID_TIPO_DATO INTEGER,
  ID_INFORME INTEGER,
  CONSTRAINT FK_ID_TIPO_DATO FOREIGN KEY(ID_TIPO_DATO) REFERENCES
  TIPO_DATO(ID_TIPO_DATO),
  CONSTRAINT FK_PARAMETRO_ID_INFORME FOREIGN KEY(ID_INFORME) REFERENCES
  INFORME(ID_INFORME)
)
CREATE TABLE VARIABLE
(
  ID_VARIABLE INTEGER NOT NULL PRIMARY KEY,
  NOMBRE VARCHAR(32),
  EXPRESION VARCHAR(128),
  CALCULO VARCHAR(128),
  TIPO_REINICIO VARCHAR(128),
  ID_INFORME INTEGER,
  ID_TIPO_DATO INTEGER,
  CONSTRAINT SYS_FK_102 FOREIGN KEY(ID_INFORME) REFERENCES INFORME(ID_INFORME),
  CONSTRAINT SYS_FK_103 FOREIGN KEY(ID_TIPO_DATO) REFERENCES
  TIPO_DATO(ID_TIPO_DATO)
)
```

- **Mapeado Hibernate**

Todo se realizó a través de ficheros *hbm* de mapeado, ya que la versión 1.4 de Java no soporta especificar anotaciones directamente en las clases Java.

En general el proceso de mapeo no fue muy complejo, aunque como era la primera vez que se hacia se tuvieron algunos problemas. Las relaciones con la entidad informe se mapearon como set bidireccionales uno a muchos y viceversa. Se Muestran dos secciones de ficheros de mapeado como ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.steria.reportmanager.model.businessobject">
  <class name="Informe" schema="PUBLIC" table="INFORME">
    <id name="idInforme" type="java.lang.Integer">
      <column name="ID_INFORME"/>
      <generator class="increment"/>
    </id>
    <property name="nombre" type="java.lang.String">
      <column length="32" name="NOMBRE" unique="true"/>
    </property>
    <many-to-one class="Plantilla" fetch="select" name="plantilla" not-
null="true">
      <column name="ID_PLANTILLA"/>
    </many-to-one>
    <many-to-one class="TipoDatasource" fetch="select" name="tipoDatasource" not-
null="true">
      <column name="ID_TIPO_DATASOURCE"/>
```

```
    </many-to-one>
    <property name="datasourceValue" type="java.lang.String">
      <column length="0" name="DATASOURCE_VALUE"/>
    </property>
    <property name="legacyUse" type="java.lang.Integer">
      <column name="LEGACY_USE"/>
    </property>
    <property name="codigoFuente" type="java.lang.String">
      <column length="0" name="CODIGO_FUENTE"/>
    </property>
    <property name="fechaModificacion" type="java.util.Date">
      <column length="0" name="FECHA_MODIFICACION"/>
    </property>
    <property name="binaryCode" type="blob">
      <column name="CODIGO_COMPILADO"/>
    </property>
    <property name="fechaCompilacion" type="java.util.Date">
      <column length="0" name="FECHA_COMPILACION"/>
    </property>
    <set cascade="all-delete-orphan" name="parametros" table="PARAMETRO">
      <key column="ID_INFORME" not-null="true"/>
      <one-to-many class="Parametro"/>
    </set>
    <set cascade="all-delete-orphan" name="campos" table="CAMPO">
      <key column="ID_INFORME" not-null="true"/>
      <one-to-many class="Campo"/>
    </set>
    <set cascade="all-delete-orphan" name="grupos" table="GRUPO">
      <key column="ID_INFORME" not-null="true"/>
      <one-to-many class="Grupo"/>
    </set>
    <set cascade="all-delete-orphan" name="variables" table="VARIABLE">
      <key column="ID_INFORME" not-null="true"/>
      <one-to-many class="Variable"/>
    </set>
  </class>
</hibernate-mapping>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="com.steria.reportmanager.model.businessobject">
  <class name="Parametro" schema="PUBLIC" table="PARAMETRO">
    <id name="idParametro" type="java.lang.Integer">
      <column name="ID_PARAMETRO"/>
      <generator class="increment"/>
    </id>
    <many-to-one column="ID_INFORME" insert="false" name="informe" not-
null="true" update="false"/>
    <many-to-one column="ID_TIPO_DATO" insert="true" name="tipoDato" not-
null="true" update="true"/>
    <property name="nombre" type="java.lang.String">
      <column length="32" name="NOMBRE"/>
    </property>
  </class>
</hibernate-mapping>
```

