Universitat Autònoma de Barcelona

escola d'enginyeria

# 5126-1: Implementación de una herramienta de gestión de cambios dentro del ERP eSengo®, e introducción al ERP eSengo® y al Framework SherpaBeans®.

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica

Realizado por:
**Dídac Pons Aguilar**
Dirigido por:
**Pilar Gómez**

# Índice

# 1. Introducción

En el siguiente proyecto se detallan los pasos para realizar un gestor de cambios. En este apartado se explican los motivos por los que se optó realizar este proyecto, además de una introducción a diferentes conceptos generales del estado del arte, también son mencionados los objetivos del proyecto así como la estructura general de la memoria.

## 1.1. Estado del Arte

Los *ERP* (sistemás de planificación de recursos empresariales), son sistemás de información gerenciales que integran y manejan muchos de los negocios asociados con las operaciones de producción y de los aspectos de distribución de una compañía en la producción de bienes o de servicios.

En la última década el ERP se ha consolidado como producto útil para las empresas, ya sea como aplicación o aplicación web. Actualmente la gran mayoría utilizan algún ERP de más alto o bajo nivel (ya sea *opensource* o por licencias), o disponen de alguna herramienta software de administración. Por lo tanto nos encontramos en un mercado en auge, con bastantes productos para satisfacer las necesidades básicas de administración de una empresa, pero en los últimos años con la irrupción del *Cloud Computing* la industria de los ERP se ha visto convulsionada, ya que el hecho de combinar estas 2 tecnologías puede aportar beneficios, como por ejemplo:

- Permitir más accesibilidad a los empleados sin necesidad de instalar la aplicación en el terminal de trabajo.
- Facilidad para poder integrar otros sistemás, como conectar con otro ERP o conectar a un sistema auxiliar.
- Reducción de gastos del ERP (ya que con *Cloud* solo se paga por el volumen de uso, en vez de tener que pagar un servidor exclusivo).
- Disminución en el tiempo y la complejidad de desarrollo de nuevas versiones (ya que no se debe hacer ningún esfuerzo económico en mejorar las infraestructura, el software o el soporte de recursos)[1].

Además los ERP integran diferentes módulos que deben ayudar a las empresas a administrar eficientemente sus recursos así como agilizar las operaciones relacionadas con su práctica, estos módulos no son fijos, los componentes típicos de un ERP son[2]:

- Finanzas.
- Contabilidad.
- Planificación y Control de la Producción.
- Recursos Humanos.
- Costos.
- Ventas.
- Marketing.

## 1.2. Motivación

Así pues partiendo de lo expuesto en el punto anterior una empresa deberá escoger el ERP que más se ajuste a su negocio, según los módulos que implemente, y del que mejor rendimiento pueda obtener. Ese podría ser el caso de nuestro producto eSengo®, un ERP desarrollado por Isencia®[3], e implementado íntegramente en el *Cloud*, cuenta con mucha versatilidad ya que consta de diferentes módulos que siempre pueden ser fácilmente ampliables gracias a la implementación en *Cloud* debido a que no es necesario que el cliente invierta en hardware ni en capacitar personal para el uso, mantenimiento o desarrollo de estas ampliaciones. Además, eSengo®[4], consta de una alta personalización orientada al usuario, cosa poco frecuente en los ERP, gracias a que está implementado con la plataforma SherpaBeans[5],también desarrollado por Isencia®, (en el capitulo 2 y 3, se realiza una pequeña introducción tanto sobre SherpaBeans como eSengo®).

Uno de los módulos que los ERP no suelen implementar, y eSengo® no es una excepción, pero que son muy interesantes para las empresas, son los denominados *Gestores de Cambios* (normalmente basados en los *estándares de ITIL*[6]), su finalidad principal es la evaluación y planificación del proceso de cambio para asegurar que, si éste se lleva a cabo, se haga de la forma más eficiente, siguiendo los procedimientos establecidos y asegurando en todo momento la calidad y continuidad del servicio TI, en la *Imagen 1.1* se resumen las acciones que define ITIL para un Gestor de Cambios:

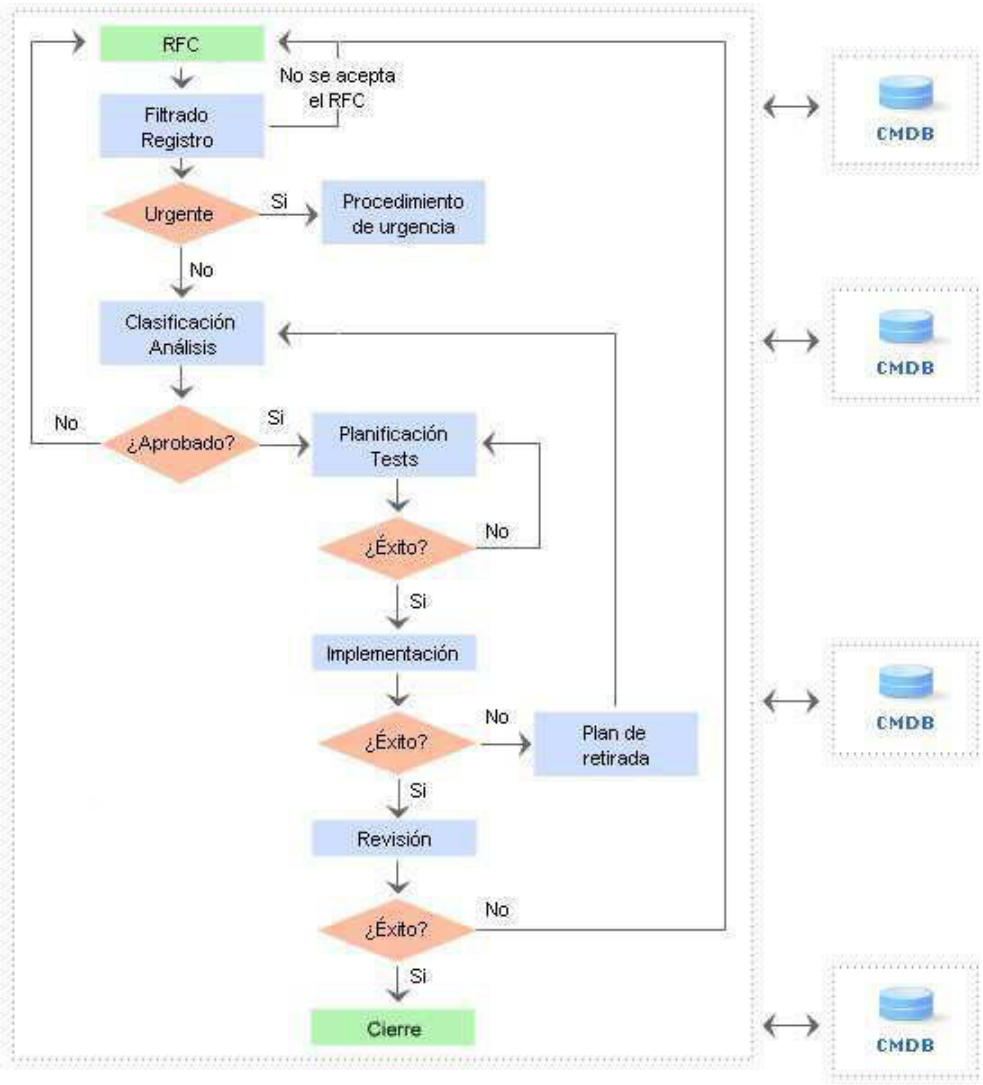


Imagen 1.1: Flujo de las acciones de un Gestor de Cambios.

Breve resumen de las acciones:

1. Filtrado/Registro: Es el estado inicial donde se reciben las RFC (Request For a Change)[7], los responsables deciden si se aceptan o no para analizar, dependiendo de diferentes varemos que ellos consideren (complicación del cambio, claridad de la explicación, etc.), además en este punto se pueden clasificar como urgentes o no, teniendo un protocolo de aplicación inmediata cuando se da el caso de una RFC urgente.

2. Clasificación/Análisis: Esta es una de las acciones más importantes del proceso ya que es donde se debe estimar exactamente el alcance del cambio, si el cambio es viable, así como el coste que pueda suponer y los posibles riesgos. Además dependiendo de este análisis se decidirá si se acepta o no el cambio, en caso afirmativo se le asignará una prioridad u otra al RFC.

3. Planificación/Pruebas: En este punto, considerando todo el análisis realizado previamente, es donde se debe planificar el tiempo que se requiere para poder hacer el cambio, revisarlo, así como los responsables de realizarlo, además se pueden establecer fechas de vencimiento dependiendo de la prioridad de este. También es donde se acordarán las pruebas a realizar para verificar que el cambio ha sido satisfactorio.

4. Implementación: Una vez pasadas todas las fases de análisis entramos en el momento de implementar (realizar) el cambio, si en algún momento de la implementación surgen incidencias no analizadas, por lo tanto no mitigadas, se debe contar con un Plan de retirada (roll-back) para poder retornar al estado estable anterior al cambio y esta petición deberá ser re-analizada.

5. Revisión: Finalmente, el cambio es probado y revisado por los responsables de dicha tarea, para asegurarse que es realmente lo que se esperaba y que no provoca ningún tipo de fallo en otras facetas en las que se vea envuelto. Si se hayan problemás, el originador del cambio no está conforme,etc. se deberá empezar el proceso desde el principio, en cambio si el resultado es satisfactorio para todas las partes, se cierra el cambio y se da el proceso por finalizado.

Algunos de los beneficios de utilizar un Gestor son[7]:

- Se reduce el número de incidencias y problemás potencialmente asociados a todo cambio.
- Se evalúan los verdaderos costes asociados al cambio y por lo tanto es más sencillo valorar el retorno real a la inversión.
- Se desarrollan procedimientos de cambio estándar que permiten la rápida actualización de sistemás no críticos.
- Se puede retornar a configuraciones estables de manera sencilla y rápida en caso de que el cambio tenga un impacto negativo en la estructura TI.

Así pues se observa que un ERP que implemente también esta herramienta será mejor valorado por las empresas ya que toda empresa está sumergida en un entorno que cambia continuamente por lo tanto tener un procedimiento que agilice y estandarice el proceso de cambiar un elemento de la empresa será beneficioso tanto a nivel interno (saber quién, cómo y el porqué de un cambio), como externo (auditorias).

Por lo tanto se ha llegado a la conclusión de que eSengo® debe incluir un modulo de Gestión de Cambios para así ofrecer mejor servicio.

## 1.3. Objetivos

El objetivo principal del proyecto es desarrollar un nuevo modulo denominado Change Manager (gestor de cambios o incidencias) que estará orientado hacia el *Change Management de ITIL* y tendrá por objetivo realizar peticiones de cambio de cualquier elemento de una empresa. Este modulo se incluirá en el ERP eSengo® (desarrollada íntegramente como aplicación web dentro del *Cloud* con la plataforma SherpaBeans) , para ello se deberán ejecutar diferentes tareas:

- Estudiar como funcionan los Gestores de Cambios bajo el estándar ITIL.
- Estudiar el funcionamiento de eSengo® y la plataforma SherpaBeans 6.0.
- Implementación de una solución que permita crear y administrar peticiones de cambio, y que sea próxima a ITIL.
- Dotar de seguridad de acceso al módulo así como a sus recursos a través de la creación de roles y permisos, además de la seguridad que ya implementa eSengo®.
- Cumplir con los requerimientos de la empresa: Incluir fechas para saber cuando se realizan las transiciones de estados de la petición, tener estados para poder controlar en todo momento en que estado se haya el RFC, incluir *dashboards*, incluir gestión de riesgos.

## 1.4. Estudio de viabilidad del proyecto

En los siguientes subapartados se expondrán los recursos necesarios para el desarrollo del proyecto.

### 1.4.1. Recursos Software

Para poder realizar el objetivo principal, implementación módulo Gestión de Cambios, se utilizará la herramienta de desarrollo *opensource* Eclipse®[8]conjuntamente con la plataforma, desarrollada por Isencia S.L, SherpaBeans. Por lo tanto se deberán utilizar diferentes conocimientos sobre programación como Java, HTML o SQL. Se realizará una breve explicación sobre algunos conceptos de eSengo® que influyen en nuestro desarrollo. También se utiliza la herramienta Dezign for Databases, como su nombre indica es una herramienta para el diseño de bases de datos y el modelado de datos para múltiples tipos de bases de datos como Oracle, MySQL, IBM DB2, etc. Como ya se comentó anteriormente en los capítulos 2 y 3 se realizará una breve explicación sobre SherpaBeans.

Como estas herramientas son propias de nuestra empresa el coste es 0. Aunque si esta herramienta se proporcionara a un cliente/empresa el coste sería solamente de 32 euros por el ERP, ya que SherpaBeans es open source y Compás® una herramienta exclusiva de Isencia®.

### 1.4.2. Recursos Hardware

Se utilizará un portátil Dell Intel Core2 Duo CPU 9700 con 2.8 Ghz, una RAM de 4 Gb y un sistema operativo de 64 bits para el desarrollo. Se usan servidores independientes para tener distintos entornos donde desarrollar los módulos, además de separar el servidor donde se encuentran las aplicaciones del servidor encargado de almacenar la base de datos. Estos dos últimos servidores tienen las siguientes prestaciones: Intel Xeon dualcore 2,13Gh con 4 gb de RAM.

### 1.4.3. Recursos humanos

En cuanto a recursos humanos:

- 1 Analista/Programador Junior(también desempeñará funciones de coordinador: pruebas, documentación, administración del proyecto).

### 1.4.4. Coste

El coste del proyecto solo supone el sueldo del analista/programado, suponiendo que ha costado unas 300-400 horas realizarlo, según ofertas de *infojobs* un Analista/Programador Junior cobra alrededor de unos 15.000 - 18.000 euros/año, por lo tanto sale a unos 7,5 euros la hora, de media. Así pues el proyecto tendría un coste de 3000 euros + 32 euros de la licencia de eSengo® = 3032.

Después de analizar los recursos software, hardware y humanos se puede concluir que el proyecto es viable.

## 1.5. Estructura de la memoria

En el primer capítulo de la memoria, Introducción, se han presentado el estado del arte, la motivación, objetivos, y viabilidad del proyecto en cuestión. Además también se explica la estructura de la memoria.

En el segundo capítulo se explican las tecnologías utilizadas por SherpaBeans, también se explica la estructura del proyecto, el entorno y la metodología de desarrollo, así como unos pequeños apuntes a modo de introducción sobre la plataforma SherpaBeans.

En el tercer capítulo se detalla la implementación del módulo, el diseño y su funcionamiento, además de algunos detalles relacionados con eSengo® y SherpaBeans, para así ampliar un poco la información del segundo capítulo. También se exponen, uno a uno, los submódulos que se han desarrollado así como el diagrama de estados del módulo general y la explicación de cada estado.

En el cuarto punto se repasa brevemente como se ha relizado la validación de cada submódulo y del módulo general.

En el quinto punto se exponen las conclusiones del proyecto así como posibles ampliaciones.

Finalmente en el último punto encontramos las referencias bibliográficas.

# 2. Tecnologías utilizadas

Durante la etapa de diseño se han debido de hacer varios aprendizajes, mediante investigación y consulta, para poder obtener el mayor rendimiento de las opciones ya implementadas por eSengo®, así como utilizar lo mejor posible el gran potencial que ofrece la plataforma SherpaBeans 6.0. A continuación se introducen brevemente las tecnologías utilizadas por la plataforma así como varios puntos a tener en cuenta de esta, de cara a facilitar la comprensión del módulo que desarrollaremos.

## 2.1. Tecnologías utilizadas por SherpaBeans

- **Java EE5**[9,10]: *Java EE (Java Platform Enterprise Edition)* es una plataforma de programación (parte de la plataforma Java), para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java, consta de una arquitectura de N capas distribuidas y se apoya ampliamente en componentes de software modulares que se ejecutan sobre un servidor de aplicaciones. Uno de los beneficios de utilizar esta tecnología es que se tiene la posibilidad de empezar un proyecto con poco o ningún coste ya que la plataforma puede ser descargada de *Sun Mictosystems®* de forma gratuita, y cuenta con muchas herramientas de código abierto, ya sean para extender la plataforma o simplificar el desarrollo. Java EE5 se centra en facilitar el desarrollo manteniendo la riqueza de *la plataforma J2EE 1.4*, para conseguirlo se ofrecen tecnologías como *JSF (Java Server Faces)*[11]o *APIs web* (Application Programming Interface)[12]. Actualmente es una de las plataforma punteras para servicios web y desarrollo de aplicaciones empresariales, por eso se creyó conveniente utilizarlo para desarrollar la plataforma de Isencia®, SherpaBeans. En la actualidad encontramos que ya existe la versión Java EE6, así pues se ha migrado la mayor parte de la funcionalidad de SherpaBeans de Java EE5 a Java EE6, pero no nos extenderemos en este tema ya que no influye en la comprensión de nuestro proyecto.