- **Interfaz gráfica**

  - Listado de informes

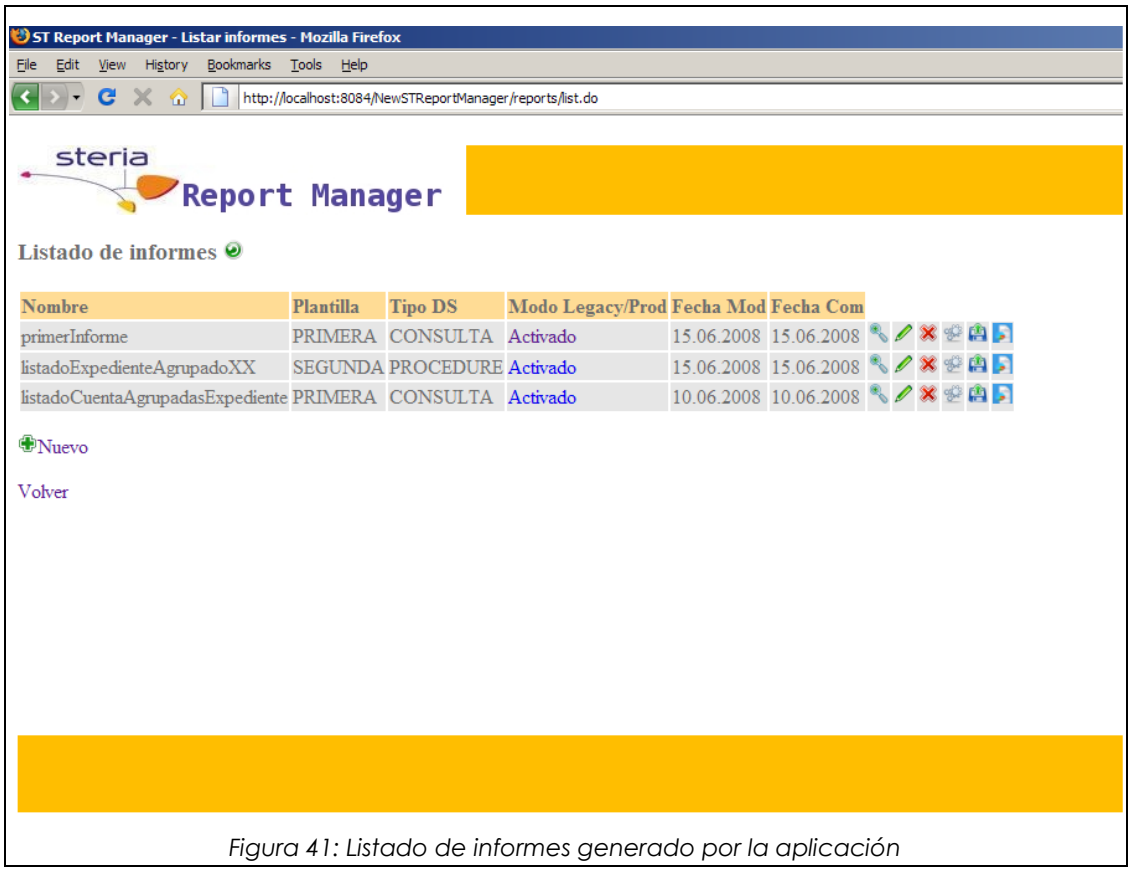


Figura 41: Listado de informes generado por la aplicación

- Listado de plantillas

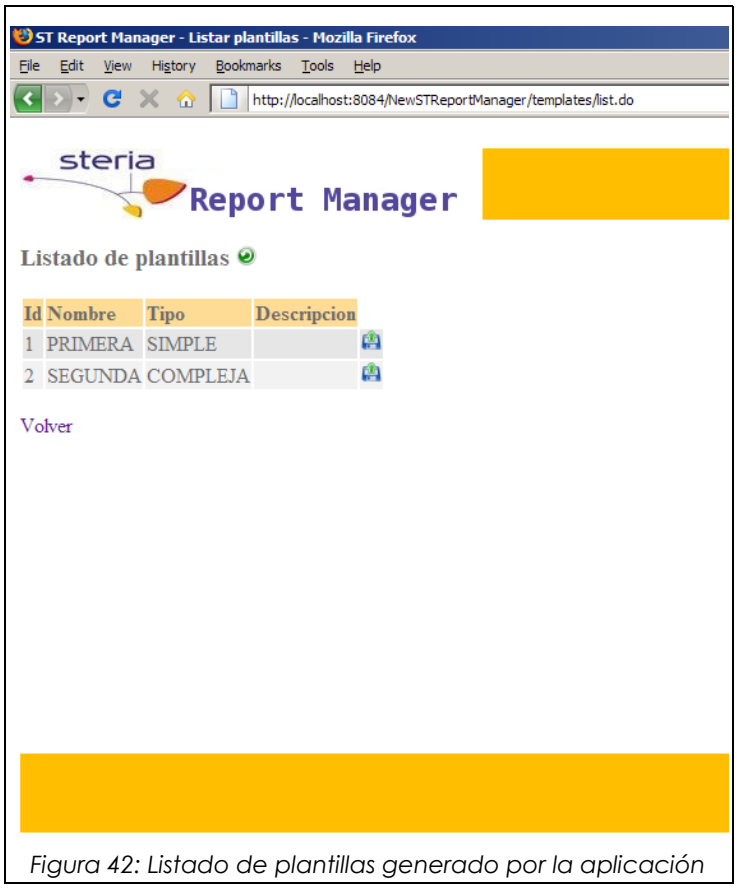


Figura 42: Listado de plantillas generado por la aplicación

- Modificación de informe

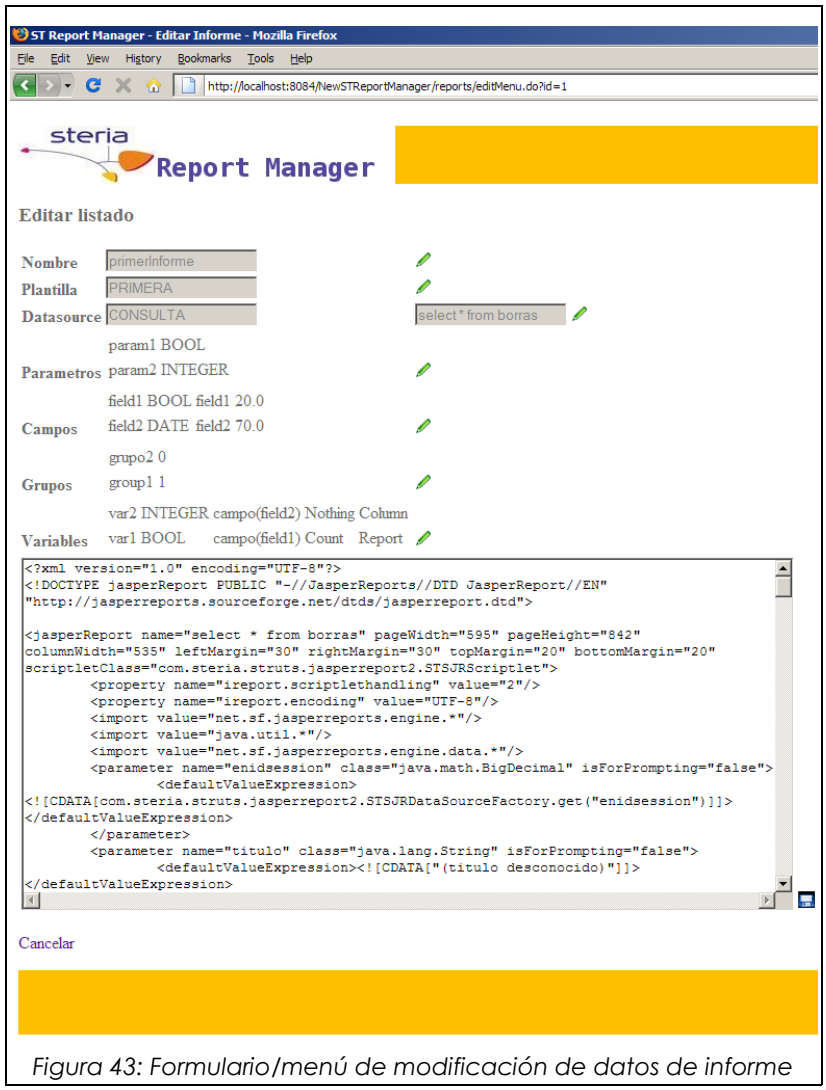


Figura 43: Formulario/menú de modificación de datos de informe

La implementación en Struts que concreta esta vista es la siguiente:

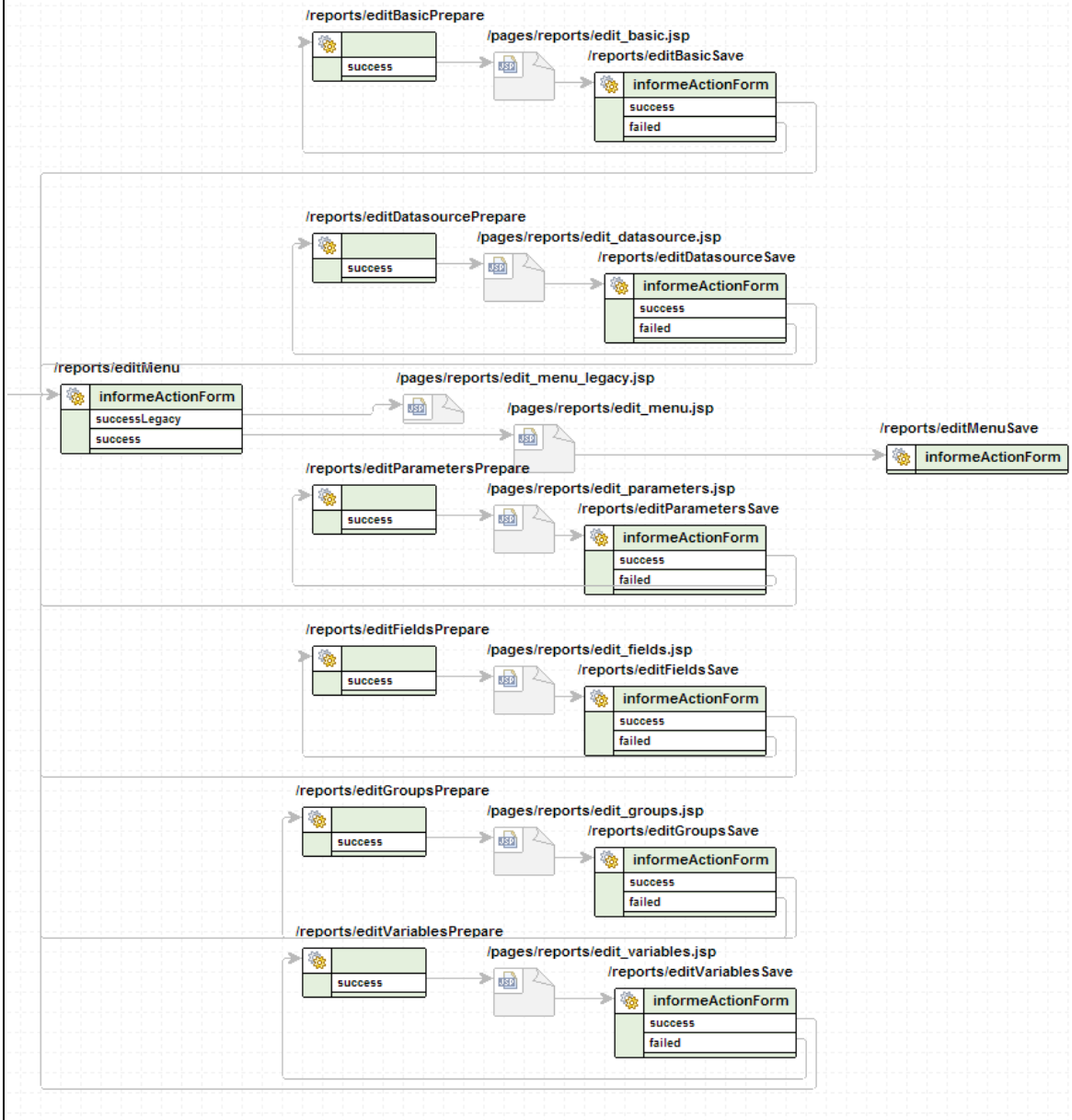


Figura 44: Implementación en Struts del menu de edicion de listados

Se ha simplificado el esquema para poder mostrarlo en esta memoria. Se puede observar donde hay un elemento **/report/editMenu**, que es el que prepara la información del informe que se desea modificar, y que muestra un formulario JSP dependiendo si el listado está en modo producción o no (***edit_menu_legacy.jsp*** y ***edit_menu.jsp*** respectivamente). A través de ese *JSP* se puede ejecutar

cualquiera de las otras de acciones ***/reports/edit*Prepare*** o la que finalmente se encargará de guardar los cambios: ***/reports/editMenuSave***.

- Creación de informe

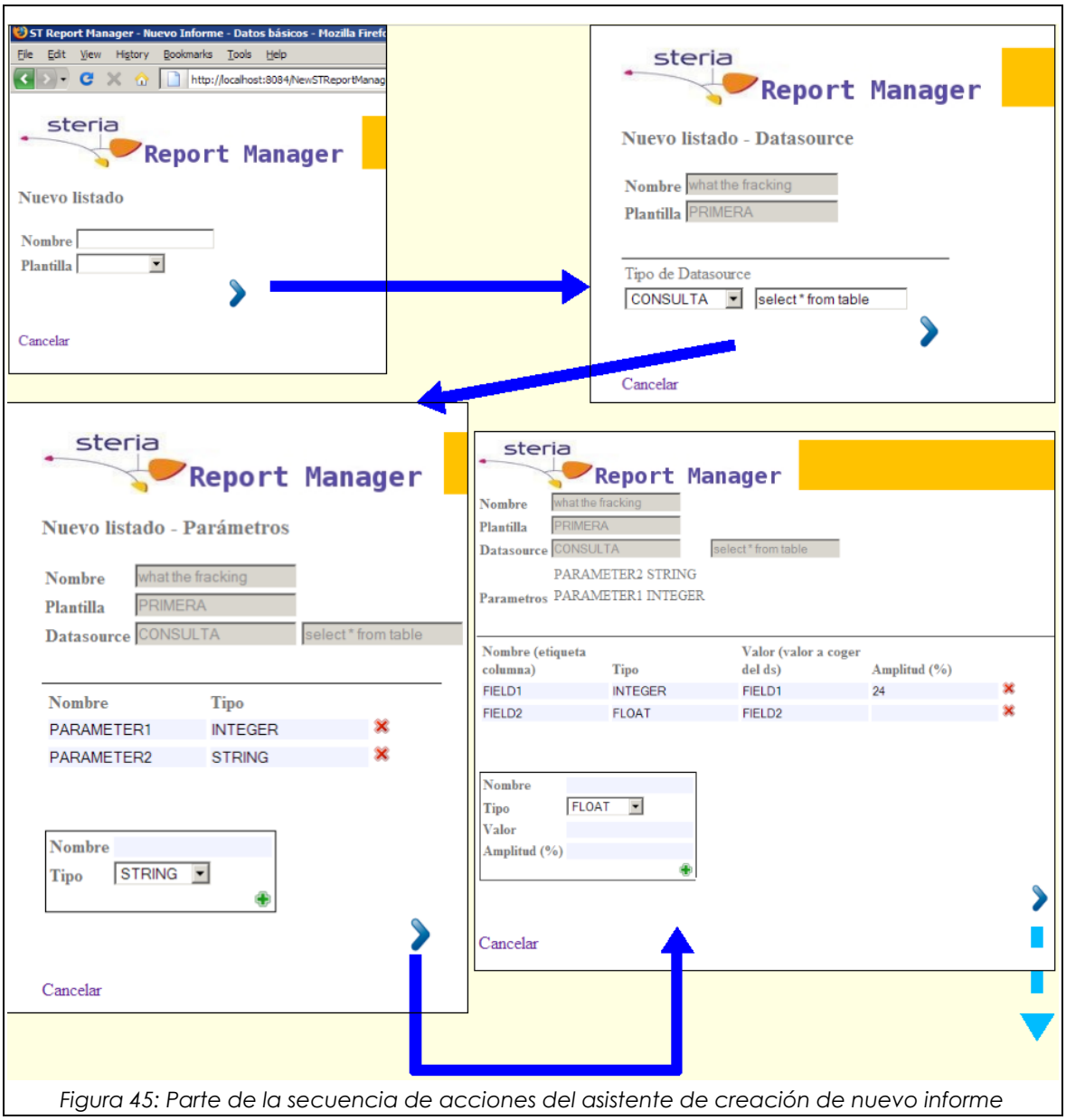


Figura 45: Parte de la secuencia de acciones del asistente de creación de nuevo informe

La misma secuencia está definida en el fichero de configuración de Struts tal y como se muestra en la siguiente figura:

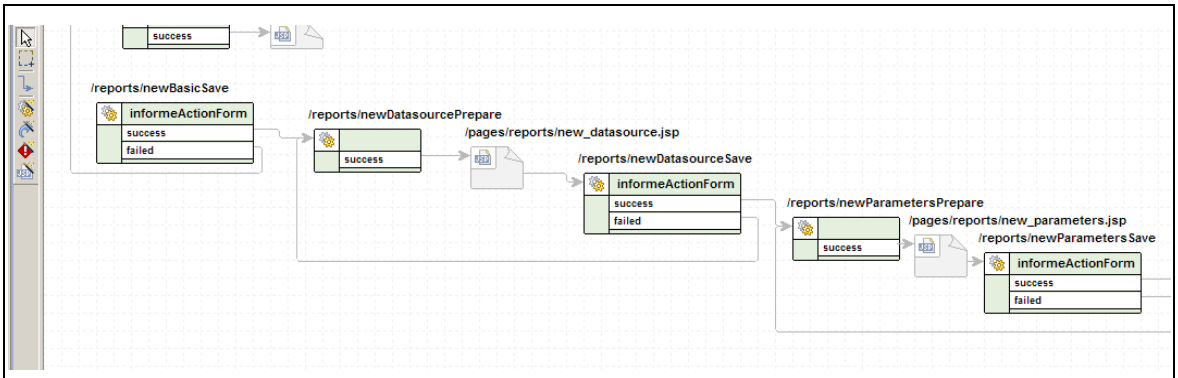


Figura 46: Parte inicial de la secuencia de acciones Struts del asistente de nuevo informe

Como comentábamos anteriormente, antes de generar cada página JSP se ejecuta una acción que prepara la información que se mostrará y otra que recibe la información introducida y actúa en consecuencia. En este caso, si la información es correcta se va al siguiente paso del asistente, y si es incorrecta se vuelve al mismo formulario. Por supuesto, siempre se da la posibilidad de salir del asistente en cada página.