- **Web 2.0** [13,10]**:** Es una tendencia en el uso de la tecnología *World Wide Web* y de diseño web que tiene como objetivo facilitar la creatividad, el intercambio de información, y, sobre todo, la colaboración entre usuarios. Todos estos conceptos han conducido al desarrollo y evolución de las comunidades en Internet y los servicios de hospedaje como los sitios de redes sociales, wikis, blogs, etc. Los *sites* con esta tecnología permiten a sus usuarios interactuar con otros usuarios o cambiar el contenido del sitio de forma dinámica, en contraste con *sites* no interactivos donde los usuarios se limitan a mirar pasivamente la información que se les proporciona. Una de las máximás del Web 2.0 es que la aplicación desarrollada con esta tecnología mejora a medida que la utilizan más usuarios, cosa que la hace muy conveniente para aplicaciones web en las que se prevea un aumento de usuarios de forma escalada para poder beneficiarse del *feedback* con los usuarios e ir mejorando la aplicación o módulo paulatinamente.

- **Tecnología OSGI** [10]**:** El objetivo de OSGI (Open Services Gateway Initiative) es el de definir las especificaciones abiertas de software que permita diseñar plataformás compatibles que puedan ofrecer múltiples servicios. Además proporciona funcionalidad a Java que hace de este el entorno principal para la integración y el desarrollo de software, ya que OSGI define la estandarización de primitivas que permiten construir las aplicaciones con componentes pequeños, reutilizables y asociativos entre ellos, también ofrece la funcionalidad necesaria para cambiar la composición de forma dinámica en el dispositivo de una variedad de redes, sin la necesidad de reiniciar. Para minimizar el acoplamiento y facilitar la administración de componentes esta tecnología ofrece una arquitectura orientada al servicio que permite que estos componentes se encuentran dinámicamente entre si para poder colaborar. La OSGI Alliance ha desarrollado muchas interfaces de componentes estándar para funciones comunes como los servidores HTTP, configuración, registro, seguridad, administración de usuarios, XML, entre muchas otras.

- **Wicket** [10,14]**:** Es un framework de desarrollo de aplicaciones web para la plataforma Java EE, que facilita enormemente el desarrollo de dichas aplicaciones. Entre sus ventajas se debe destacar el potente depurador que tiene, así como la sencillez de código para poder implementar componentes de gran alcance y reutilizables, pudiendo ser escritos en Java y HTML. Otra de las ventajas a mencionar es que Wicket incorpora *POJO (Plain Old Java Object) Component Model* que se basa en el concepto de que las páginas y los componentes de Wicket son objetos Java que soportan encapsulación, eventos y herencia. En cuanto a seguridad tenemos que por defecto Wicket es seguro ya que las URLs no exponen información confidencial y todos los *path* de los componentes son relativos a la sesión, además la encriptación de la URL permite crear *web*s con una gran seguridad. Otra característica interesante es que todas las aplicaciones desarrolladas en Wicket trabajarán sobre un cluster sin coste adicional. Además también se facilita la escritura de aplicaciones que utilicen multi-ventanas y multi-pestaña permitiendo al desarrollador reaccionar de forma eficiente cuando el usuario abra una nueva ventana o pestaña.

- **Hibernate** [10,15,16]**:** Es una potente herramienta ORM (Object Relational Mapping) que permite mapear objetos persistentes en tablas relacionales, además incorpora servicios de consulta. Permite desarrollar clases persistentes con lenguaje orientado a objetos que permite herencia, polimorfismos, colecciones, etc. Las *consultas hibernate* pueden ser expresadas en su extensión portable del SQL (HQL, Hibernate Query Lenguage), en SQL o criterio expresado en un lenguaje orientado a objetos. Además a diferencia de otras soluciones de persistencia utilizar Hibernate ofrece ventajas como:

  - **Productividad**: Hibernate elimina mucho trabajo sucio de los códigos relacionados con persistencia y permite al desarrollador concentrarse en el problema de negocio.
  - **Mantenibilidad**: Se proporcionan buffers para hacer que la relación entre la representación relacional y la implementación del modelo del objeto sea menos crítica, otros tipos de relación implican que si en una parte de la relación hay cambios la otra también sufrirá cambios.
  - **Performancia**: Hibernate te permite aplicar técnicas de optimización a todos los elementos mientras que cuando la persistencia es implementada "a mano" la optimización suele hacerse 1 por 1.
  - **Vendor Independence**: Hibernate ofrece más portabilidad .

## 2.2. SherpaBeans y el concepto MVC

La plataforma SherpaBeans, basado en Struts[17], fue concebida para facilitar el desarrollo de aplicaciones web utilizando esta tecnología y sin la necesidad de conocer todas las tecnologías que envuelven Struts. Una de las principales ventajas de las aplicaciones web es que ofrecen un punto centralizado de configuración e instalación, por contra, las aplicaciones instaladas en el sistema operativo requieren una puesta en marcha por parte de los técnicos además de un seguimiento por si se produce algún incidente. En el caso de la aplicación web los paquetes son instalados en un servidor único y no se tiene en cuenta las particularidades de cada ordenador personal, además también ofrecen mucha portabilidad y poco consumo de recursos de la máquina, al fin y al cabo una aplicación web solo requiere del usuario tener instalado un navegador compatible, cosa con la que cuentan la mayoría de sistemás operativos de hoy en día. Para poder desarrollar este tipo de aplicaciones, SherpaBeans utiliza las tecnologías mencionadas anteriormente en un patrón de arquitectura de software, muy común en el desarrollo de aplicaciones web, denominado MVC (Modelo-Vista-Controlador) que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos, este patrón fue descrito por primera vez por *Trygve Reenskaug*, un trabajador de Smallpark en los laboratorios de investigación de Xerox. La implementación original se puede encontrar en el *document Applications Programming in Smalltalk80: How to use Model-View-Controller*[18].

Los tres componentes del modelo se definen de la siguiente manera:

- **Modelo**: Representación específica de la información con la que el sistema opera. Es importante destacar que la lógica de datos no solo facilita el acceso a estas si no que también asegura la integridad de los datos y permite la derivación de nuevas. La lógica se obtiene a partir del modelo de datos de la aplicación.

- **Vista**: Ofrece las herramientas necesarias para mostrar la interfaz de la aplicación y que el usuario pueda interactuar con ella.

- **Controlador**: Se encarga del control de flujo de la aplicación. Dependiendo de las acciones del usuario en la vista se realizarán las acciones necesarias en el modelo y se obtendrá la información a mostrar en la vista.

Son varios los beneficios de utilizar este patrón como arquitectura de trabajo. Uno de los más importantes es que favorece el trabajo en equipo y la organización del proyecto. La separación de tareas entre capas permite desarrollar paralelamente siempre y cuando se definan unas interfaces que mantengan la misma estructura. Utilizando estas interfaces se permite la integración de diferentes tecnologías en cada capa del modelo, de tal manera que se pueden buscar soluciones optimás para cada lógica. El uso de interfaces, también es un punto a favor ya que se permite establecer estereotipos de trabajo de tal manera que la implementación sea independiente a las definiciones de las clases. Finalmente una de las características más importantes es la gran escalabilidad que tiene. Se pueden modificar las tecnologías aplicadas en cada capa y permite ampliar fácilmente cualquier componente, sin que los otros se deban ver afectados en ningún caso. Una propiedad que demuestra la escalabilidad que proporciona el modelo MVC es que dentro de cada componente este permite que pueda haber otra tripleta modelo-vista-controlador y así recursivamente.

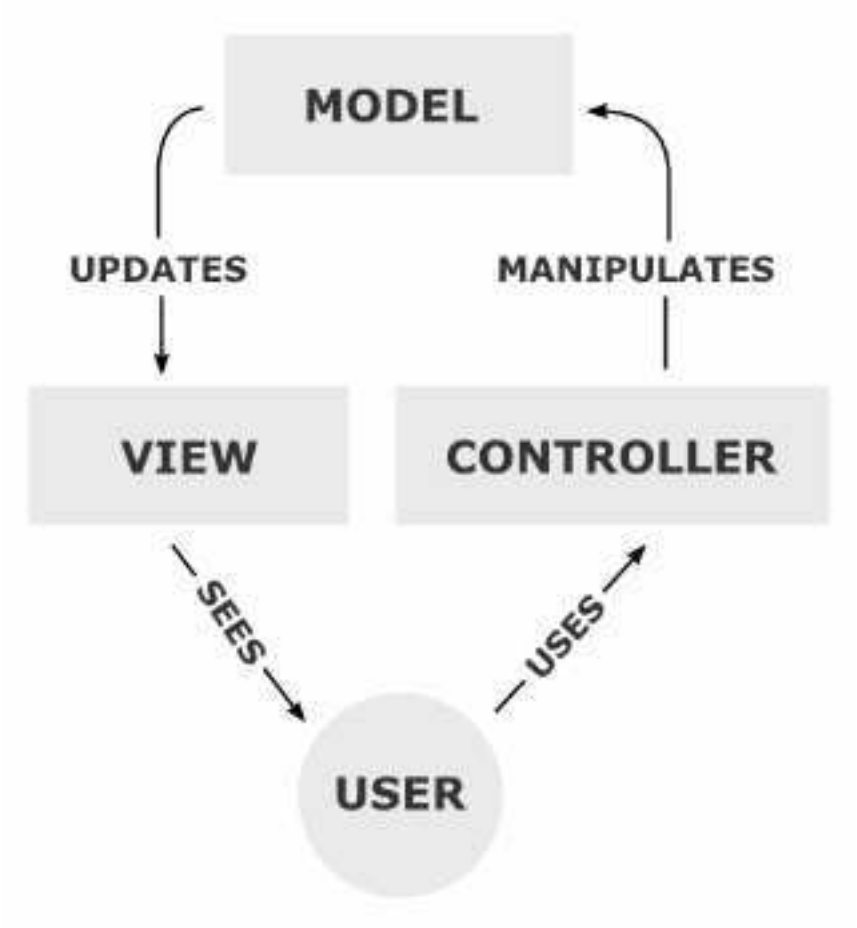


Imagen 2.1: Modelo Vista Controlador.

A continuación analizaremos brevemente en que modulo se utilizan las tecnologías de SherpaBeans antes comentadas:

#### Modelo

La representación lógica del modelo de datos se ha realizado mediante *Hibernate* ya que facilita el mapeado de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación. Esta relación se establece mediante la definición de las relaciones en el código de la aplicación, tiene 2 formás de mapear las clases y atributos, mediante archivos declarativos XML o mediante anotaciones en los beans de las entidades que permiten establecer las relaciones, en nuestro caso nos decantaremos por las anotaciones por ser más intuitivas y no requerir producir ficheros extra de declaraciones. Además *Hibernate* permite que la consulta y modificación de la base de datos sea mucho más dinámica y flexible.

#### Vista

En la capa de la vista o de interficie gráfica es donde, principalmente, entran 2 tecnologías que interactúan entre ellas para facilitar el desarrollo: Wicket y Web 2.0.

Wicket es una tecnología pionera en creación de interfícies gráficas similar a JSF y Tapestry[19], la programación de la interfaz se realiza mediante componentes y está basado en eventos, todo esto permite que se trabaje de forma más intuitiva sobre la capa de presentación, de como se quieren estructurar las paginas y los eventos que se deben lanzar cuando el usuario realiza una determinada acción. Wicket además facilita la creación y reutilización de componentes escritos tanto en Java como HTML de tal forma que se puede tener de forma intuitiva cualquiera de los componentes más comunes como botones, listas desplegables, etc. Además otra de las ventajas de utilizar Wicket es que la mayoría de sus componentes pueden utilizar AJAX ( Asyncronous JavaScript And XML)[20] por lo tanto la página web puede adoptar unos dinamismos muy interesantes. Además todas estas características se combinan con los estándares definidos por Web 2.0, que proporcionan mucha versatilidad y capacidad de mejora a esta capa.

#### Controlador

Al estar basado en Struts, SherpaBeans tiene similitudes a este en la implementación del *controlador*. Struts utiliza las acciones, los *servlets*[21] y ficheros de configuración[22] , en cambio SherpaBeans, si es cierto que también utiliza acciones y *servlets,* pero no utiliza un fichero de configuración, se vale de una clase java denominada *ProcessModel* que actúa a similitud de este fichero, donde se definen los diferentes estados del proceso así como las transiciones entre ellos, pudiendo utilizar, entre otros: estados iniciales, de vista, finales, decisión, de control, entre otros.

### 2.3. Seguridad

Como se debe dotar de seguridad a la aplicación en dos sentidos: autentificación de los usuarios y autorización para utilizar determinados recursos, SherpaBeans utiliza la tecnología JAAS (Java Authentication and Authorization Service)[23] respaldada por LDAP (Lightweight Directory Access Protocol)[24] . JAAS es un conjunto de librerías que permite a los desarrolladores de aplicaciones empresariales olvidarse de implementar código responsable de controlar permisos y accesos a diferentes páginas y/o funciones de la aplicación. El objetivo principal de JAAS es el de separar las tareas en la autentificación del usuario para que estos puedan ser gestionados de forma independiente, además este se centra en un nuevo marcador para poder identificar *quien está ejecutando* el código y validar que se tenga acceso al mismo ,esto es denominado como *autentificación*, y los permisos necesarios para realizar acciones concretas una vez autentificado, *autorización* [25]. Para ello utiliza roles, grupos y permisos pudiendo combinarlos a merced del programador.

Por su parte LDAP es un protocolo a nivel de aplicación el cual permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red, como una base de datos en la que se pueden hacer consultas. Así pues JAAS se sirve de LDAP para consultar las relaciones usuario-contraseña, usuario-rol, usuario-grupo, usuario-permisos, rol-grupo, rol-permisos, etc.

## 2.4. Estructura del proyecto

Todo el proyecto se encuentra dentro de un paquete llamado por convención interna: com.isencia.enterprise.changemanager. Para diferenciar las capas del modelo MVC se agrupan las diferentes clases en diferentes paquetes con un nombre para identificar la naturaleza de los mismos, además existen otras carpetas en el paquete principal donde se almacenan archivos, como por ejemplo: imágenes, el modelo de datos o *scripts* de inicialización y destrucción del modelo en la base de datos. A continuación se clasificarán todos estos elementos dependiendo en que componente del modelo MVC actúan, lo podemos ver en la imagen 2.2, así como una breve explicación de cada uno de ellos, esto también nos servirá como pequeña introducción a la metodología de desarrollo con SherpaBeans. En el punto 3.2 y 3.3 se entrará en más detalle sobre SherpaBeans ilusatrandolo con porciones de código para poder explicar parte de su funcionamiento.

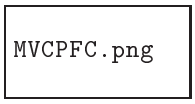


Imagen 2.2: Estructura del proyecto

**Elementos relacionados con el modelo:**

- Paquete **com.isenca.enterprise.changemanager.entities**: es donde se encuentran definidas las entidades que representan todo el modelo de datos, cada una de ellas corresponde a la definición de una clase, con sus atributos y métodos.
- Carpeta **doc/datamodel/schema**: esta carpeta contiene un archivo .dez generado con Dezigner for Databases, que define todo el modelo de datos de nuestro módulo, es decir la entidades existentes y sus relaciones.
- Carpeta **doc/datamodel/scripts**: aquí es donde se almacenan los scripts de creación, destrucción y alteración de nuestro modelo de datos en la base de datos. Los 2 primeros suelen ser generados por el Dezigner for Databases una vez hemos definido todo el modelo.

**Elementos relacionados con la vista:**

- Paquete **com.isenca.enterprise.changemanager.criteria:** aquí encontramos ficheros con *criteria querys* definidos, son clases que extienden de la clase Java *CriteriaQuery*, utilizadas por Hibernate para realizar consultas sobre la base de datos, el resultado de estas consultas son los datos a mostrar por la vista.
- Paquete **com.isenca.enterprise.changemanager.views:** este paquete es el más importante de los relacionados con la vista ya que es donde encontramos la gran mayoría los archivos que influyen con los aspectos visuales de las pantallas. Cada pantalla suele tener 2 tipos de archivos relacionados: el .java, clase de tipo *view*, que define el marco general de la vista, que tipo de estructura utilizaremos para mostrar los datos: en forma de tablas, *tab*, *dashboard*. Además ofrece la posibilidad de definir acciones: ocultar/mostrar elementos, ejecutar consultas auxiliares, inicializar objetos necesarios,... o añadir componentes web en

la página previamente referenciados en un archivo .html. Este archivo .html es dónde se puede dotar de más personalización a las paginas mediante este conocido lenguaje, así como crear componentes web (*wicket*), que podrán ser utilizados como un objeto en la clase .java en cuestión. Todas las pantallas de la aplicación tienen una, o más, clase .java asociada, pero no es obligatorio que tenga un .html.