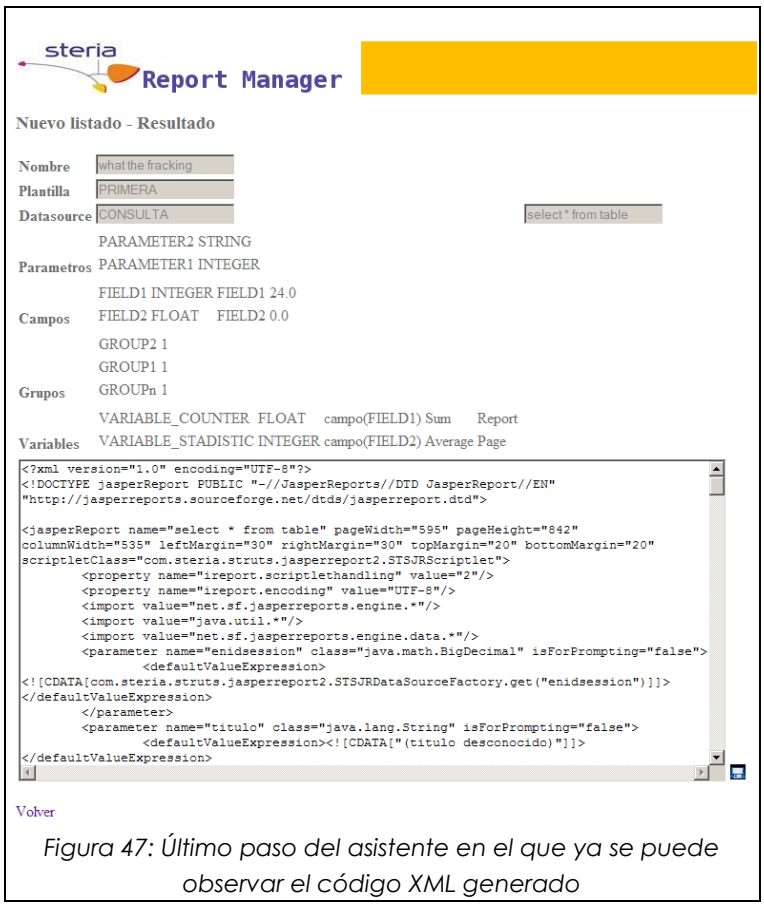


Figura 47: Último paso del asistente en el que ya se puede observar el código XML generado

Finalmente se muestra toda la información obtenida y el código *XML* generado. Para terminar el proceso tan solo es necesario pulsar la opción de guardar para disponer de todo ello en la base de datos. El resultado podemos visualizarlo a través de *iReport* en la siguiente figura:

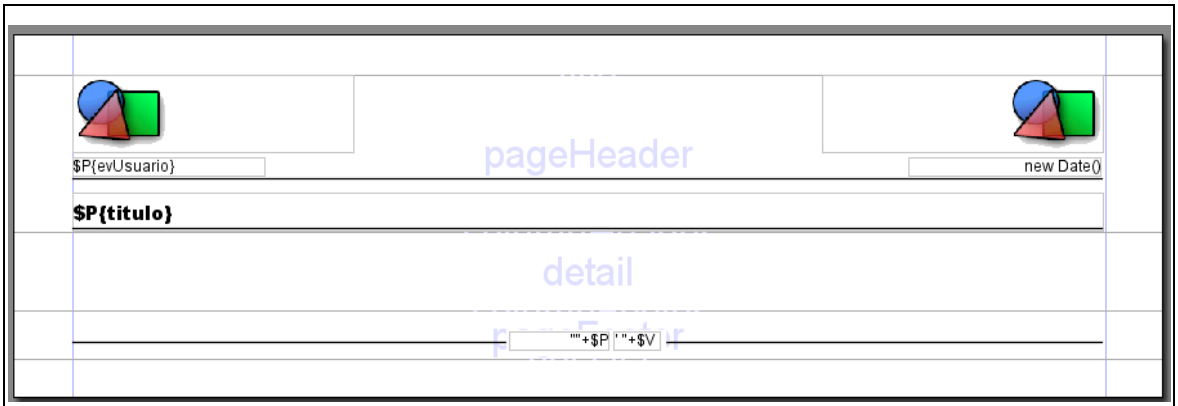


Figura 48: Vista en iReport de la plantilla de la cual se parte

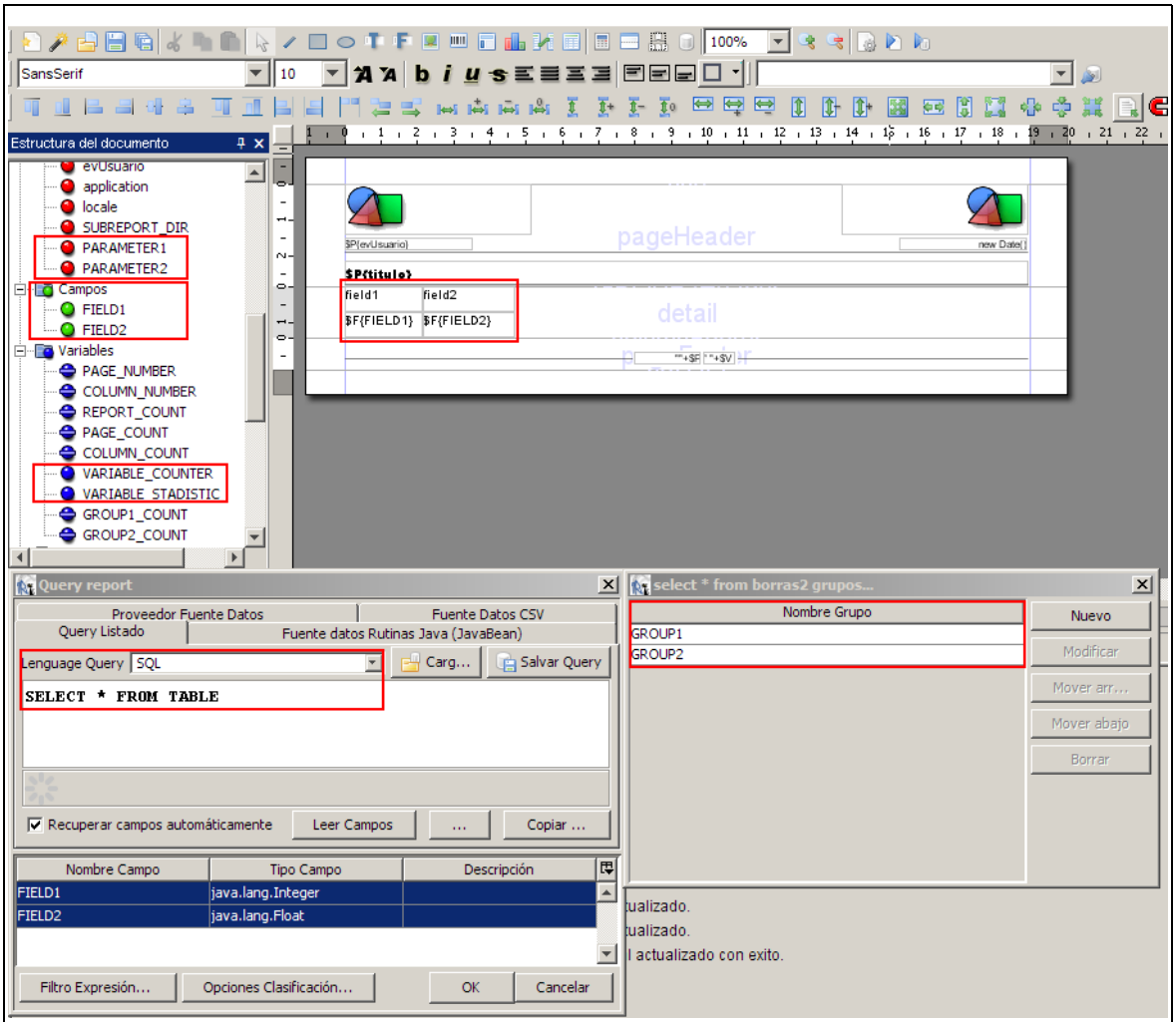


Figura 49: Vista en iReport del resultado de unir la plantilla con los parametros a través de ST Report Manager

5.5.5 Resultados

Se ha creado un primer prototipo que permite realizar cierta gestión de listados así como su generación de forma trivial. Se ha hecho especial hincapié en crear una base sólida que ayude posteriormente a su mantenimiento y también cree un precedente sobre el trabajo que se llevará a cabo el estudiante posteriormente.