- Paquete **com.isenca.enterprise.changemanager.settings:** finalmente, pero no menos importante, encontramos las clases de tipo *setting*, encargadas de relacionar las clase de tipo *view* y *criteria* con la entidad a la que hacen referencia, así como de definir diferentes opciones que afectan directamente a la pantalla mostrada: atributos que aparecen en las tablas, proporcionar filtros para realizar búsquedas *in situ*, así como poder definir acciones web que pueden depender de eventos generados por el usuario: ocultar campos, campos obligatorios, validaciones, etc. Para ello se utilizan diferentes funciones internas de SherpaBeans que añaden diferentes atributos a la clase de tipo *settings*, a modo de plantilla, que más tarde será interpretada por SherpaBeans para crear toda la estructura de la página en cuestión, algunos atributos que pueden ser añadidos son: *criteria* y *view* a utilizar, atributos de clase a mostrar, ancho de columnas, visibilidad de campos, comportamiento de campo a eventos de usuario (llamado *behavior*), validaciones, entre otros. Esto permite al programador no tener que centrarse tanto en el aspecto visual de la pagina con los quebraderos de cabeza que ello conlleva.

**Elementos relacionados con el controlador:**

- Paquete **com.isenca.enterprise.changemanager:** Aquí encontramos diferentes clase de configuración interna que necesita SherpaBeans para poder mostrar los elementos que añadamos al menú general, así como definición de constantes o excepciones, que serán mostradas en caso de error, de nuestra aplicación.

- Paquete **com.isenca.enterprise.changemanager.behaviors:** cuando se quiere definir algún comportamiento de algún componente web se suelen crear clase llamadas *behaviors* (comportamientos) que contendrán las acciones a realizar si el evento generado por el usuario se cumple. Estas clase son utilizadas por los *settings* para añadir la funcionalidad a los componentes requeridos, estos suelen ser campos o filtros.

- Paquete **com.isenca.enterprise.changemanager.flows:** En este paquete se almacenan las clase que se utilizan para redefinir o añadir funcionalidad a las transiciones de las páginas, así como añadir nuevas. Encontramos por defecto, en la parte superior de las tablas con los datos, estas transiciones, denominadas acciones: crear, editar, eliminar y ver. Las acciones nos suelen llevar al mismo estado inicial, pasando por los estados intermedios necesarios, después de realizar una de ellas, además suelen referirse a los registros que se muestran en la página en forma de tablas, ya sea en una tabla simple, dentro de un *tab* o de un *dashboard*. El otro tipo de transiciones, que nos suelen dirigir hacia otro estado diferente del inicial, también pueden ser redefinidas o añadidas desde estas clase, pero por convención interna y por una mejor organización solo se hará en caso que desde las clase especifica, almacenadas en el paquete **changemanager.workflow**, no fuera posible dotarlas de toda la funcionalidad requerida.

- Paquete **com.isenca.enterprise.changemanager.models:** Es donde se encuentra la clase con la definición del "ProcessModel", esta definición sirve para poder crear una nueva instancia de cualquier objeto al iniciar el flujo de estados al que esta asociado y así trabajar con él. En nuestro caso solo tenemos 1 objeto (Petición de cambio) así pues solo tenemos un ProcessModel.

- Paquete **com.isenca.enterprise.changemanager.tasks:** En este paquete encontramos clase que definen comportamientos auxiliares que no se encuentran ya definidos, la diferencia con los *behaviors*, es que los *tasks* no suelen definir comportamientos para componentes web, por ejemplo se suelen utilizar para autorizar el uso de un modulo, acción, transición dependiendo de si se cumplen los requisitos, o para realizar operaciones internas de nuestro flujo de acciones, tales como generar un documento, guardar/modificar/eliminar/cargar datos en base de datos que sea necesario para el buen funcionamiento de nuestro módulo, mostrar mensajes por pantalla, etc.

- Paquete **com.isenca.enterprise.changemanager.triggers:** Aquí encontraremos clase que definen nuevos "disparadores" diferentes a los que ya tenemos por defecto y que utilizarán los usuarios para ejecutar acciones sobre el estado actual o transiciones a nuevos estados.

- Paquete **com.isenca.enterprise.changemanager.validations:** Como su nombre indica es donde almacenaremos clase que definen validaciones sobre los datos introducidos por el usuario en nuestros formularios: campos obligatorios y/o campos con formato obligatorio.

- Paquete **com.isenca.enterprise.changemanager.valuetypes:** Contiene las clases que definen de diferentes tipos de datos auxiliares para facilitarnos la implementación de nuestro módulo. Normalmente serán clase de tipo *enum*.

- Paquete **com.isenca.enterprise.changemanager.workflow:** Aquí es donde se encuentran las clase con la definición de todo el flujo que tendrá nuestro módulo general, llamado *ProcessDefinition*, así como los estados que deberá transitar y son parte del flujo, si también tenemos que añadir transiciones que no existan lo haremos en este paquete. En nuestro caso deberemos definir 2 *ProcessDefinition*, uno por cada tipo de petición, además de todos los estados que los componen y algunas transiciones, estos estados y transiciones pueden reutilizarse en diferentes *ProcessDefinition*, como es nuestro caso.

- Carpeta /**META-INF:** En esta carpeta se encuentran ficheros de propiedades, de tipo *dictionary* que son utilizados por una clase interna, desarrollada por Isencia, para poder cambiar el idioma de los menús, etiquetas, *trigguers*, etc, del módulo. En nuestro caso tendremos el catalán, el castellano y el inglés.

## 2.5. Entorno de desarrollo

Las herramientas utilizadas son comunes con todos los empleados de la empresa, en nuestro proyecto en cuestión se han utilizado las herramientas que ofrecían la mejor relación estabilidad/facilidad de uso, aparte de los servidores que se necesitan para que el desarrollador realice pruebas en el entorno local (se explica más adelante), contamos con: Eclipse Plataform para Java, Dezign for Databases y MySQL.

#### Eclipse Plataform

Eclipse Platform en su versión Java es un entorno de trabajo (IDE) para los programadores Java que proporciona herramientas de desarrollo de aplicaciones. Además se le pueden incorporar plugins desarrollados por la comunidad que facilitan enormemente el trabajo. En nuestro caso no ha sido necesario la instalación de ningún plugin externo especifico ya que con las herramientas mencionadas anteriormente y los plugins por defecto de eclipse (caso del repositorio CVS) ya podemos desarrollar nuestra aplicación.

#### Dezign for Databases

Es una herramienta de diseño y modelado de bases de datos muy intuitiva para los desarrolladores que es de gran ayuda para modelar, crear y mantener nuestra base de datos. Este software utiliza diagramás de entidad-relación para diseñar gráficamente la base de datos y automáticamente generar los ficheros SQL de creación y destrucción del modelo de datos.

#### MySQL

Es un sistema de gestión de bases de datos relacional, multihilo y multiusuario, es software libre, pero si una empresa quiere incorporarlo a productos privativos deben comprar a la empresa una licencia especifica. En nuestro caso solo lo utilizaremos para ejecutar los scripts generados por el Dezign, además de cualquier tipo de modificación o consulta que debamos hacer en la base de datos.

## 2.6. Metodología de desarrollo

Durante el desarrollo del modulo, se han hecho diferentes pruebas de cada submódulo que lo compone para garantizar el funcionamiento individual de cada uno de ellos. En general se suele seguir el siguiente proceso circular:

1. El coordinador del proyecto, usando Compás, genera los documentos correspondientes a los casos de uso de cada submódulo: especificaciones a tener en cuenta, acciones que se realizan, campos que influyen, comportamiento deseado, transiciones a otros submódulo o estados.

2. El desarrollador plantea y razona la solución para cumplir con los casos de uso y sus indicaciones.

3. Se crean las definiciones del submódulo siguiendo las especificaciones del caso de uso asociado:crear la entidad en base de datos y añadirla, crear las clase asociadas al submódulo ya sean de *flow*, *settings*, *views*, etc. Se implementaran todas las acciones y transiciones que tengan como estado final el mismo estado en el que nos encontramos.

4. Una vez tenemos una versión que pase el proceso de compilación se debe asegurar que el modulo cumple con todas las acciones requeridas en el use case mediante la realización de un juego de pruebas. Este juego de pruebas se podría crear con una herramienta que nos facilite la tarea como por ejemplo *Junit*, pero en nuestro caso nos serviremos de las acciones previstas en el caso de uso para realizar dichas pruebas manualmente. Además de tener que pasar otras pruebas de performancia y casos no previstos realizado manualmente por el mismo desarrollador.

5. Se realizan todas las pruebas obtenidas en el punto 4, si el resultado es óptimo, se conecta este submódulo con el resto que hayan pasado por todos los puntos, incluido el 6. En caso contrario se busca el error, se solventa y se retorna al punto 4.

6. En este punto es donde se debe probar que todos los submódulo se comunican correctamente, los que requieran comunicarse, y que la adición del nuevo no perjudique la performancia del resto, ni de todo el conjunto. Para ello contamos con los use cases, estos también definen la interacción de los submódulo. Así pues se deberán pasar todos los requerimientos hasta el punto donde tengamos desarrollado el módulo. Si los resultados son satisfactorios se pasará a desarrollar otro submódulo. En caso de error deberemos encontrar que lo produce (submódulo, conexión entre submódulo, etc.), corregirlo y volver al punto 4.

En caso de que haya un cambio en los requerimientos, se debe volver a empezar por el primer punto revisando todas las tareas. Cada vez que se crea un submódulo nuevo o se modifica uno se deben realizar todas las pruebas de todos los submódulos para asegurar que el cambio no ha supuesto ningún problema para el funcionamiento del módulo y este se mantiene estable

# 3. Implementación y despliegue

A continuación se comentarán detalles de la implementación del módulo además de algunos detalles relacionados con eSengo®. Lo dividiremos en 4 apartados: el modelo de datos, breve explicación de eSengo®, los submódulos, unos funcionan conjuntamente con el Gestor de Cambios y otros son utilizados para generar datos para el gestor u otros módulos y el módulo Gestor de Cambios, que tiene sus estados y transiciones de unos a otros, así como sus acciones internas.

## 3.1. Modelo de datos

El modelo de datos es el punto de partida del módulo. Se debe diseñar un modelo consistente y eficaz que permita almacenar la información del módulo. En este punto se diferencian 2 puntos: el diseño de la base de datos y como se ha integrado este diseño utilizando Hibernate.

### 3.1.1. Diseño

En la imagen 3.1 se puede ver el diseño del modelo de datos. A continuación se detallan las entidades así como una breve explicación de su utilidad.

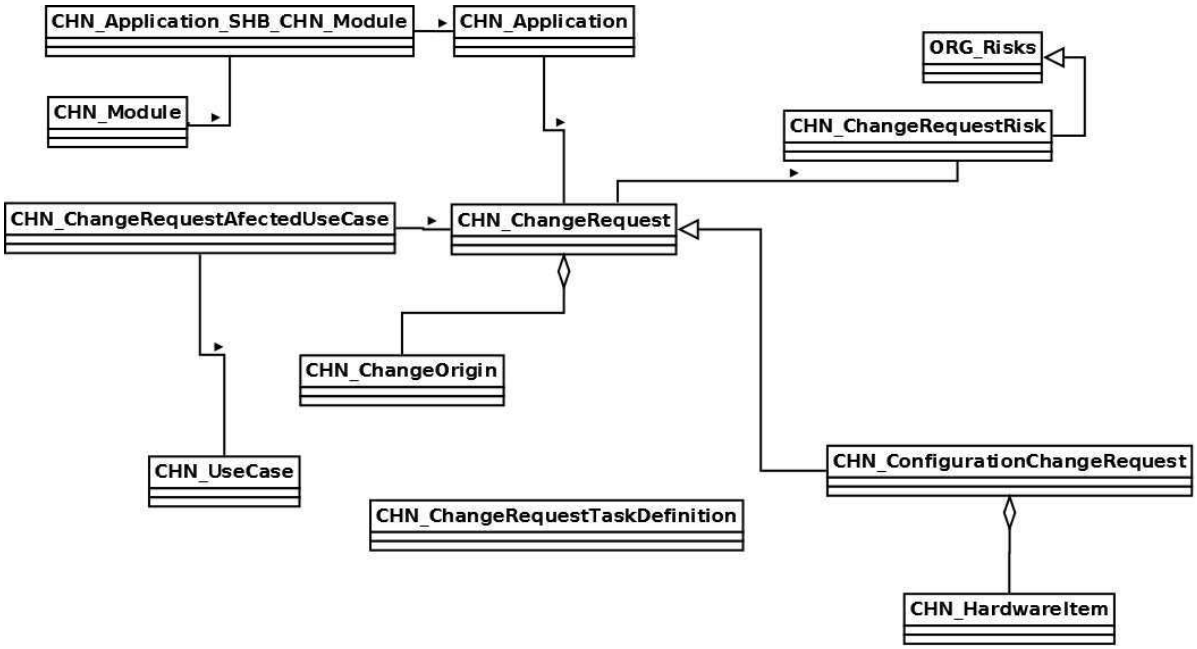


Imagen 3.1: Diseño modelo de datos

- **CHN_ChangeRequestTaskDefinition:** Esta es una entidad de configuración interna que sirve para almacenar los tipos de petición de cambio que tenemos así como asociarlos a una id para poder tener flujos independientes por cada tipo. En nuestro caso solo tendremos 2 tipos, software y hardware
- **CHN_Application:** Almacena los datos correspondientes a la aplicación a la cual va ligada una petición de cambio de tipo software, en ella encontramos atributos como el nombre y el código de la aplicación.
- **CHN_Module:** Encargada de contener los datos correspondiente con los módulos que pueden componer una aplicación, sus atributos son, como en el caso de la aplicación, nombre y código.
- **CHN_Application_SHB_CHN_Module:** Tabla intermedia, definida internamente, que se encarga de guardar las relaciones entre aplicaciones y módulos. Una aplicación puede tener más de un módulo, pero los módulos son únicos.
- **CHN_UseCase:** Entidad que almacenará los documentos conocidos como UseCase, además también contiene la ID de la aplicación a la que va asociado. También sería útil para almacenar cualquier tipo de documento que queramos asociar a una aplicación en concreto, pero no es nuestro caso.
- **CHN_ChangeRequestAfectedUseCase:** Tabla intermedia, definida internamente, que contiene las relaciones entre las peticiones de cambio y los UseCase, dependiendo de la aplicación que afecte el cambio esta tendrá unos UseCase asociados u otros.
- **CHN_ChangeOrigin:** Contiene los datos correspondiente a los originadores del cambio, ya pueden ser internos (mismo personal de la empresa en cuestión) o externos (clientes), como atributos tiene el código, el nombre y si el registro pertenece a un cliente o no.
- **ORG_Risks:** Encargada de almacenar información sobre los riesgos como por ejemplo: el tipos, los diferentes niveles de impacto, comentarios sobre los niveles de impacto, comentarios generales, mitigación, solución...

- **CHN_ChangeRequestRisk:** Es la encargada de guardar los datos sobre los diferentes riesgos que pueden producir los cambios requeridos si se aplican, hereda de ORG_Risk. Ya pueden ser riesgos nuevos derivados de la petición o ya almacenados en la base de datos.
- **CHN_HardwareItem:** Clase que hereda de una clase interna denominada *Item*, encargada de almacenar información sobre los elementos hardware que tienen relación con una petición de cambio de tipo hardware, entre otros algunos de sus atributos son: tipo de item, numero de serie, cantidad...
- **CHN_ChangeRequest:** Contiene toda la información relacionada a la petición de cambio como el estado en el que se encuentra, el originador, comentarios, el tipo de petición, etc.
- **CHN_ConfigurationChangeRequest:** Hereda de la clase CHN_ChangeRequest, a diferencia de esta tiene campos específicos para almacenar información de una petición de tipo hardware, ya que cuenta con un objeto de tipo CHN_HardwareItem entre ellos.

### 3.1.2. Integración

La integración del modelo de datos con nuestro módulo se lleva a cabo mediante Hibernate. No se hará una explicación extensa ya que estamos hablando de convenciones internas de la empresa, pero si un breve resumen de algunos puntos a tener en cuenta, por ejemplo:

- Por cada entidad en la base de datos existirá una clase, excepto en el caso que la entidad de base de datos de una entidad auxiliar relacional de 2 entidades diferentes, en este caso se definirá un atributo en una de las 2 clases, normalmente en la que tiene más peso, para posteriormente mapearlo con anotaciones propias de hibernate y así tener la referencia a esa tabla intermedia desde la clase que deba utilizar sus registros. Según nos dicta Hibernate, cada clase tendrá sus atributos miembros como privados y con sus correspondientes funciones de acceso (*getters* y *setters*).
- "Mappings" de Hibernate. En nuestro caso mapearemos la clase mediante anotaciones en la misma clase para que hibernate sepa a que tabla corresponde, así mismo deberemos utilizar diferentes anotaciones de Hibernate para mapear los atributos, dependiendo del atributo será necesario incorporar más de una anotación, por ejemplo para definir el formato en el que se guardará la información.

## 3.2. Breve introducción a elementos de eSengo®

En este apartado veremos una pequeña parte de la funcionalidad que puede llegar a ofrecer el ERP. Nos centraremos en explicar, con ayuda de imágenes, algunas de las acciones y elementos que ya incorpora eSengo® y que influyen en nuestros submódulos y módulos, así como su funcionamiento.

**Campos y botones por defecto en la presentación de una tabla cualquiera con datos:**

1. El campo y el botón de buscar: realiza una búsqueda en la tabla de datos por la cadena de caracteres que se escriba tanto en el campo como en los filtros que estén asociados a la tabla en ese momento, en nuestro caso esos filtros serían *Código* y *Nombre*. Cualquiera de estos elementos se puede dejar en blanco al realizar una búsqueda a no ser que sean requeridos, en ese caso tendrían otro color o mostrarían un mensaje.

2. Desplegable *Filtros*: Como podemos ver en la imagen 3.3 el desplegable consta de 4 opciones, la opción seleccionada por defecto es *Filtros,* que nos muestra encima de la tabla de datos los filtros que hayamos seleccionado mediante la opción de configuración que tiene el icono de llave inglesa. La opción *Datos* abre una ventana emergente que permite seleccionar que columnas queremos que aparezcan, estas deben ser definidas previamente en los ficheros de *settings,* también permite aplicar otras opciones sobre las columnas, como por ejemplo que tipo de orden tendrán los datos que contienen. La opción más abajo, *Vista,* nos permite agregar a la parte inferior de la tabla unas pequeñas estadísticas sobre las columnas de la tabla, por ejemplo cual es el máximo o cantidad de filas. La opción superior, *Básico*, nos permite ocultar todos los filtros y hacer la búsqueda solo con el campo y el botón de buscar (1).

3. Botón *Por Defecto*: Permite guardar en base de datos la configuración creada a partir del desplegable *Filtros,* además de poder cambiar la actual por otra previamente almacenada.

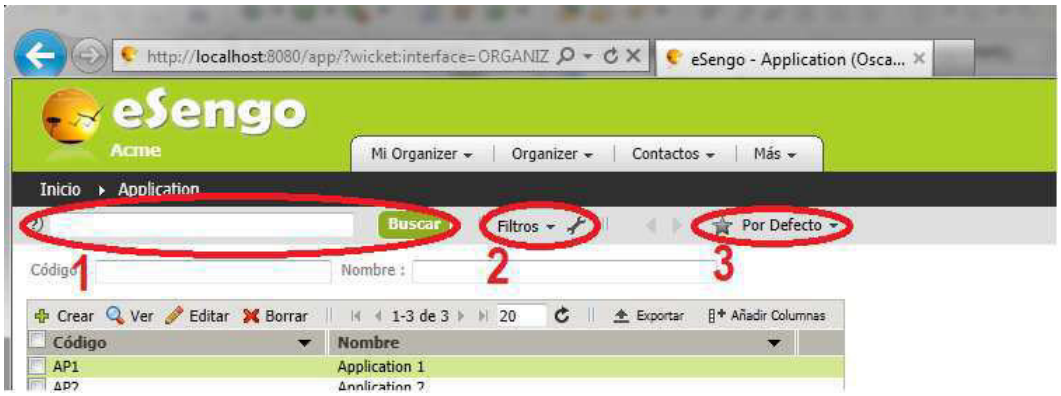


Imagen 3.2: Pantalla del submódulo Aplicación


Imagen 3.3: Menú desplegable *Filtros*

**Acciones por defecto en toda tabla:**

1. En este campo seleccionamos el número de filas que queremos mostrar en la tabla y con el botón contiguo refrescamos la tabla.

2. Esta acción nos permite exportar el contenido de la tabla a una hoja de cálculo, PDF o fichero XML.

3. Nos permite añadir o quitar las columnas que se hayan definido como seleccionables en la clase de tipo *settings* correspondiente a la tabla.


Imagen 3.4: Barra de acciones de una tabla

**Características de los usuarios, roles y permisos en eSengo®:**

- Para gestionar los roles, usuarios y los permisos de acceso a los módulos y opciones se utilizará el usuario Administrador, configurado por Isencia al adquirir eSengo®, el Administrador, también puede proporcionar sus mismos permisos a cualquier usuario.
- El Administrador (o los usuarios con sus permisos) sera el encargado de crear/eliminar los roles y los usuarios.
- Un usuario puede tener más de un rol, y un rol puede ser asignado a más de un usuario.
- Los roles suelen tener permisos diferentes a los de los usuarios que contiene.
- Los roles pueden ser utilizados por el programado para dar o quitar visibilidad a cualquier componente del código, al igual que los permisos.
- Los permisos deben ser creados desde el código para que eSengo® pueda mostrarlos al Administrador, por lo tanto el programador es responsable de agregar los permisos para todos los componentes que lo necesiten. En la imagen 3.5 Se muestra la porción de código necesaria para definir los permisos:

```
1  public class ChangeManagerFeatures {
2      public static final Feature CHANGE_REQUEST = new Feature(ChangeRequest.KEY, Action.CRUD_ACTIONS);
3      public static final Feature CHANGE_ORIGIN = new Feature(ChangeOrigin.KEY, Action.CRUD_ACTIONS);
4      public static final Feature APPLICATION = new Feature(Application.KEY, Action.CRUD_ACTIONS);
5      public static final Feature MODULE = new Feature(Module.KEY, Action.CRUD_ACTIONS);
6      public static final Feature USECASE = new Feature(UseCase.KEY, Action.CRUD_ACTIONS);
7      public static final Feature CHANGE_REQUEST_TASK_DEFINITIONS = new Feature(ChangeRequestTaskDefinition.KEY,Acti
8
9      public static final Feature[] FEATURES = {
10         CHANGE_REQUEST,
11         CHANGE_ORIGIN,
12         APPLICATION,
13         MODULE,
14         USECASE,
15         CHANGE_REQUEST_TASK_DEFINITIONS
16     };
17 }
18
```

Imagen 3.5: Código de la clase ChangeManagerFeatures

Como podemos observar en el código para definir un permiso utilizamos el objeto *Feature*. Su constructor requiere la id correspondiente al submódulo/modulo/entidad/objeto a la que se le quiere hacer referencia, así como las acciones que incluirá el permiso. En nuestro caso serán todo acciones básicas (ver, crear, eliminar, editar), de ahí que utilicemos la constante del objeto *Action*, *Action.CRUD_ACTIONS*. Finalmente se añade el nombre por el que hemos definido el permiso en un *Array* de tipo Feature, este *Array* será utilizado por SherpaBeans para mostrarnos los permisos por la interfaz gráfica así como de darles funcionalidad.

## 3.3. Submódulos

Estos suelen ser submódulos auxiliares que proporcionarán datos que el módulo general necesita para poder tener la máxima información, la mayoría de ellos son independientes entre si. Al ser independientes del general podrán ser utilizados por otros módulos. En las siguientes secciones se detalla su funcionalidad así como una muestra de como son sus pantallas, además algunos de ellos nos servirán para hacer una pequeña introducción a la plataforma SherpaBeans 6.0

### 3.3.1. Application y Module

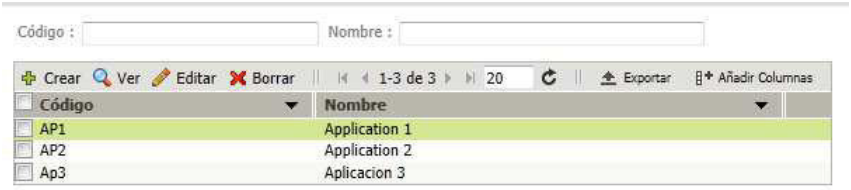


Imagen 3.6: Consulta de aplicaciones

En la Imagen 3.6 se observa la pantalla principal de este submódulo en ella se puede ver una lista de las aplicaciones según el criterio de filtrado. Estos criterios son los siguientes:

- Código: Busca según el código que corresponde a la aplicación, no hace falta que sea el código entero, se pueden hacer búsquedas parciales.
- Nombre: Igual que el código.

También encontramos las acciones de crear, ver, editar y borrar, normalmente estas opciones actuarán de la misma forma en todas las tablas, lo único que cambiara serán las pantallas asociadas a cada acción. Las acciones de eliminar y editar pueden aplicarse a 1 o varios registros mediante la selección múltiple de casillas:

- Crear un nuevo registro: Como podemos ver en la imagen 3.7 al pulsar en la acción de crear se nos abre este *popup*, de tipo *dashboard*. Una de las diferencias entre un *dashboard* y una *view* es que los *dashboard* son mucho más configurables, por ejemplo, puedes cambiarlos de tamaño, esconder/mostrar tablas y acciones, etc. En cambio las tablas son fijas y sin ningún tipo de personalización, aparte de las acciones comentadas que nos ofrece eSengo®. Podemos observar que cuenta con 2 campos obligatorios que se marcan en color amarillo para identificarlos como tales, además en la parte derecha observamos la tabla Módulo, esta es otra de las características de los *dashboards*, poder añadir tablas de un submódulo B en los estados de un submódulo A, en este caso nos sirve para poder crear los módulos que tienen relación con la aplicación que creemos.

También podríamos añadir otros elementos como pestañas o un *dashboard* diferente dentro del *dashboard* inicial. Al pulsar el botón de añadir este nos abre otro *popup* con los campos, Código y Nombre, a rellenar que también serán obligatorios, una vez finalizado se añade el nuevo módulo a la aplicación. Cada fila de la tabla módulo cuenta con las opciones, de editar y borrar (la cruz roja). *Editar* nos abre un *popup* igual que crear, con los datos rellenados, donde podremos editar los datos del módulo. La acción de borrar nos abre otro *popup* donde se nos pide confirmación para borrar el elemento deseado. Una aplicación puede, o no, tener módulos, así pues se pueden añadir en el momento de creación o más tarde con la acción de edición, por lo tanto la relación entre la aplicación y los módulos no se guardará hasta que no le demos al botón de *Crear*. Cuando terminamos las acciones volvemos a la pantalla inicial con la tabla de datos, esto sucederá con la mayoría de submódulos y acciones.

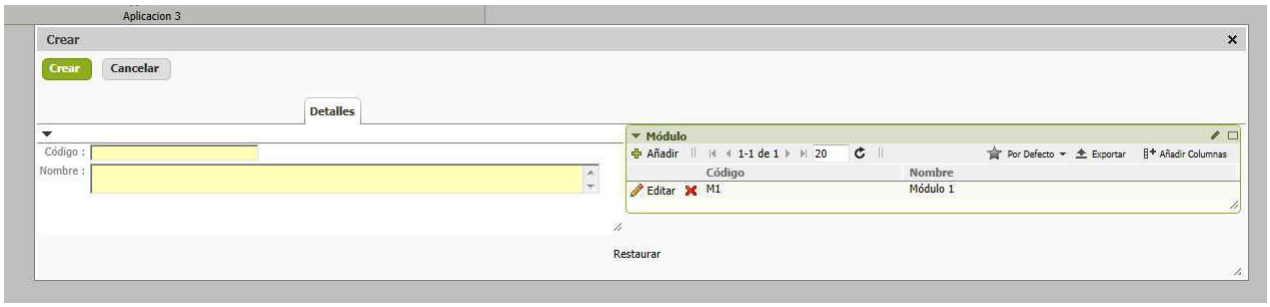


Imagen 3.7: Submódulo aplicación, estado crear.

- Ver un registro: Al utilizar la acción *Ver,* se abre un *popup* con la información del registro seleccionado, como podemos observar en la imagen 3.8. Esta información no es editable, pero si utilizamos la acción del botón *Editar* nos lleva a un estado nuevo donde podremos editar el registro, esta acción la encontraremos por defecto en todos los estados de este tipo.


Imagen 3.8: Submódulo aplicación, estado ver.

- Editar un registro: Cuando utilizamos la acción de *Editar* nos aparece un *popup* muy similar al de la acción crear, salvo que en este caso los campos están rellenados con los datos que corresponden al registro que queramos editar, tanto campos de texto como los módulos incluidos. Podemos modificar la información de los campos así como agregar/eliminar módulos. Si son varios los registros a editar, una vez editemos el primero nos aparecerá la misma ventana pero con los datos del segundo así sucesivamente hasta haberlos editados todos. Si en uno de ellos cancelamos la edición los cambios se aplicarán a todos los registros modificados anteriores.

- Eliminar un registro: Utilizando la acción eliminar nos aparecerá un *popup* como el de la imagen 3.9, si seleccionamos múltiples entradas el *popup* sería como la imagen 3.10. Al confirmar la acción se elimina/n la/s entrada/s de la base de datos.

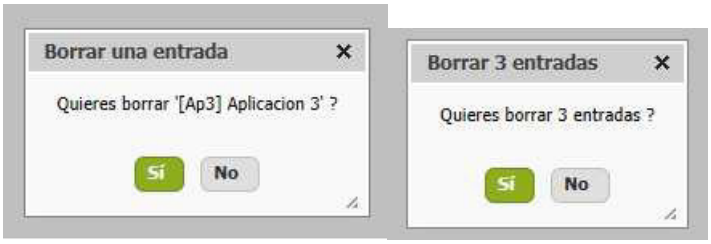


Imagen 3.9: Borrar 1 entrada  Imagen 3.10: Borrar varias entradas

Para realizar este módulo se han debido utilizar, clases de tipo *flow y settings*. A continuación pondremos algunos ejemplos para poder explicar mejor como son este tipo de clases y alguna de sus características:

- Ejemplo clase tipo *flow*: En la imagen 3.11 vemos el código del *flow* del submódulo *Application*. La clase *ApplicationFlow* extiende de *ExtendedCrudFlow* esta nos proporciona en nuestra tabla las acciones de crear,ver,eliminar,editar por defecto que tiene el sistema. Las lineas 24-27 nos sirven para indicar que todos los estados (crear, editar, ver, eliminar) se abrirán en *popup*. Si queremos que cuando transitemos entre estos estados se realice alguna acción especifica, podemos añadir una nueva tarea a la transición entre estados, en nuestro caso vemos en la linea 28 como añadimos una tarea de salida, *addExitTask*, a la confirmación de la creación de un registro, *getConfirmCreateTransition()*, que se ejecutará la primera de todas, *addExitTask(**0**,...* , ya que algunas transiciones ya tienen tareas por defecto y puede que nos interese que la nuestra esté en una posición determinada. Nuestra tarea, *new Task(){...,* se encargará de guardar la relación de todos los módulos añadidos a la aplicación. Para ello utilizamos la clase *EntityManager* que nos permite almacenar de forma temporal las relaciones, si las hay, a la espera de que una vez acabada la transición se guarden los datos en base de datos. También podemos observar que a la transición de cancelar del estado crear se le ha añadido otra tarea de salida, esta sirve para eliminar los módulos añadidos y que estos no se creen en base de datos, ya que si no lo hiciéramos podríamos encontrarnos que al cancelar la creación de una aplicación se creasen registros fantasma de estos módulos, en base de datos, que no podrían ser utilizados nunca ya que no tendrían relación con ninguna aplicación.

```
16  public class ApplicationFlow extends ExtendedCrudFlow{
17
18      private static final long serialVersionUID = 1L;
19      public static final String ID = "ApplicationFlow";
20
21
22      public ApplicationFlow(IFlowDefinition parent) {
23          super(parent,ID,Application.KEY);
24          getCreateState().setScope(Scope.DIALOG);
25          getEditState().setScope(Scope.DIALOG);
26          getDeleteState().setScope(Scope.DIALOG);
27          getViewState().setScope(Scope.DIALOG);
28          getConfirmCreateTransition().addExitTask(0,new Task(){
29              private static final long serialVersionUID = 1L;
30
31              public boolean execute(IActionEvent event) {
32                  EntityManager em = EntityManagerPool.getEntityManager();
33                  Application application = ModelUtils.findBean(event);
34                  if(application.getModule()!=null){
35                      for(Module module : application.getModule()){
36                          module.setApplication(application);
37                      }
38                      em.merge(application);
39                  }
40
41                  return true;
42
43              }});
44
45          getCancelCreateTransition().addExitTask(0, new Task(){
46              private static final long serialVersionUID = 1L;
47
48              public boolean execute(IActionEvent event) {
49                  Application application = ModelUtils.findBean(event);
50                  if(application.getModule()!=null){
51                      CommandExecuter.removeAll(application.getModule());
52                  }
53                  return true;
54              }});
55      }
56  }
```