No ha sido posible terminar todos los apartados que se propusieron en un inicio por razones de tiempo y recursos. El desarrollo de la base de la aplicación, así como la documentación que se tuvo que seguir, hizo que se dedicase bastante más tiempo a ello que a la implementación de otras funcionalidades igualmente necesarias, por lo que de cara a su uso final en el proyecto *SEDAS* o otro en el cual se pudiese enmarcar habría que realizar algún ajuste.

Aun así, el balance es muy positivo ya que se ha logrado un buen resultado y se han aprendido gran cantidad de conceptos y habilidades que serán indispensables en los objetivos propuestos tras este proyecto.

6 Conclusiones

6.1 Conclusión

Con la realización de este proyecto se ha conseguido en gran medida lograr los objetivos propuestos.

Se ha realizado el análisis de una serie de problemas y limitaciones existentes en un gestor de expedientes, proponiendo opciones para resolverlos, obteniendo resultados válidas en la mayoría de los casos.

Se ha logrado desarrollar un conjunto de mejoras y ampliaciones al proyecto SEDAS entre las cuales destaca el análisis y desarrollo de las siguientes características:

- Actualización de la librería de reporting.
- Parametrización del sistema de reporting para permitir la exportación en diferentes formatos.
- Añadida funcionalidad que controla y actualiza la lista de accesos a expedientes por parte de los empleados.
- Se ha creado un gestor de expedientes que permite tener un mayor control sobre los listados utilizados por el sistema y además generar otros de forma sencilla.

Otra cosa conseguida ha sido fruto de la formación recibida y el trabajo en un entorno laboral real con el cual el alumno ha podido enriquecerse personal y profesionalmente.

6.2 Revisión de la planificación

Una vez finalizada la fase de análisis se revisó la planificación para separar las fases de diseño y planificación en las *n* necesarias para cada desarrollo previsto. El resultado puede verse en el siguiente diagrama:

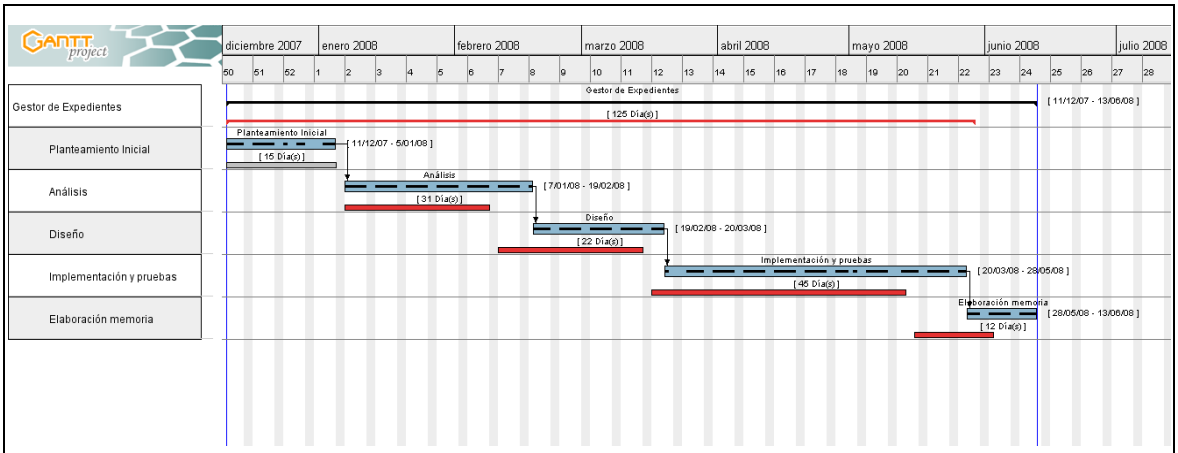


Figura 50: Diagrama de Gant con comparación entre planificación inicial y desarrollo

El diagrama muestra una comparación entre el desarrollo final, representado en azul, y la planificación inicial realizada que se muestra en la parte inferior de cada tarea, y de color rojo en las tareas que se finalizaron más tarde de lo previsto.

A tener en cuenta que la desviación que se observa está producida por una prolongación de la fase de análisis, y es debida en gran parte a la incertidumbre e inexperiencia existente en este tipo de proyectos.

PROYECTO	Planificada	Real	Desviación
Fecha inicial	11/12/07	11/12/07	0
Fecha finalización	03/06/08	13/06/08	10 días
Duración	468h	500h	32h = 6,84%

PLANTEAMIENTO	Planificada	Real	Desviación
Fecha inicial	11/12/07	11/12/07	0
Fecha finalización	05/01/08	05/01/08	0 días
Duración	60h	60h	0h = 0,00%

ANÁLISIS	Planificada	Real	Desviación
Fecha inicial	07/01/08	07/01/08	0
Fecha finalización	09/02/08	19/02/08	10 días
Duración	100h	124h	24h = 24,00%

DISEÑO	Planificada	Real	Desviación
Fecha inicial	11/02/08	07/01/08	4 días
Fecha finalización	15/03/08	20/03/08	5 días
Duración	100h	88h	-12h = -12,00%

IMPLEMENTACIÓN	Planificada	Real	Desviación
Fecha inicial	17/03/08	20/03/08	3 días
Fecha finalización	16/05/08	28/05/08	12 días
Duración	160h	180h	20h = 12,50%

ELABOR. MEMORIA	Planificada	Real	Desviación
Fecha inicial	16/05/08	28/05/08	12 días
Fecha finalización	03/06/08	13/06/08	10 días
Duración	48h	48h	0h = 0,00%

Finalmente, el diagrama de *Pert* con las fecha de inicio fin y duración de cada tarea:

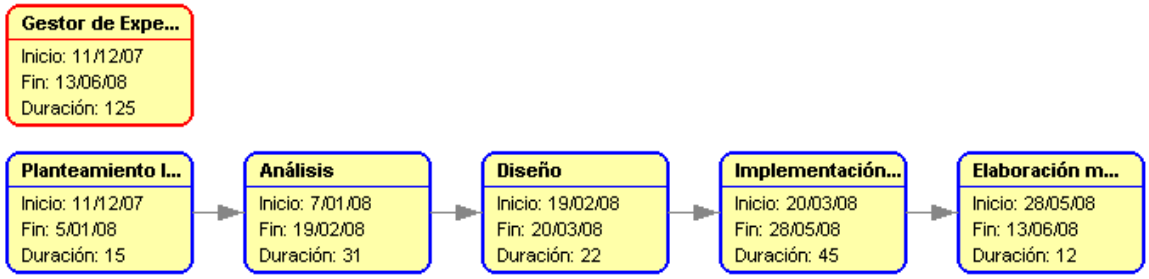


Figura 51: Diagrama de Pert con el desarrollo final del proyecto

6.3 Lineas de ampliación

Cada una las partes implementadas puede ser objeto de mejora, ya que simplemente la experiencia obtenida a partir del desarrollo de las últimas aportó idea para mejorar las anteriores. Con todo lo implementado se dispone de un prototipo que perfectamente podría ser aplicable a cualquier versión del sistema para el cual se requieran las funcionalidades desarrolladas.