Imagen 3.11: ApplicationFlow.java

- Ejemplo clases tipo *settings*: En la imagen 3.12 podemos ver el código de la clase de tipo *settings* que hace referencia al submódulo *Application*. Como se ha comentado anteriormente este tipo de clases nos proporcionan objetos con muchos métodos por defecto para poder configurar las pantallas de nuestros submódulos y del módulo general. Los objetos y métodos por defecto vienen heredados de la clase *ISettingsInitializer*. Como podemos observar normalmente estructuramos los *settings* por diferentes apartados para facilitar su comprensión, iremos comentándolos brevemente uno a uno, así como los métodos que se definen en ellos. La primera linea que nos llama la atención es la de *Authorization Settings,* con el objeto *AuthorizationSettings* podemos configurar varios aspectos sobre permisos y tipos de autorización, pero en nuestro caso solo lo utilizaremos para darle una id al submódulo, todos los submódulos y módulos deben tener una id en su fichero de *settings* para poder ser identificados y relacionados por SherpaBeans. La siguiente sección, *Tab Settings*, hace referencia a las opciones para las pestañas de nuestra vista que utilizaremos para mostrar tablas, se podrían utilizar otras organizaciones diferentes a las pestañas, pero en nuestro caso siempre utilizaremos este formato, con el método *setTabs(<tabs>)* definimos cuantos *tabs* (pestañas) tendremos en nuestra vista, como podemos ver solo tendremos 1 con los campos que definamos en el fichero de *settings* (de ahí la constante *MASTER_FIELDS*). En la siguiente linea encontramos el método *setContent*(<content>,<tab>) nos servirá para indicarle que clase de vista tendrá este *tab* ya puede ser una *view*, otro *tab* o un *dashboard* como en el ejemplo. Seguidamente viene la configuración de los filtros (*Filter Settings*) como podemos observar utilizamos el objeto *FilterSettings* para añadir los filtros que aparecerán en pantalla por defecto con el método *setDefaults(<properties>)*. Las *properties* son los atributos de las diferentes entidades definidas. También utilizamos el método *setColumns(<int>)* para definir en cuantas columnas se distribuirán los filtros, en nuestro caso, si no la utilizamos, tendríamos los 2 filtros uno debajo del otro, con este objeto también podríamos definir que filtros podrán ser añadidos desde las opciones de *Filters* ya comentada en el punto 3.2. El siguiente punto a tener en cuenta son los datos que aparecerán en la tabla, para ello haremos uso del objeto *DataSettings*, con la función *setDefaults(<properties>)* podemos indicar que columnas queremos que aparezcan, también podríamos definir que columnas podrán ser añadidas desde la opción *Añadir columnas*, también comentada en el punto 3.2. Seguidamente pasaremos a definir los campos que aparecerán en el formulario de creación con el objeto *FormSettings* y su método *setFields(<properties>),* con este objeto también podremos definir la posición de los mismos, como con los filtros, entre otras opciones. Una vez tenemos definidos los campos que aparecerán pasamos a editar las propiedades de estos, lo haremos con el objeto *FieldSettings* que consta de muchos métodos por ejemplo: añadir *behaviors* a los campos, hacerlos visibles u ocultarlos, editables o no, etc. En nuestro caso utilizaremos la función *setRequired(<properties>),* para definir que campos serán obligatorios. Finalmente tenemos el objeto *ViewSettings*, que nos servirá para definir que componente tendremos en nuestra vista, en el ejemplo estamos añadiendo la funcionalidad de un *tab* a nuestra vista de crear, esto normalmente se hace cuando hemos definido el contenido del algún *tab* con un *dashboard*, ya que si no se hiciera no podríamos acabar de ver correctamente todos los elementos que hayamos añadido a nuestro *dashboard*.

```java
public class ApplicationSettings implements ISettingsInitializer{

    private static final long serialVersionUID = 1L;

    public void init(ISettings settings) {
        //Authorization Settings
        AuthorizationSettings.setAuthorizationId("Application");

        //Tab Settings
        TabSettings.setTabs(TabSettings.MASTER_FIELDS);
        TabSettings.setContent(ApplicationDashBoardView.class,TabSettings.MASTER_FIELDS);

        //Filter Settings
        FilterSettings.setDefaults("code","name");
        FilterSettings.setColumns(2);

        //Data Settings
        DataSettings.setDefaults(
                "code",
                "name"
        );

        // Form Settings
        FormSettings.setFields(
                "code",
                "name"
            );

        //Field Settings
        FieldSettings.setRequired(true, "code","name");

        // View Settings
        ViewSettings.setBeanView(new ModeSettingValue(EnterpriseTabView.class, Mode.CREATE));
    }

}
```

Imagen 3.12: ApplicationSettings.java

### 3.3.2. ChangeOrigin


Imagen 3.13: Pantalla principal del submódulo ChangeOrigin

En la imagen 3.13 vemos el estado inicial del submódulo *ChangeOrigin* (origen del cambio), vemos una tabla con los registros correspondientes a los originadores del cambio según los criterios de los filtros, estos filtros son:

- Código: Busca según el código que corresponde al registro, no hace falta que sea el código entero, se pueden hacer búsquedas parciales.
- Nombre: Igual que el código.
- Proviene de un cliente?: En este caso nos encontramos con un *checkbox* que solo admite 2 opciones, igual que el atributo que le corresponde en la entidad, este atributo es de tipo boleano. SherpaBeans asigna por defecto el componente *checkbox* a este tipo de campos, si quisiéramos otro componente web en el campo, deberíamos cambiarlo utilizando la clase de *settings* asociada, más adelante se expone un pequeño ejemplo de como añadir un componente a un campo.

Las acciones son análogas al submódulo anterior ya que este esta construido con los mismos fundamentos:

- Crear un nuevo registro: Como podemos ver en la imagen 3.14 al pulsar en la acción de crear se nos abre este *popup*, de tipo *dashboard*. Este estado tiene 3 campos, 2 de texto obligatorios, Código y Nombre, y uno de *checkbox*, Proviene de un cliente?, este último sirve para identificar si el originador es un cliente externo o un empleado de la empresa. Cuando terminamos las acciones volvemos a la pantalla inicial, guardando el registro en memoria.

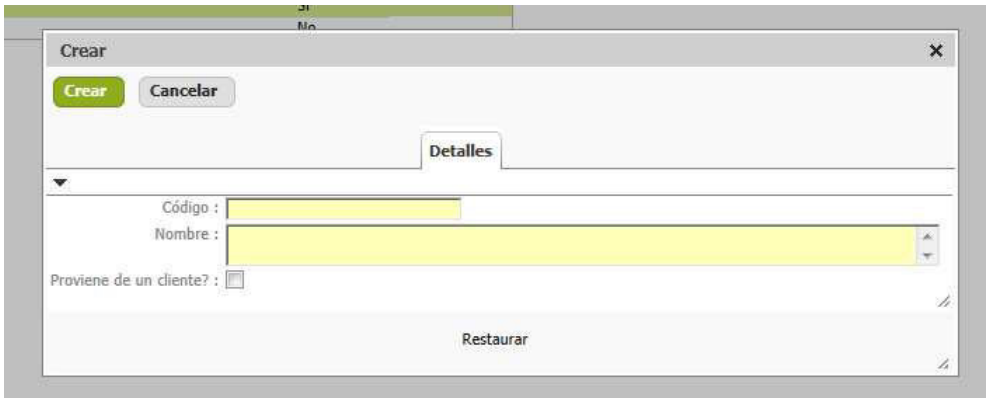


Imagen 3.14: Submódulo ChangeOrigin, estado crear.

- Ver un registro: Al utilizar la acción *Ver*, se abre un *popup* con la información del registro seleccionado, con las mismás características que en el submódulo anterior, como podemos observar en la imagen 3.15.

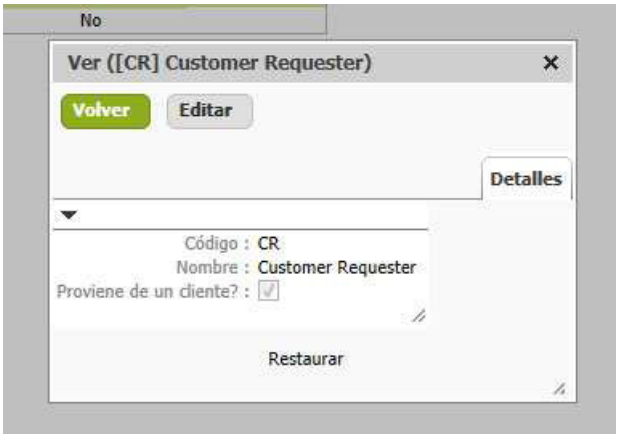


Imagen 3.15: Submódulo ChangeOrigin, estado ver.

- Editar y eliminar un registro: Análogo a las acciones del submódulo descrito en el punto 3.3.1.

### 3.3.3. UseCase

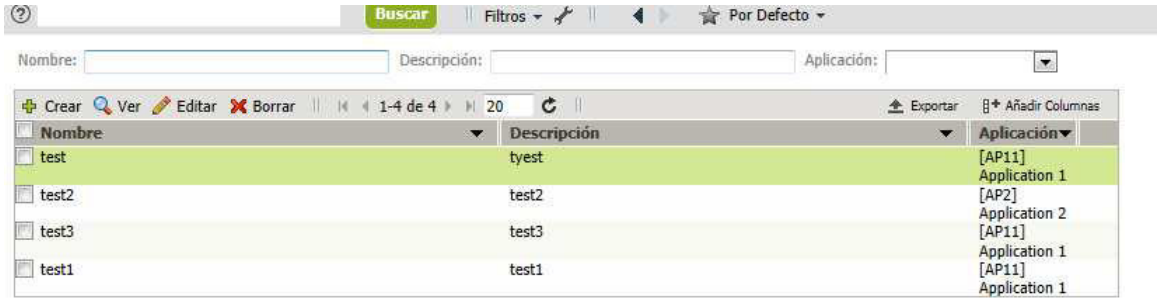


Imagen 3.16: Submódulo UseCase, estado inicial.

En la imagen 3.16 vemos el estado inicial del submódulo UseCase (Casos de uso), vemos una tabla con los registros correspondientes a los UseCase registrados según los criterios de los filtros, estos filtros son:

- Nombre: Busca según el nombre que corresponde al registro, no hace falta que sea el nombre entero, se pueden hacer búsquedas parciales.
- Descripción: Igual que el nombre, puede buscar cualquier palabra o frase que aparezca en la descripción, se mostraran los registros que contengan esa palabra o frase en su atributo descripción.
- Aplicación: En este caso nos encontramos con un desplegable que admite tantas opciones como aplicaciones hayamos creado, esto se configura con el archivo de *settings* asociado al módulo y SherpaBeans te crea la relación entre el combo y los datos que haya en base de datos. Como podemos ver en la imagen 3.17, se le debe añadir un componente al campo que queramos que tenga ese formato utilizando el objeto *FieldSettings* y su método *setComponent(<component>,<properties>)*. En nuestro caso añadimos el componente desplegable (*combo*) al objeto-atributo application, de la clase *UseCase*, de tipo *Application*.

El inconveniente es que dicho campo siempre tendrá ese formato, en todos los estados. Si queremos que no sea así deberíamos sobrescribir el método, desde los mismos *settings*, y con un condicional retornar el componente que queramos que tenga en cada estado (*edit, view, create*, etc...).

```
//Field Settings

FieldSettings.setComponent(ComboField.class,"application");
```

Imagen 3.17: Porción de código de la clase UseCaseSettings.java.

Las acciones básicas son las mismás que los submódulos anteriores:

- Crear un nuevo registro: Como podemos ver en la imagen 3.18 al pulsar en la acción de crear se nos abre este *popup*, de tipo *dashboard*. Este estado tiene 3 campos obligatorios, 2 de texto, Nombre y Descripción, que como podemos observar tienen tamaños diferentes. Esto se debe a que en la definición de la entidad se les ha añadido una longitud máxima, mediante una anotación, esto se hace con todos los campos de tipo *String*, vemos un ejemplo en la imagen 3.19. Mientras más longitud máxima más lineas tendrá el campo de texto, de todos modos también se puede modificar el número de lineas del campo con los *settings* asociados al submódulo. El 3er campo es el de Aplicación, como se ha comentado previamente es de tipo desplegable y tendrá como opciones todas las aplicaciones creadas con el submódulo *Application*. Finalmente nos llama la atención que hay una pestaña de nombre *Adjuntos/Documentos,* como se ha comentado previamente en un *dashboard* también se pueden añadir pestañas para mostrar los componentes que se deseen, a continuación hablaremos sobre esta pestaña y mostraremos su contenido. Cuando terminamos de crear el registro volvemos a la pantalla inicial, guardando el registro en memoria.

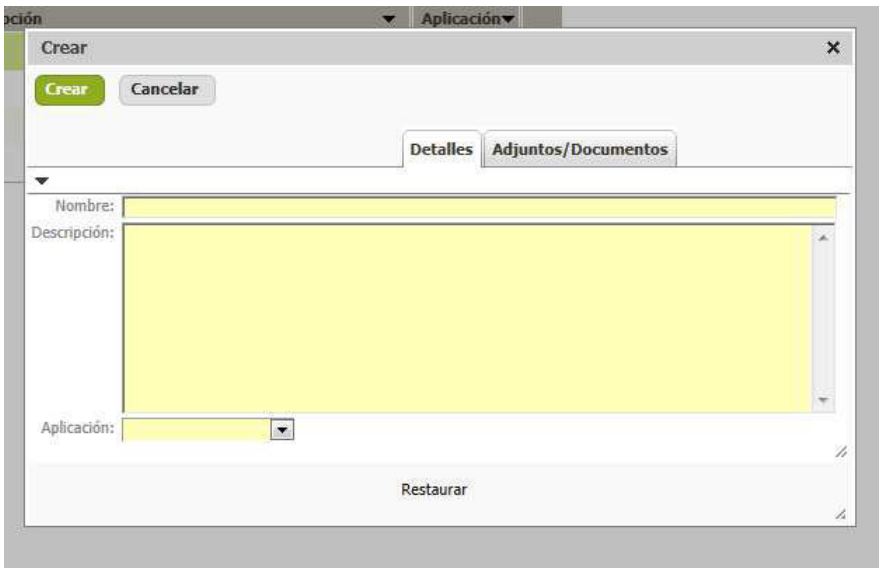


Imagen 3.18: Submódulo UseCase, estado crear.

```
@Length(max=100)
private String name;

@Length(max=1000)
private String description;
```

Imagen 3.19: Ejemplo de definición de atributos de tipo String, clase UseCase.

- Pestaña *Adjuntos/Documentos* del estado crear: Cuando pulsamos en la pestaña se nos muestra el siguiente contenido que vemos en la imagen 3.20. Esta pestaña es otra de las funcionalidades por defecto que nos ofrece eSengo®, pero para utilizarla debemos añadir la clase correspondiente en los *settings,* además de sobrescribir o añadir los métodos que necesitemos. Lo primero que nos llama la atención es el árbol de carpetas de más a la izquierda, este árbol es totalmente editable, partiendo siempre de la raíz podemos crear el árbol de directorios que nos plazca para almacenar de forma ordenada nuestros documentos. Si no creamos ninguna carpeta el sistema no nos dejará crear ningún documento. La única opción que no ha salido hasta ahora es la de *Arrastrar*, sirve para poder arrastrar un documento creado a la carpeta que deseemos. La acción de crear nos mostrará la pantalla, por defecto, que nos muestra la imagen 3.21. Podríamos editarla mediante los *settings* asociados a la clase, pero en nuestro caso no será necesario ya que los campos que muestra por defecto ya nos sirven. Destacamos: Documento Fuente, que es para adjuntar el documento que queramos añadir al sistema, por ahora solo acepta PDFs, pero en un futuro se añadirá un convertidor para poder aceptar varios formatos más. El campo Autor que consta de un componente llamado *LookupField,* siempre se utilizará cuando uno de los campos tenga un atributo que sea un objeto de una clase y la tabla asociada al submódulo al que corresponde la clase tenga numerosos registros, es un buscador que nos permite trasladarnos a la tabla que corresponde al objeto asociado al atributo y seleccionar el registro que nos convenga. Los asteriscos que vemos en los iconos (clip y lupa) son para indicarnos que ese campo es obligatorio. Si pulsamos el botón de *Crear,* en la parte superior izquierda, se validará que se hayan rellenado los campos obligatorios de la pestaña *Detalles*.