Además, algunas de las propuestas analizadas que se descartaron en primera instancia podrían ser una buena ampliación del presente proyecto.

La línea abierta para la ampliación del gestor de informes creado es muy amplia, y podría extenderse en gran medida utilizando ideas que fueron surgiendo durante el desarrollo de este proyecto. La más interesante sería su mejora para poder potenciar su funcionamiento como diseñador de listados, a pesar que ello podría eclipsar algunas de las ventajas que supone la sencillez en cuanto a la creación de informes.

6.4 Valoración personal

El proyecto ha supuesto una excelente experiencia y buena forma de poder finalizar temporalmente el periodo formativo para comenzar una carrera profesional utilizando todos los conocimientos acumulados durante los años anteriores.

7 Referencias

7.1 Material impreso

7.1.1 Documentación interna de la empresa

- Howto PLSQL – Struts (versión Septiembre 2006)

  El documento describe a modo de referencia aspectos a tener en cuenta en el desarrollo de aplicaciones web que hagan uso de *Struts* y un sistema transaccional basado en *Oracle PL/SQL*.

- Reports Howto PLSQL-Struts (versión Septiembre 2006)

  El documento es una introducción a *JasperReports* e *iReport*, así como una aplicación sobre como utilizar este último para modificar o crear nuevos listados.

- Taglibs Howto PLSQL-Struts (versión Septiembre 2006)

  El documento describe el conjunto de etiquetas creado para facilitar el trabajo con ficheros *JSP*.

- Instalación SEDAS Tomcat 4.1 (versión Junio de 2006)

  Este es un tutorial sobre como instalar el *frontend* de *SEDAS* en un servidor web *Tomcat*.

- Nociones básicas de procesal (versión Febrero 2007)

  Documento en el que se explican conceptos sobre el funcionamiento de la parte procesal y aspectos relativos a la lógica de negocio específicos de la gestión de morosidad.

7.1.2 Documentación sobre aplicaciones software utilizadas o estudiadas

- Chuck Cavaness (2004): *Programming Jakarta Struts* (2ª Edición). O'Reilly Media, Inc. ISBN 0596006519.

- Chanoch Wiggers, Ben Galbraith, Vivek Chopra, and Sing Li (2002): *Professional Apache Tomcat*. Wrox, ISBN 0764543725.

- Bruce Eckel (2006): *Thinking in Java* (4ª Edición). Prentice Hall. ISBN 0131872486.

- Christian Bauer, Gavin King (2006): *Java Persistence with Hibernate*. Manning Publications. ISBN 1932394885.

- Teodor Danciu (2007): *The Definitive Guide to JasperReports*. Apress. ISBN 1590599276.

7.2 Recursos electrónicos

- Documentación oficial del servidor web Apache Tomcat versión 4.1
  - [http://tomcat.apache.org/tomcat-4.1-doc/index.html] (2005)
- Documentación sobre la JDK/J2SE 1.4.2
  - [http://java.sun.com/j2se/1.4.2/docs/api/] (2003)
- Página web oficial de Apache Struts versión 1.1
  - [http://struts.apache.org/1.1/index.html] (2003)
- Portal sobre programación Java
  - [http://programacion.com/java/] (2008)
- Sun Developer Network
  - [http://developers.sun.com/downloads/] (2008)
- Portal dedicado al SGBD Oracle y tecnologías relacionadas
  - [http://developers.sun.com/downloads/] (2008)

8 Glosario

A |

Apache Ant

Herramienta que permite realizar tareas mecánicas o repetitivas que está siendo utilizada principalmente para la compilación y generación de aplicaciones desarrolladas en Java, a pesar de no ser su único propósito. Permite disponer de un entorno y comandos similares a los que ofrecen la mayoría de sistemas operativos en un gran número de plataformas siendo portables entre todas ellas.

Herramientas que se utilizan para propósitos similares son *make*, *gnumake*, *automake* o *Maven*.

Apache Tomcat

También llamado *Jakarta Tomcat* o simplemente *Tomcat*, es un servidor de páginas web y *servlets* implementado en *Java*. Tiene una licencia muy permisiva en cuanto a su uso y está muy extendido en diversos ámbitos.

Apache Velocity

Velocity es un motor de plantillas codificado en *Java* y está especialmente indicado para su uso en *servlets J2EE*. Es software libre y tiene una de las licencias más permisivas respecto a su uso.

Action

En Struts se entiende como tal una acción o evento determinado generado por el usuario. Las acciones están implementadas como clases Java y dirigen el flujo de ejecución entre formularios o otras acciones.

B |

Backend

Parte opuesta al *Frontend*. Se considera como tal la sección de las aplicaciones o sistemas que a partir de la información que provee el *frontend* ejecuta los procesos.

Batch

En este contexto se considera como tal al proceso que se ejecuta periódicamente tareas de mantenimiento de la información. En el caso

de *SEDAS*, es el proceso que automatiza la tarea de actualiza la información contenida añadiendo personas, movimientos bancarios, bienes, etc.

Bea Weblogic

Es un servidor web y de aplicaciones compatible en las últimas versiones con la especificación 5 de la *J2EE*. Tiene una licencia propietaria y es necesario adquirirla para su uso.

C|

CLSID

También conocido como *GID* (Acrónimo de ***G****lobally* ***U****nique* ***I****dentifier*), corresponde en sistemas *Windows* al identificador de una clase *COM*. Se suele utilizar en aplicaciones web para invocar la ejecución de *applets ActiveX*.

Composite View

Patrón de diseño basado en el *composite* que permite definir estructurar las vistas (normalmente ventanas) creando jerarquías o elementos compuestos unos de otros.

CSS

Acrónimo de ***C****ascade* ***S****tyle* ***S****heets*. Es un lenguaje que permite especificar la apariencia o presentación con la cual se muestran contenido especificado en lenguajes de marcado tales como HTML, XML o SVG.

CSV

Acrónimo de ***C****omma-****S****eparated* ***V****alues*, es un formato de especificación de información en forma tabular. Para ello utiliza un carácter determinado para separar cada fila (salto de línea) y cada columna (coma). A pesar de existir un *RFC* (*RFC4180*), este solo cubre un caso concreto, por lo que puede variar la forma en que se implementa la lectura y escritura por las aplicaciones.

D|

DAO

Acrónimo de ***D****ata* ***A****cces* ***O****bject*, es un objeto que provee una interfaz genérica para obtener información desde una aplicación a una fuente de datos. Está basados en un patrón muy extendido en aplicaciones

web *J2EE*, que actúa como un conector o interfaz de acceso a la capa de persistencia.

Datasource

Fuente o origen de un conjunto de datos. Generalmente se usa como tal a los *SGBD*, pero existen otras fuentes de datos tales como ficheros planos, secuenciales, *XML*, etc.

Delegates

En este caso se refiere al patrón de diseño el cual asignación la ejecución de una determinada acción a un determinado objeto diferente del cual se le solicitado.

Dispatcher

Es un proceso que en tiempo de ejecución, asigna una determinada acción según el tipo de petición realizada.

DOCTYPE

Acrónimo de ***Doc****ument* ***Type*** *Declaration*, es la forma de especificar en *SGML* la sintaxis de un documento. El termino es muy utilizado sobretodo en aplicaciones web ya que es una directiva que permite especificar en que versión de *HTML* está expresado el documento que la contiene.

DTD

Acrónimo de ***D****ocument* ***T****ype* ***D****efinition*, es un lenguaje que permite definir la estructura de la información que puede contener un determinado documento. Su uso mayoritario es para definir la estructura de documentos *XML*.

E|

Eclipse

Entorno de desarrollo desarrollado por la *Eclipse Foundation*, mayoritariamente se utiliza como *IDE Java*, pero su estructura ha permitido que se realizaran muchas extensiones y *plugins*, por lo cual permite desarrollar aplicaciones en diversos lenguajes tales como *C/C++*, *Ruby*, *PHP*, etc.