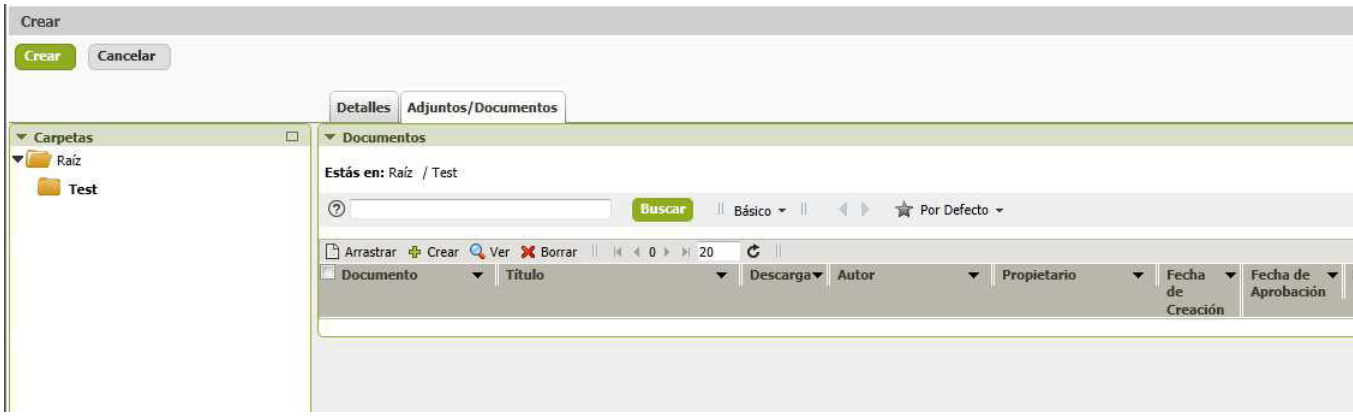


Imagen 3.20: Pestaña Adjuntos/Documentos del estado crear del submódulo UseCase.

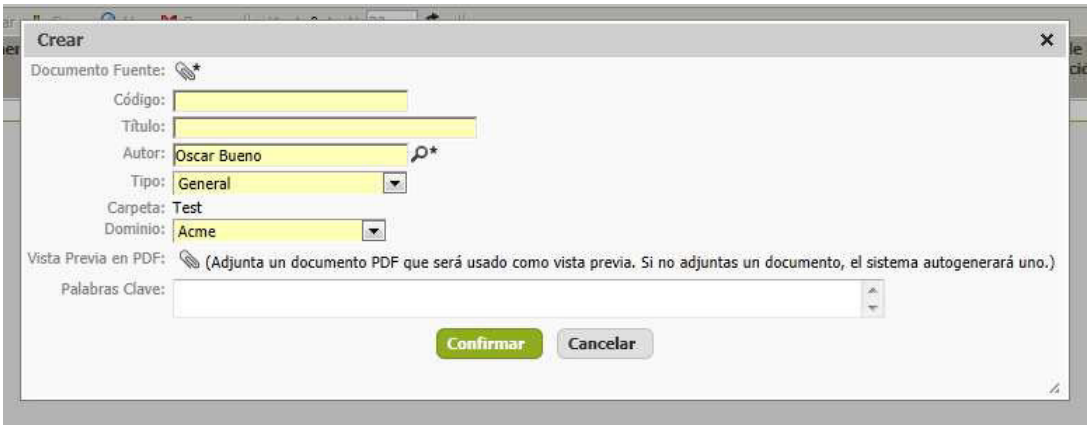


Imagen 3.21: Estado crear de la pestaña Adjuntos/Documentos del estado crear del submódulo UseCase.

- Ver un registro: Al utilizar la acción *Ver,* se abre un *popup* con la información del registro seleccionado, con las mismás características que en los submódulos anteriores, como podemos observar en la imagen 3.22. La diferencia aquí es que si vamos a la pestaña de Adjuntos/Documentos, veremos una tabla con los documentos asociados a este registro, pero si seleccionamos una de las carpetas del árbol de directorios podremos añadir más documentos al mismo, pero seguiremos sin poder editar el resto de atributos, esto es así por convención interna.


Imagen 3.22: Submódulo UseCase, estado ver.

- Editar y eliminar un registro: Análogo a las acciones del submódulo descrito en el punto 3.3.1. Cuando eliminamos un registro también eliminamos los documentos que tenga asociados.

### 3.3.4. Risk


Imagen 3.23: Submódulo Risk, estado inicial.

En la imagen 3.23 tenemos el estado inicial del submódulo *Risk* (Riesgo o Gestor de Riesgos), vemos una tabla con los registros correspondientes a los riesgos registrados, siempre según los criterios de los filtros, estos filtros son, definidos por *settings* ya que en este caso si que contamos con la opción de añadir más filtros, además los 3 filtros que aparecen por defecto son de tipo desplegable:

- Tipo: Busca según el tipo de riesgo que seleccionemos en el filtro, tenemos 4 tipos: Técnico, Funcional, Conformidad y Continuidad de Negocio.
- Severidad: La severidad nos sirve para saber si un riesgo debe ser abordado inmediatamente (no se puede tolerar) o por el contrario no afecta demásiado en el desarrollo de las acciones a las que esté asociado el riesgo. Tenemos 4 niveles de severidad:Emergencia, Alta, Media y Baja.
- Impacto: El impacto nos sirve para determinar el área de impacto del riesgo. Por convenio interno tenemos 4 niveles, pero pueden ser ampliables en un futuro dependiendo de las necesidades: Ámbito, Cronología, Presupuesto, Calidad.

Las acciones básicas son las mismás que los submódulos anteriores:

- Crear un nuevo registro: Como podemos ver en la imagen 3.24 al pulsar en la acción de crear se nos abre este *popup*, de tipo *dashboard*. Este estado tiene 4 campos de texto, Título, Comentarios, Mitigación y Descripción, 2 de ellos son obligatorios (Título y Comentarios). También tiene 5 campos de tipo desplegable: Tipo, Severidad, Impacto, Estado y Proyecto. El campo de Estado nos indica en que punto está el riesgo: Nuevo, En Proceso o Cerrado. El campo Proyecto nos permite asociar este riesgo a alguno de los proyectos abiertos, para poderlos tener en cuenta al analizar el proyecto y organizar sus tareas, así como para dar presupuestos a los clientes y estimar las horas que costará realizarlo. Finalmente tenemos 2 campos más que aun no habían aparecido, los campos de tipo fecha, SherpaBeans añade automáticamente el componente de fecha a los atributos que son definidos como tipo *Date*. Se puede introducir la fecha manualmente, pero si pulsamos el botón se nos abrirá un calendario, se nos muestra un ejemplo en la imagen 3.25, donde podremos navegar hasta encontrar la fecha que queramos introducir. Este componente también cuenta con un desplegable para poder seleccionar fechas concretas, en la imagen vemos que tipos tenemos. Cuando terminamos de crear el registro volvemos a la pantalla inicial, guardando el registro en memoria.

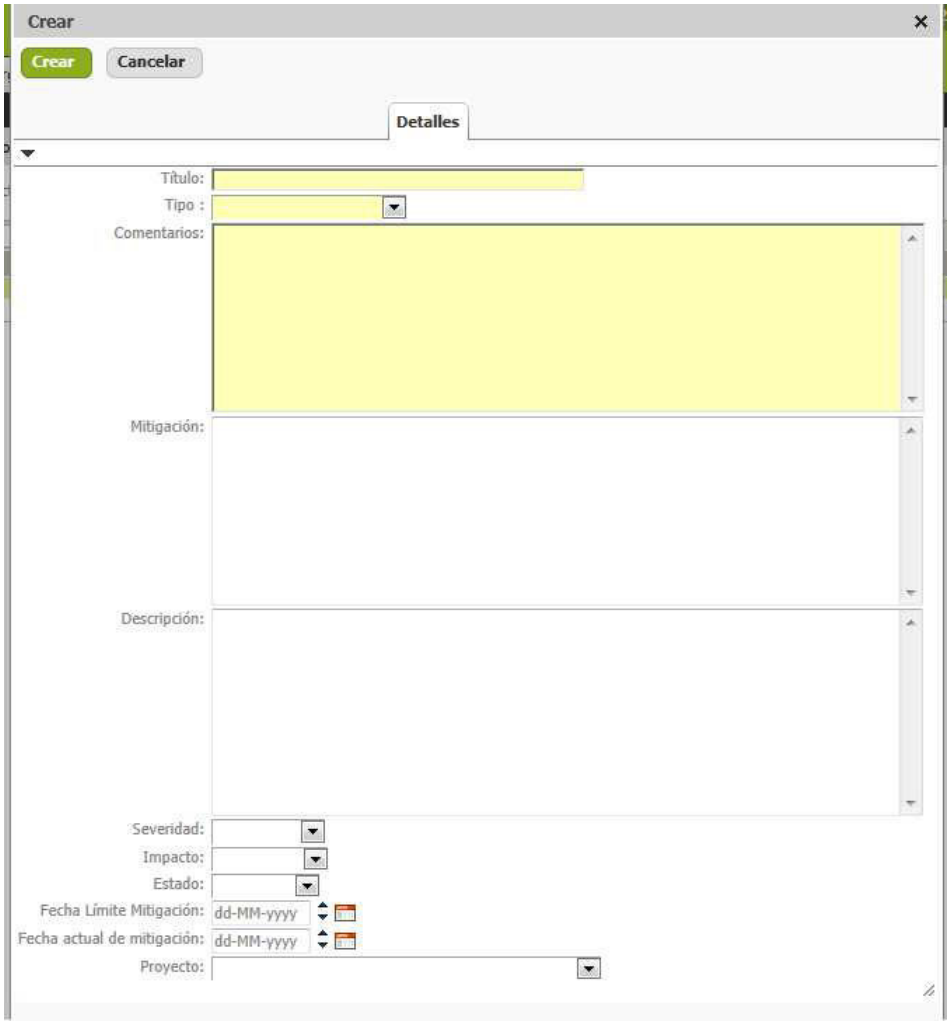


Imagen 3.24: Submódulo Risk, estado crear.

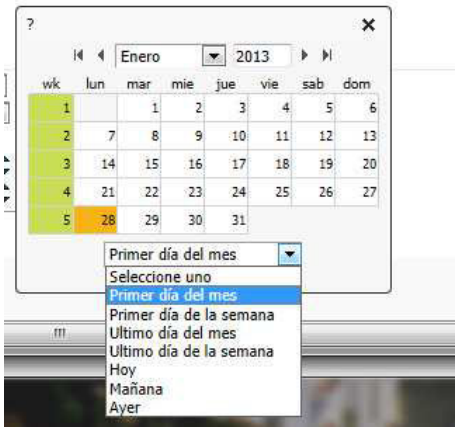


Imagen 3.25: Ejemplo del componente, de SherpaBenas, fecha.

- Ver un registro: Al utilizar la acción *Ver,* se abre un *popup* con la información del registro seleccionado, con las mismás características que en los submódulos anteriores, como podemos observar en la imagen 3.26.


Imagen 3.26: Submódulo Risk, estado ver.

- Editar y eliminar un registro: Análogo a las acciones del submódulo descrito en el punto 3.3.1.

## 3.4. Módulo ChangeRequest

Este módulo nos servirá para poder realizar las peticiones de cambio, consultarlas, editarlas, además de que estas deberán pasar por todo una serie de estados y transiciones hasta que lleguen al estado final. Primero nos centraremos en los elementos básicos de todo módulo, los estados de crear, ver, editar, eliminar...Seguidamente explicaremos el diagrama de flujo que debe seguir una petición de cambio, este flujo es similar para los 2 tipos de peticiones que tenemos, software y hardware. Las diferencias las encontraremos en los datos mostrados en los diferentes estados, los datos que se tengan que rellenar. De todos modos el módulo esta pensado para que si en algún momento alguno de los 2 flujos debe ser modificado se podrá hacer de manera independiente sin que afecte al otro, incluso si se añadieran más flujos de tipos de cambio, esto no afectaría a los ya definidos.

En la imagen 3.27 se puede observar la tabla que contiene las peticiones de cambio:


| Código de Petición | Tipo | Descripción del cambio | Nivel de riesgo de compilación | Nivel de riesgo de continuidad de negocio | Estado | Prioridad |
|---|---|---|---|---|---|---|
| 000012 | Software | test | Bajo | Bajo | | |
| 000014 | Software | test | | | | |
| 000016 | Software | test | | | | |
| 000002 | Hardware | test | | | | |
| 000003 | Hardware | test | | | | |
| 000004 | Hardware | test | | | | |
| 000005 | Hardware | test | | | | |
| 000020 | Software | test | | | | |
| 000006 | Hardware | test | | | | |
| 000007 | Hardware | test | | | | |
| 000022 | Software | test | | | | |
| 000026 | Software | | | | | |
| 000008 | Hardware | r | Bajo | Bajo | | |
| 000028 | Software | gfdsgdfsa | Medio | Bajo | | |

Imagen 3.27: Tabla de peticiones de cambio.

Se mostraran los registros de acuerdo con los criterios de los filtros, por defecto tenemos:

- Código de Petición: Busca según el código único que se genera al crear una nueva petición.
- Estado: Es un desplegable que contiene todos los estados posibles de una petición, filtra por el estado que le indiquemos.
- Tipo: Otro desplegable, en esta ocasión filtra por el tipo de petición, software o hardware.
- Prioridad: Este campo que está compuesto de varios *checkbox*, se pueden seleccionar los que se precisen, nos permite filtrar por la prioridad que se le haya asignado a la petición, este atributo ya tiene el componente por defecto, pero para crearlo solo hay que añadir el componente *CheckBoxField* al campo en cuestión.

Si observamos la tabla hay 2 columnas que nos llaman la atención, Estado y Prioridad, ya que no tienen texto, si no imágenes. Esta substitución se ha realizado mediante configuración de *settings*, se ha utilizado el objeto *TableSettings* para editar las propiedades de la tabla, y se le ha añadido a la tabla una funcionalidad de celda, para que substituya los valores por imágenes, en la imagen 3.28 observamos la linea de código para añadir esta funcionalidad a la tabla.

```
//Table Settings
TableSettings.setCellFactory(new ChangeRequestCellFactory());
```

Imagen 3.28: Linea de código de la clase ChangeRequestSettings.

Esta funcionalidad se ha programado en la clase *ChangeRequestCellFactory,* donde se ha indicado que columna será la afectada y que imágenes debe utilizar para substituir los valores, normalmente este tipo de componente debe utilizarse con atributos de tipo *enum* ya que se debe indicar en la clase la relación entre valores e imágenes, por lo tanto si se utilizaran atributos de tipos de datos diferentes podríamos encontrar con valores sin imágenes asociadas, ya que quizás el número de valores no dejarían nunca de crecer. Las imágenes se han debido añadir físicamente al *workspace*, no es necesario que estén en el mismo *package*, ni si quiera en el mismo plugin, ya que la clase *ChangeRequestCellFactory* extiende de *ActivityCellFactory* que tiene métodos para acceder a las rutas de las imágenes, así como realizar la substitución de valor por imagen. Si dejamos el puntero encima de la imagen nos aparece una descripción, esto también se configura en la clase *ChangeRequestCellFactory* con los métodos que hereda.

Como podemos observar la tabla de peticiones de cambio también dispone de las típicas acciones que ya hemos visto en los submódulos,a excepción de la acción de editar que es substituida por el símbolo que se muestra al lado del Código de Petición en las filas de la tabla, en la imagen 3.29 se observa mejor el símbolo. Este será nuestro editar ahora, se denomina *WorkWith*, ya que las peticiones son actividades y a diferencia de los submódulos, se podría decir que su estado de creación es todo un flujo que usa transiciones para transitar de un estado a otro, esto permite muchas más acciones además de la utilización de roles, también tienen un *ProcessModel* y un flujo de trabajo (*WorkFlow*) asociado. Por lo tanto este símbolo nos permitirá entrar en el estado actual en el que se encuentre la petición pero sin poder editar los atributos rellenados en estados anteriores, ni volver a estos estados, a no ser que exista alguna transición especifica para ello.

- Crear una nueva petición: Si pulsamos el botón de crear nos aparece el submenú de la imagen 3.29 para poder seleccionar que tipo de petición que queremos crear:

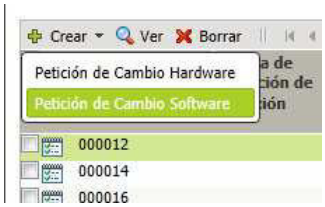


Imagen 3.29: Submenú botón crear, del módulo ChangeRequest.

- Software: Si seleccionamos el tipo software aparece un *popup* como el de la imagen 3.30. Encontramos 3 campos de texto obligatorios, Nombre, Descripción del cambio y Comentario del origen del cambio. Además también tenemos 3 campos de tipo desplegable obligatorios, Aplicación, Módulo y Origen del cambio. Los campos de Aplicación y Módulo están relacionados igual que la relación que existe entre ellos en base de datos, es decir que cuando seleccionamos una aplicación en el 1er campo, automáticamente se nos cargan los módulos que estén asociados a esta aplicación en el segundo desplegable, si intentamos seleccionar un elemento en el desplegable de Módulo sin antes seleccionar una aplicación, el desplegable de Módulo saldrá vacío. Además también contamos con un campo de tipo *LookupField,* Proyecto. También nos percatamos de que hay un nuevo botón llamado Crear/Siguiente, este se utiliza para poder crear varios registros del mismo tipo sin tener que salir de la pantalla de creación.