Eclipse RCP

Acrónimo de ***Eclipse*** ***R****ich* ***C****liente* ***P****latform*. Es una base que permite desarrollar tanto *IDE*'s para cualquier lenguaje de programación, como aplicaciones de escritorio genéricas. La ventaja que aporta esta

plataforma es que ofrece la posibilidad de aprovechar funcionalidades genéricas de cualquier aplicación tales como editores de texto, interfaz de acceso a disco, gestión de actualizaciones, métodos de distribución, etc.

ECMAScript / JavaScript

Lenguaje de programación interpretado, débilmente tipado y basado en prototipos. Es el lenguaje de ejecución en el cliente más utilizado en navegadores web, aunque también se utiliza en otros entornos para tareas en las cuales se aprovecha en gran medida su tipología de lenguaje interpretado. El lenguaje está basado en una especificación de la organización *ECMA Internaltional*, de la cual se han diseñado diversas implementaciones tales como *Jscript* (*Microsoft*), *Javascript* (*Sun*), *ActionScript* (*Adobe*), *QtScript* (*Trolltech*), etc.

ERP

Acrónimo de ***E****nterprise* ***R****esource* ***P****lanning*. Se denomina como tales a los sistemas que permiten unificar todos los procesos de gestión empresariales tales como gestión financiera, recursos humanos, producción, relaciones con clientes, etc.

El término se suele utilizar a pesar de no ser del todo correcto con ciertas aplicaciones clásicas denominadas como "de gestión", que permiten gestionar más de un tipo de proceso o que al menos están diseñadas con una base que permite su fácil extensión.

Exadel Struts Studio

Entorno de desarrollo de aplicaciones *Java* creado especialmente para la creación de proyectos basados en *Apache Struts*. Desde su creación, la aplicación ha servido de base para otros IDE como *Exadel Studio PRO*, el cual está basado en Eclipse y permite trabajar con otros *frameworks* web tales como *JSF* además de *Struts*.

Desde la adquisición de la empresa desarrolladora por *Red Hat Inc* se ha estado desarrollando con el nombre de *JBoss Developer Studio*.

F

Form bean

En *Struts* se denomina como tales a clases que representan a objetos o *beans* mapeados sobre formularios *HTML* que permiten tener una manera común de interacción para gestionar su alta, modificación,

validación, etc.

Footprint

Se denomia así al tamaño que un programa ocupa en la memoria de volatil de un ordenador.

Front Controller

Es un patrón de diseño que propone unificar tareas comunes a una aplicación de forma centralizada de forma que se evite repetición de código que suponga redundancia y problemas de mantenimiento.

G|

GoF

Acrónimo de ***G****ang* ***o****f* ***F****our*. Se conoce como tal al libro escrito por *Erich Gamma*, *Richard Helm*, *Ralph Johnson* y *John Vlissides* titulado "*Design Patterns: Elements of Reusable Object-oriented software*". Esta publicación supuso uno de las primeras aproximaciones a los patrones de diseño de software, donde se explicaba de forma detallada las motivaciones, problemas, restricciones y solución a una serie de problemas utilizando un total de 23 patrones.

H|

Hibernate

Librería escrita en *Java* que permite establecer de una capa de persistencia en las aplicaciones, obteniendo de esta forma cierta independencia de esta respecto a la fuente de datos y facilitando el mantenimiento de las aplicaciones, al poder separar en dos partes muy diferenciadas el acceso a la información de su tratamiento.

Una de las funcionalidades que aporta es la del mapeo objeto-relacional (*ORM*) que permite definir como enlazar la estructura de objetos disponible en las aplicaciones a la relacional utilizada en la mayoría de *SGBD*.

Su uso implica una complejidad que puede no hacerla recomendable en casos en los cuales el rendimiento sea un aspecto crítico o el modelo de datos de la aplicación sea muy simple y no sea necesario su uso.

HSQL

SGBD relacional implementado en *Java* y compatibles con estándares como *SQL-92* muy utilizado en entornos de desarrollo y como SGBD empotrado.

HTML

Acrónimo de *HyperText Markup Language*. Es el lenguaje de marcado predominante y estándar *de facto* para la publicación de páginas web, a través del cual se expresan documentos basados en estructuras de texto.

Desde su primer esbozo en 1993 publicado por el *IETF*, se han ido realizando modificaciones debido en su gran mayoría a la fulgurante popularización de Internet y los diferentes requisitos y escenarios que de ello surgió.

Actualmente las dos versiones más utilizadas son la 4.01, estándar *ISO* de 2000, y la *XHTML* 1.0, versión basada en la 4.01 pero reformulada para su especificación en lenguaje *XML* y promovida entre otras organizaciones por la *W3C Foundation*.

I

Internet Explorer

Navegador web creado por *Microsoft*. Es el usado mayoritariamente en todo el mundo gracias a estar incluido de serie en los sistemas operativos *Windows*.

Las versiones 6 y anteriores del mismo tienen grandes problemas para el renderizado de páginas web que siguen los estándares establecidos mundialmente, por lo que suele ser fuente de problemas relacionado con su diseño.

iReport

Aplicación que permite la creación, diseño y visualización de informes basados en *JasperReports*.

J

JasperReports

Librería programada en *Java* para la creación de informes. Permite

disponer con facilidad de documentos preparados para su impresión o para exportarlos en diversos formatos tales como *PDF*, *OpenDocument*, *XML*, *XLS*, etc.

Java

Lenguaje de programación estático, orientado a objeto y fuertemente tipado desarrollado inicialmente por *Sun Microsystems* y liberado desde 2007 como parte del *Java Community Process*.

Una de las características del mismo es que el código generado, llamado *Bytecode*, está pensado para su ejecución sobre una máquina virtual, lo cual lo hace especialmente portable, siendo uno de los primeros lenguajes en proponer esta arquitectura.

Java JRE

Es la máquina virtual creada por *Sun Microsystem* para la ejecución de *bytecode*. Tal y como se distribuye permite la ejecución de aplicaciones desarrolladas con el core *Java* y la librería estándar *J2SE*.

Java JDK

Acrónimo de ***J**ava **D**evelopment **K**it*. Biblioteca y herramientas que permiten la compilación de aplicaciones *Java*.

Java J2SE

Acrónimo de ***J**ava **2** **S**tandard **E**dition*. Nombre de una parte de la *JDK* después de ser segmentada para diferenciar la parte base (*J2SE*) de la avanzada orientada a la programación de servidores (*J2EE*).

Java J2EE

Extensión de la *J2SE* para la programación de servidores.

JBoss

Servidor web y de aplicaciones mantenido por la comunidad de *JBoss* y por *Red Hat Inc.* La versión estable actual, 4.2 es compatible con *J2EE* 1.4, y está en desarrollo la siguiente versión compatible complemetamente con J2EE 5.

Es software libre y no requiere licencia para su uso y distribución.

JDBC

Acrónimo de **J**ava **D**ata**B**ase **C**onnectivity. Es un API Java para el acceso a bases de datos.

JPA

Acrónimo de ***J**ava **P**ersistence **API***. Es una especificación que trata de definir una base a partir de la cual desarrollar capas de persistencia en aplicaciones Java.

JSF

Acrónimo de ***J**ava**S**erver **F**aces*. Es un marco de desarrollo de aplicaciones Java siguiente el patrón *MVC*. Al contrario que otros *frameworks*, su diseño está orientado a facilidad el desarrollo de interfaces de usuario gracias a que está basado en componentes en vez del clásico modelo de eventos.

JSP

Acrónimo de ***J**ava **S**erver **P**age*. Es un lenguaje de marcado que permite a través de su compilación interpretación de la generación de contenido dinámico dentro de un servidor web *Java*. Está basado en *XML* siendo las etiquetas entidades especificadas en un *DTD*.