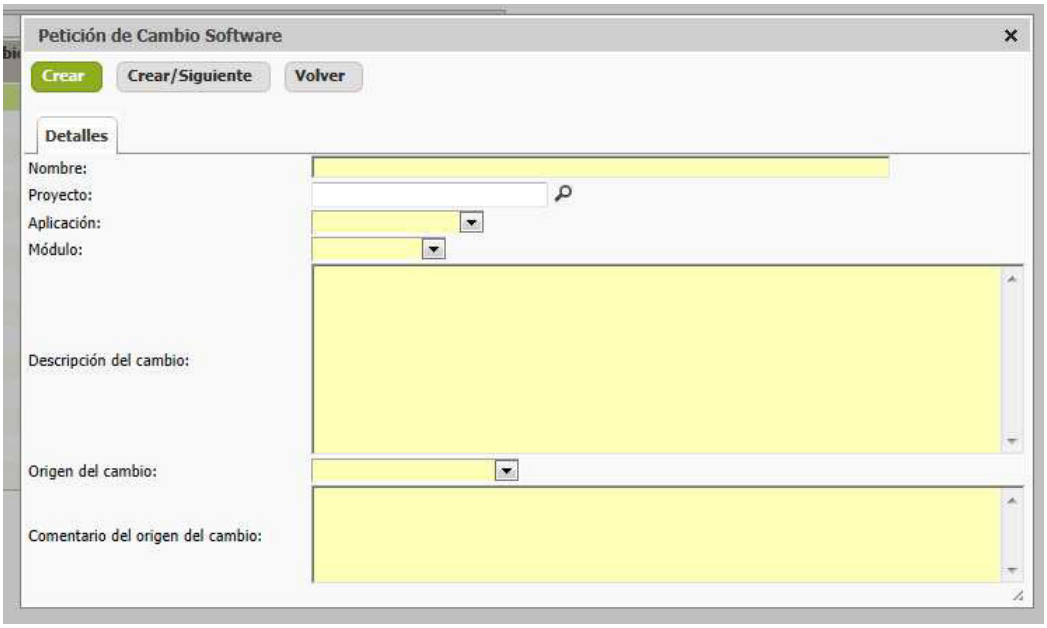


Imagen 3.30: Crear una petición de tipo software, del módulo ChangeRequest.

- Hardware: Si seleccionamos el tipo hardware aparece un *popup* como el de la imagen 3.31. Encontramos 3 campos de texto obligatorios, Nombre, Descripción del cambio y Comentario del origen del cambio. Además también tenemos 1 campo de tipo desplegable obligatorios, Origen del cambio. Además tenemos un campo de tipo *LookupField* obligatorio, es el Artículo, representara el elemento de hardware de la empresa que se querrá modificar, ya puede ser una pantalla de ordenador, una mesa de escritorio, etc.

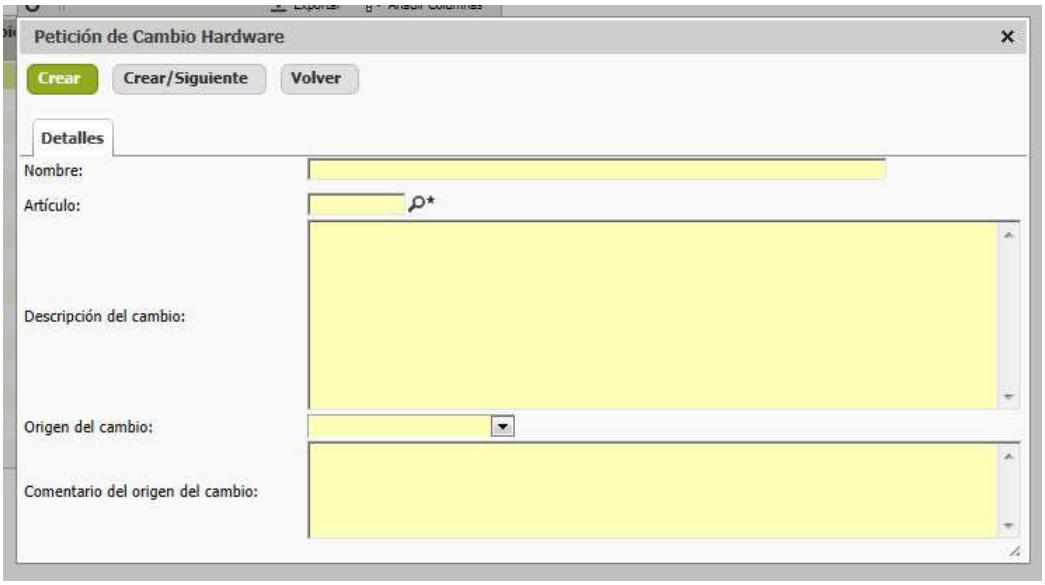


Imagen 3.31: Crear una petición de tipo hardware, del módulo ChangeRequest.

- Ver una petición: Dependiendo en el estado que se encuentre veremos más o menos campos, todos ellos no editables.
- Borrar una petición: Análogo a la acción de borrar de los submódulos explicados. Pero solo los roles específicos podrán hacerlo.

En la imagen 3.32 podemos ver el flujo de una petición de cambio, hardware y software. Recordemos que solo el rol específico podrá acceder a ciertos estados de la petición de cambio, esto se abordará más detalladamente en la explicación del flujo.

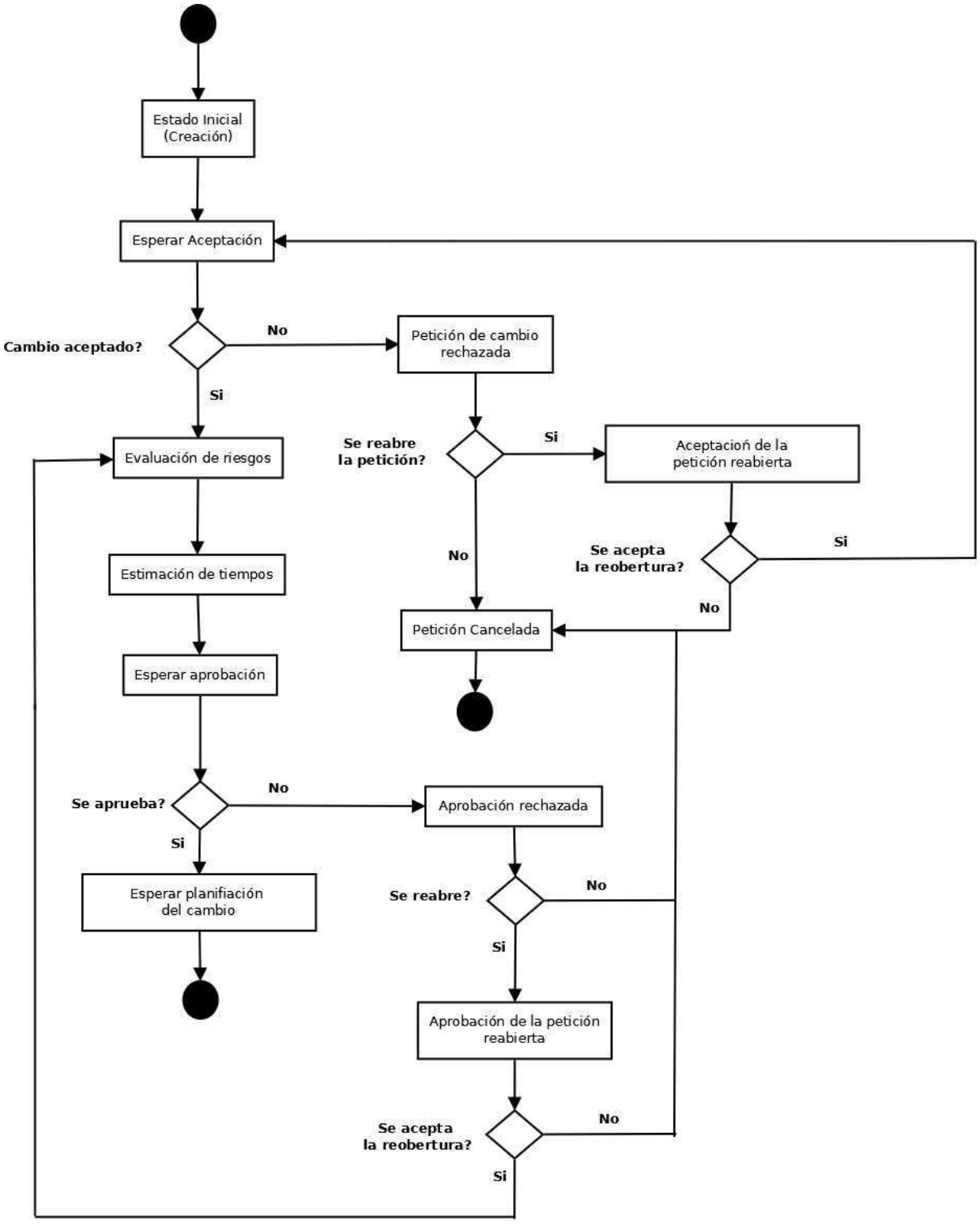


Imagen 3.32: Diagrama de flujo de una petición.

El estado inicial del flujo corresponde al estado de crear que acabamos de comentar. Cada vez que cambiamos de un estado a otro se anota en unos campos no editables internos de la clase, la fecha en la que sucede. El usuario podrá obtener esta información en los diferentes estados. Desde la tabla general también se puede obtener de 2 maneras, añadiendo estos campos como filtros y/o columnas.

### 3.4.1. Roles

Tendremos varios roles definidos que podrán acceder a diferentes estados de la petición, serán los siguientes:

- Change Requester (CR): Tendrá permisos para poder crear peticiones de cambio, así como acceder a los estados de Petición de cambio rechazada, para poder reabrir la petición y Esperar Aprobación, para dar el visto bueno a las estimaciones que le hayan adjudicado a su petición.
- Change Owner (CO): Tendrá acceso al estado de Esperar Aceptación, será el encargado de aceptar las peticiones de cambio que le vengan de los clientes, así como de establecer un presupuesto y una prioridad, para así pasar a evaluarlos. O rechazarlos y devolver la petición al cliente. También será el encargado de aceptar la reobertura solicitada por un cliente o cancelarla. También se encargara de aceptar o cancelar la reobertura de una petición no aprobada que PT está intentando reabrir.
- Quality Team (QT): Tendrá acceso al estado de Evaluación de riesgos. Serán los encargados de evaluar y crear los posibles riesgos que comporta el cambio, y añadirlos a la petición.
- Planning Team (PT): Tendrán acceso al estado de Estimación de tiempos, serán los encargados de decidir cuanto tiempo se tardará en realizar el cambio. Además también tendrá acceso a poder reabrir una petición que no ha sido aprobada por el CR. Normalmente PT y QT los tendrán las mismás personas.

### 3.4.2. Esperar Aceptación


Imagen 3.33: Estado ChangeRequestWaitAcceptanceActivity, tipo software.

En la imagen 3.33 vemos la pantalla correspondiente al estado de Esperar Aceptación. Como vemos solo tenemos un campo obligatorio que es el de la prioridad, ya que puede haber peticiones sin importe. En este caso tenemos todas las acciones ya que nuestro usuario tiene el rol de CO, estas acciones son:

- Volver: Vuelve a la tabla general sin modificar nada.
- Modificar: Modifica el registro con los datos introducidos y vuelve a la tabla general.

- Aceptar: Modifica el registro con los datos introducidos y acepta la petición, es decir pasa al estado de Evaluación de Riesgos.
- Rechazar: Rechaza la petición, es decir transita al estado Petición de Cambio Rechazada. Cuando se rechaza aparece un *popup* como el que se nos muestra en la imagen 3.34. Donde el usuario se ve obligado a introducir un comentario explicativo del porqué del rechazo

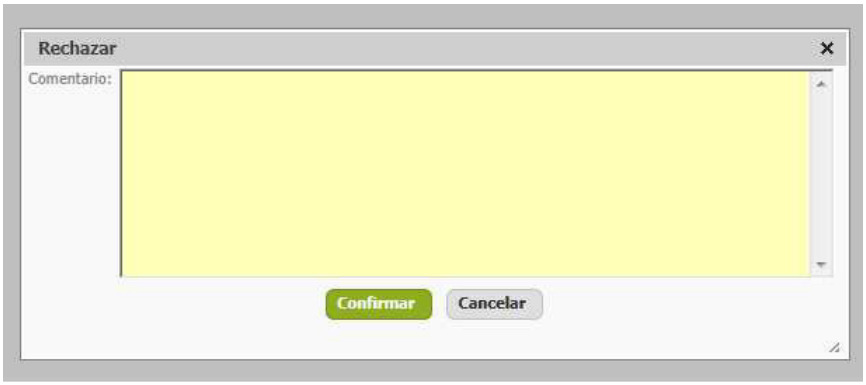


Imagen 3.34: Popup emergente al rechazar una petición.

También observamos que aparte de la pestaña con los datos principales existen 2 pestañas más:

- Comentarios: Como vemos en la imagen 3.35 esta pestaña consta de una única acción de añadir, esta acción sirve para poder añadir cualquier tipo de comentario a la petición. Al darle a la acción nos aparece un *popup* con la forma que se muestra en la imagen 3.36, con un solo campo de texto obligatorio. Una vez creado el comentario se muestra una pequeña tabla con algunos datos como el comentario y el creador del mismo, además como observamos en la imagen 3.37 también aparecen acciones en la fila para editar y eliminar el comentario. La mayoría de esta funcionalidad ya viene implementada por SherpaBeans, solo se ha tenido que añadir la pestaña en los *settings* correspondiente además de haber sido necesario de sobrescribir la clase correspondiente a la *view* de los comentarios para que solo mostrara la información relevante.


Imagen 3.35: Pestaña Comentarios del estado ChangeRequesWaitAcceptancetActivity.

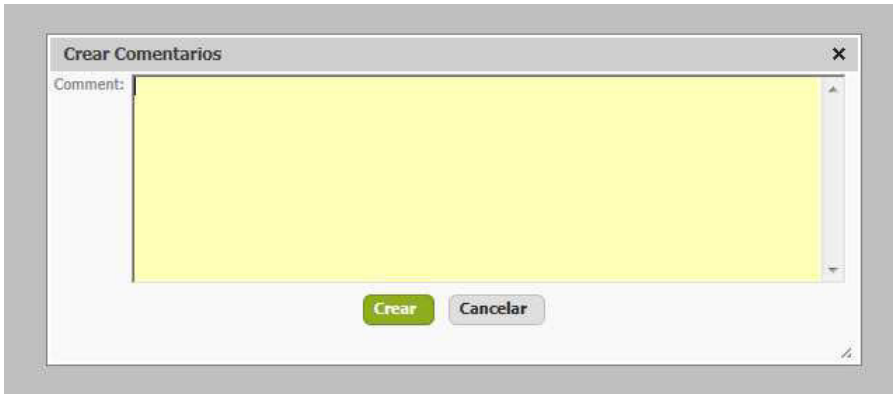


Imagen 3.36: Estado crear de la pestaña Comentarios del estado ChangeRequestWaitAcceptanceActivity.

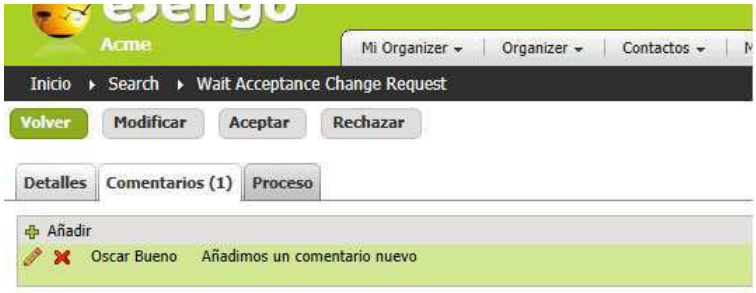


Imagen 3.37: Pestaña Comentarios del estado ChangeRequesWaitAcceptancetActivity, después de añadir un comentario.

- Proceso: Esta pestaña es otra de las características de eSengo®, está compuesta por varias pestañas más como podemos ver en la imagen 3.38. La pestaña principal es Actividades que nos muestra información sobre nuestra actividad, la hora de inicio, la de final, quien la tiene asignada, en que estado está, etc. La siguiente pestaña es el *Tracking*, esta característica de eSengo®, es muy común en todo el ERP, se usa para dejar constancia del tiempo utilizado para realizar las tareas asignadas a un usuario. La pestaña Diagrama te muestra en que estado esta la petición mediante un diagrama de bloques. Finalmente la pestaña Eventos informa de los eventos que se van sucediendo, en nuestro caso los eventos serán las asignaciones de la actividad a diferentes personas.


Imagen 3.38: Pestaña Comentarios del estado ChangeRequestWaitAcceptanceActivity.