La compilación de documentos *JSP* por parte del servidor web produce *servlets* estándar.

JSTL

Acrónimo de ***J**avaServer Page **S**tandard **T**ag **L**ibrary*. Es una librería de etiquetas o *tags* que facilita la ejecución de *scripts* y creación de interfaces en *JSP*.

K |

L |

Lotus Notes

Aplicación que permite la gestión de correo electrónico, calendario, bloc de notas, así como otras funcionalidades desarrolladas utilizando su *SDK* tales como mensajería instantánea, blogs, *CRM*, etc.

LRU

Acrónimo de ***L**atest **R**ecently **U**sed*, es una política de asignación que

establece un orden en el cual se intentan mantener disponibles los elementos más utilizados e ir desechando el resto.

M

- **MVC**

  Acrónimo de ***M**odelo-**V**ista-**C**ontrolador*, es un patrón de diseño arquitectural. Su diseño es ideal para realizar interfaces de usuarios en las cuales se puede modificar el aspecto o la lógica de negocio sin por ello tener que cambiar otras partes.

  En este patrón el modelo contendría tantos los datos como la lógica de negocio, la vista la forma de representar la información, y el controlador la forma o reglas para interactuar con el usuario.

Mozilla Firefox

Navegador web mantenido por la *Mozilla Foundation* muy extendido en la actualidad. Su funcionamiento intenta seguir de cerca los estándares creados para las páginas web, dando especial importancia a la seguridad y la facilidad de uso.

Modelo Relacional

Es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos.

Model Objeto-Relacional

Es un modelo derivado del relacional que trata de reducir la distancia entre las aplicaciones orientadas a objetos y los *SGBD* relacionales añadiendo soporte para objetos en estos últimos.

Microsoft Excel

Hoja de calculo de *Microsoft* siendo mundialmente la más utilizada.

Microsoft Office Wevcomponents

Plugin diseñado como un *ActiveX* para *Internet Explorer* que permite disponer de una versión reducida de los programas del paquete *Microsoft Office* en el navegador.

N

Netbeans

Entorno de desarrollo de aplicaciones *Java* creado por *Sun Microsystems* que desde las últimas versiones ha tomado un modelo de *plugins* y extensiones similar a *Eclipse* que ha hecho aumentar su porcentaje de penetración en el mercado.

Netbeans platform

Es el equivalente a la *Eclipse RCP*, es un base a partir de la cual se pueden crear otros tipo de aplicaciones que no tienen porque ser entornos de desarrollo.

O|

Oracle

Se conoce como tal al *SGBD* distribuido por la empresa *Oracle*. Es el más extendido en bases de datos empresariales de tamaño medio-alto.

Oracle IAS

Acrónimo de ***Oracle I**nternet **A**pplication **S**erver*. Marco y servidor de aplicaciones web creado por *Oracle*.

Oracle Toplink

Es una librería de mapeo objeto-relacional (*ORM*) desarrollada por *Oracle* y similar a *Hibernate*.

P|

Package

En *Java*, estructura o jerarquía de clases. Su distribución se realiza principalmente mediante ficheros comprimidos en formato *jar*.

PL/SQL

Acrónimo de ***P**rocedural **L**anguage/**S**tructured **Q**uery **L**anguage*. Es un lenguaje propietario creado por *Oracle* como extensión a *SQL*. Su éxito ha hecho que se creen lenguajes similares en otros SGBD tales como *Microsoft SQL Server*, *PostgreSQL* o *IBM DB2*.

Pojo

Acrónimo de **P**lain **O**ld **J**ava **O**bject. Término utilizado para las clases Java que no tienen ninguna dependencia fuera de las librerías estándar.

Procedure

En este contexto puede ser entendido como tal cualquier función, procedimiento o *trigger* ejecutado dentro del *SGBD*.

Q|

R|

Refractoring

Se entiende como tal al proceso que consiste en realizar modificaciones de arquitectura o funcionamiento que consiguen adaptar o mejorar un sistema a nuevas restricciones o requisitos.

Raw

Sin un formato elaborado, la información está especificada de la forma más sencilla posible.

Report

Informe. En este proyecto se puede entender como tal cualquier listado generado por el sistema.

Reporting

Proceso que permite el trabajo con informes.

S|

Scriptlet

Script o aplicación reducida. En este caso se refiere a programas que se ejecutan durante el proceso de mapeo de los informes con los datos y que permiten que estos sean más dinámicos.

Servlet

Aplicación que se ejecuta en el lado del servidor.

SGBD

Acrónimo de **S**istema **G**estor de **B**ases de **D**atos, *DBMS* en inglés. Es un sistema genérico que permite acceder y almacenar datos.

Singleton

Es un patrón de diseño en el cual se restringe el número de instanciaciones de un determinado objeto o clase.

SQL

Acrónimo **S**tructured **Q**uery **L**anguage. Es un lenguaje de programación que permite la interacción con *SGBD* relacionales. Está especificado en estándares *ISO* y *ANSI*.

String

Cadena de caracteres

Struts

Ver *Apache Struts*

Struts²

Versión 2 del marco de desarrollo *Struts* que cambia partes fundamentales de esta ya que su base se tomó de otro *framework* llamado *Webwork*.

T|

Taglib

Librería de *tags* o etiquetas. Son un conjunto de clases que permiten definir directivas especificas. Para *JSP* existen *taglib* tales como *JSTL*, *Struts-bean*, *Struts-logic*, etc.

Tapestry

Marco de desarrollo *MVC* para aplicaciones *Java*.

TOAD

Aplicación propietaria que sirve de interfaz con diversos SGBD tales como *Oracle*. Permite realizar consultas, administrador su configuración, realizar copias de seguridad, etc.

U|

V|

W|

X|

XLS

Formato nativo que utiliza *Microsoft Excel* para almacenar las estructuras tabulares que maneja.

XML

Lenguaje de marcado que permite especificar la estructura en la cual se organiza información.

....

Bellaterra, ... de .... de ....

Abstract

The project was developed at the Sabadell headquarters of the company Steria Ibérica S.A.U. with an agreement between it, and the School of Engineering (ETSE) of the Universitat Autònoma de Barcelona (UAB, Spain).

The project consisted in the study, proposal and development of some improvements made by the result of the analysis of a file management system.

The system analysed is SEDAS, it is a product that manage debt recovering processes developed by Steria.

The improvements were concentrated in the parametrizing and improving of sections like the data model, the report generation system or the graphic interface.

Resumen

El proyecto se desarrolló en la sede de Sabadell de la empresa Steria Ibérica S.A.U. gracias a un convenio realizado entre la misma y la Escuela Superior de Ingeniería (ETSE) de la Universitat Autònoma de Barcelona (UAB, España).

El proyecto consistió en el estudio, propuesta y realización de una serie de mejoras derivadas del análisis de un sistema de gestión de expedientes.

El sistema analizado es SEDAS, un producto de gestión de los procesos de recobro de deudas desarrollado por Steria.

Las mejoras se concentraron en la parametrización y mejora de diversos apartados tales como el modelo de datos, el sistema de generación de informes o la interfaz gráfica.

Resum

El projecte es va desenvolupar a la seu de Sabadell de la empresa Steria Ibérica S.A.U. gracies a un conveni realitzat entre la mateixa, i la Escola Superior d'Enginyeria (ETSE) de la Universitat Autònoma de Barcelona (UAB, Espanya).

El projecte va consistir en l'estudi, proposta i realització d'una serie de millores derivades de l'anàlisi d'un sistema de gestió d'expedients.

El sistema analitzat es SEDAS, un producte de gestió del procés de recobrament de deutes desenvolupat per Steria.

Les millores es van concrentrar en la parametrització i la millora de diversos apartats tal com el model de dades, el sistema de generació d'informes o la interficie gràfica.