Een cuanto a la petición de tipo Hardware la configuración es la misma, excepto que el campo que se refiere al módulo es substituido por el campo Artículo. Salvo indicación esto pasará en todos los demás estados.

### 3.4.3. Evaluación de riesgos

Vemos en la imagen 3.39 la pantalla correspondiente al estado de Evaluación de riesgos. Además de la información no editable que irá aumentando a medida que la vayamos introduciendo. Encontramos 2 campos de texto obligatorios y 2 campos de tipo combo obligatorios, son idénticos a los campos que se vieron en el submódulo 3.3.4, tienen nombres similares, pero los de la petición de cambio van asociados al riesgo de realizar dicho cambio en el módulo seleccionado. Recordemos que solo los usuarios con el rol de Quality Team podrán acceder a este estado, el resto solo podrá utilizar la acción de ver el estado. El Quality Team también podrá utilizar las siguientes acciones:

- Volver: Vuelve a la tabla general sin modificar nada.
- Modificar: Modifica el registro con los datos introducidos y vuelve a la tabla general.
- Aceptar: Modifica el registro con los datos introducidos y acepta la petición, es decir pasa al estado de Estimación de tiempos.

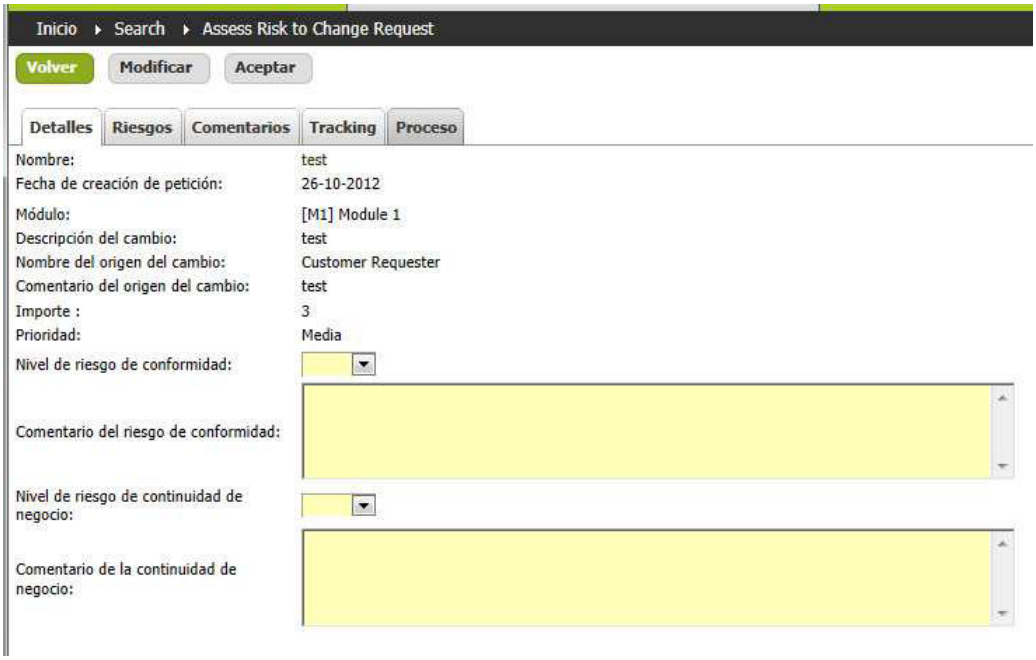


Imagen 3.39: Estado ChangeRequestWaitAssessRisksActivity, tipo software.

Y vemos 2 nuevas pestañas que en el estado anterior no figuraban:

- Riesgos: En él encontraremos una tabla similar a la del submódulo 3.3.4, donde se podrán añadir riesgos ya creados, así como crear de nuevos para asociarlos.
- Tracking: Es la tabla que se ha comentado anteriormente que sirve para añadir el tiempo utilizado para realizar cualquier tarea, normalmente tendrá la siguiente forma, siempre que esté habilitada para el estado en cuestión, en nuestro caso lo está y tiene la siguiente forma que nos muestra la imagen 3.40. Se han utilizado los *settings* para añadir la pestaña que se encuentra en *process*, en las pestañas del estado de la petición. Los campos, Und (Unidades), Usuario son obligatorios, Mon puede ser rellenado o no, igual que los comentarios. Una vez rellenados se añadirá un nuevo registro al tracking asociado a esta petición, se podrá consultar desde la pestaña tracking o desde la pestaña proceso->tracking. El tracking nos servirá para poder ver el tiempo utilizado para realizar las tareas correspondientes a cada estado.


Imagen 3.40: Pestaña Tracking del estado ChangeRequestWaitAssessRisksActivity.

Análogamente al estado anterior si la petición es de tipo Hardware el campo Módulo deja su lugar al campo Artículo. Aunque normalmente los tipos de riesgo que tengan asociados estas peticiones serán de niveles bajos.

### 3.4.4. Estimación de tiempos

En la imagen 3.41 se nos muestra la pantalla correspondiente al estado de Estimación de tiempos. Encontramos que solo tenemos un campo editable, obligatorio, correspondiente a las horas que se estime que puede durar la realización del cambio, recordemos que solo los usuarios con el rol de Planning Team. Además la información no editable irá aumentando a medida que la vayamos introduciendo y vayamos transitando de estado. También tenemos la siguientes acciones:

- Volver: Vuelve a la tabla general sin modificar nada.
- Modificar: Modifica el registro con los datos introducidos y vuelve a la tabla general.
- Aceptar: Modifica el registro con los datos introducidos y acepta la petición,pasando al siguiente estado de Esperando Aprobación.


Imagen 3.41: Estado ChangeRequestEstimationActivity, tipo software.

Vemos una nueva pestaña llamada Casos de Uso afectados solo aparecerá si la petición tiene algún riesgo que sea de tipo medio o alto, o sus niveles de riesgo son medios o altos. Dentro de la pestaña encontraremos una tabla con casos de uso, el contenido de esta tabla se genera automáticamente utilizando una clase de tipo *Criteria*, llamada *ChangeRequestUseCaseCriteria*, la podemos ver en la imagen 3.42. Como podemos observar se recupera el objeto por defecto con el que esté trabajando el flujo de ejecución con el comando *ModelUtils.findDefaultBean()*. Para poder parsear este objeto primero debemos saber a que instancia pertenece, de ahí el *if* de la linea 9. En la linea 10 realizamos el parseado y así ya tenemos nuestro objeto de tipo ChangeRequest, si el objecto cumple las condiciones para que deba tener UseCase, se añade al criteria asociado, el módulo por el que buscara los UseCase que le hagan referencia, esto se hace con el método de

clase *Criteria, setValue(<propertie>, value)*. En la imagen 3.43 podemos observar una tabla de ejemplo de la pestaña Caso de uso afectados.

```
1  public class ChangeRequestUseCaseCriteria extends EnterpriseEntityCriteria{
2
3      private static final long serialVersionUID = 1L;
4
5      public ChangeRequestUseCaseCriteria(IBeanKey beanKey) {
6          super(beanKey);
7          Object bean = ModelUtils.findDefaultBean();
8          ChangeRequest request = null;
9          if(bean instanceof TaskActivityInstance)
10             request = (ChangeRequest) ((TaskActivityInstance)bean).getBean();
11         if(request!=null && (request.getRequestType().equals(RequestType.SOFTWARE)) &&
12                 (request.getComplianceRiskLevel().equals(RiskLevelType.MEDIUM)||
13                 request.getComplianceRiskLevel().equals(RiskLevelType.HIGH)||
14                         request.getBusinessContinuityRiskLevel().equals(RiskLevelType.MEDIUM)||
15                         request.getBusinessContinuityRiskLevel().equals(RiskLevelType.HIGH))){
16             setValue("module", request.getModule());
17         }
18
19
20     }
21
22
23 }
```

Imagen 3.42: Clase ChangeRequestUseCaseCriteria..


Imagen 3.43: Pestaña Casos de uso afectados del estado ChangeRequestEstimationActivity.

En el caso de una petición de tipo Hardware nos encontraríamos con la misma configuración, haciendo el cambio del campo Módulo por el de Artículo. Además la pestaña de casos de uso no aparecerá, ya que no tiene sentido.

### 3.4.5. Esperar Aprobación

En la imagen 3.44 se nos muestra la pantalla correspondiente al estado de Esperar Aprobación. No hay ningún campo editable ya que este estado sirve para que el dueño de la petición acepte o no el análisis que se le ha hecho a su petición, así como el posible presupuesto ofrecido, recordemos que solo los usuarios con el rol de Change Requester y que haya creado la petición podrá acceder a este estado. También tenemos la siguientes acciones:


Imagen 3.44: Estado ChangeRequestWaitRequesterApprovalActivity, tipo software.

- Volver: Vuelve a la tabla general sin modificar nada.
- Aprobar: Da el visto bueno al análisis presentado y aprueba todos los valores presentados, tiempo, prioridad, presupuesto. Esto nos llevará al estado final Esperar planificación del cambio.
- Rechazar: Rechaza el análisis de la petición presentado,pasando al siguiente estado, Esperando Aprobación. También aparecerá un *popup*, con un campo de texto obligatorio, para que el usuario que rechaza pueda poner sus motivos.

Aparte de las pestañas de siempre aquí encontramos una nueva que nos muestra información sobre la petición, como la durada de ciertos estados o si ha sido rechazada en algún momento, en la imagen 3.45 se nos muestra un ejemplo del contenido de la pestaña Tiempo de Respuesta.

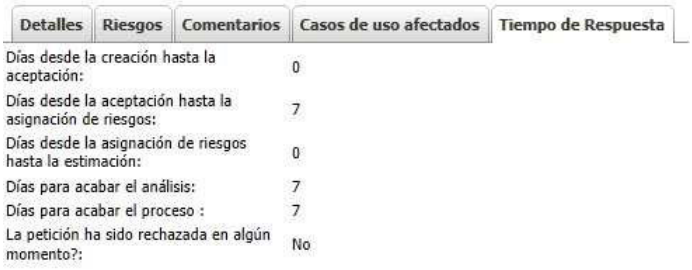


Imagen 3.45: Pestaña Tiempo de Respuesta del estado ChangeRequestWaitRequesterApprovalActivity.

En el caso de una petición de tipo Hardware nos encontraríamos con la misma configuración, haciendo el cambio del campo Módulo por el de Artículo.

### 3.4.6. Esperar planificación del cambio

Este estado será el final de nuestra petición de cambio si ha seguido todo el flujo normal y se han ido aprobando. En este estado los campos no serán editables ya que a partir de ahora servirán para ser consultadas y planificar el cambio del método que más convenga a los usuarios encargados de realizarlo.En el caso de una petición de tipo Hardware nos encontraríamos con la misma configuración, haciendo el cambio del campo Módulo por el de Artículo.

### 3.4.7. Petición de cambio rechazada

En la imagen 3.46 se nos muestra la pantalla correspondiente al estado de Petición de cambio rechazada. No hay ningún campo editable ya que este estado sirve para que el los usuarios con el rol de Customer Requester puedan reabrir la petición para volver a enviarla al Change Owner o cancelarla.Tenemos la siguientes acciones:


Imagen 3.46: Estado ChangeRequestRejectedAcceptanceActivity.

- Volver:Vuelve a la tabla general.
- Abrir: Reabre la petición para volver a analizarla, pasa al estado Aceptación de la petición reabierta
- Cancelar: Cancela la petición y esta acaba su flujo de ejecución en el estado final Petición cancelada.

### 3.4.8. Aceptación de la petición reabierta

En la imagen 3.47 se nos muestra la pantalla correspondiente al estado de Aceptación de la petición reabierta. Tiene los mismos campos que el estado inicial del tipo de petición que toque, aunque con diferente formato ya que no es el mismo estado, además debe ser reenviada de nuevo al Change Owner, en caso de ser confirmada, su siguiente estado será Esperar aceptación. Tiene las siguientes acciones:

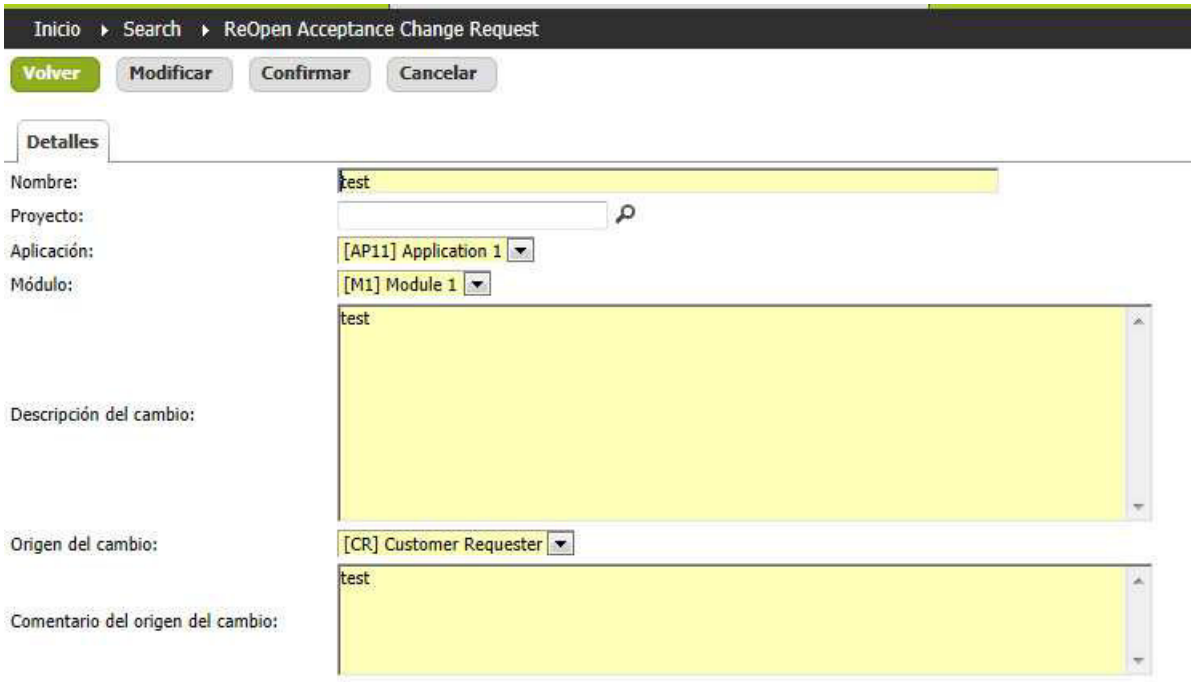


Imagen 3.47: Estado ChangeRequestReopenAcceptanceActivity

- Volver: Análoga al resto de apartados.
- Modificar: Permite modificar la petición reabierta, pero se queda en el mismo estado.
- Confirmar: Guarda los datos que se hayan modificado y transita al nuevo estado Esperar aceptación.
- Cancelar: Cancela la petición y esta acaba su flujo de ejecución en el estado final Petición cancelada.

### 3.4.9. Aprobación Rechazada

En la imagen 3.48 se nos muestra la pantalla correspondiente al estado de Aprobación Rechaza. No hay ningún campo editable ya que este estado sirve para que los usuarios con el rol de Planning Team puedan reabrir la petición para volver a analizarla o cancelarla.Tenemos la siguientes acciones:

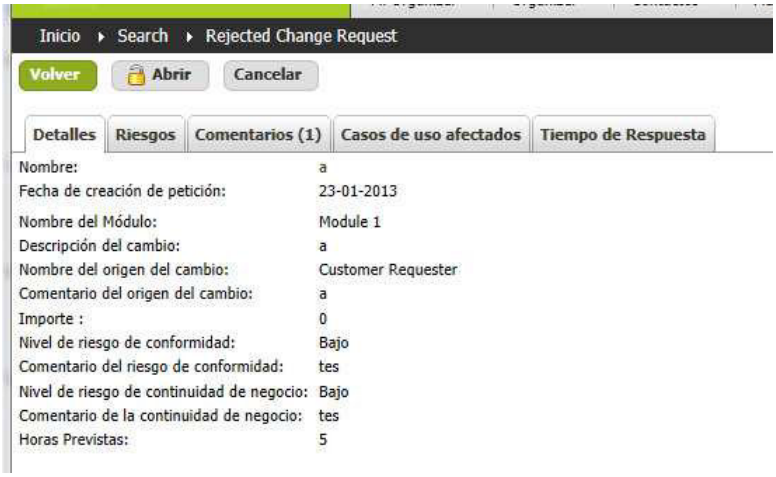


Imagen 3.48: Estado ChangeRequestRejectedActivity.

- Volver:Vuelve a la tabla general.
- Abrir: Reabre la petición para volver a analizarla, pasa al estado Aprobación de la petición reabierta.
- Cancelar: Cancela la petición y esta acaba su flujo de ejecución en el estado final Petición cancelada.

En el caso de una petición de tipo Hardware nos encontraríamos con la misma configuración, haciendo el cambio del campo Nombre del Módulo por el de Artículo.

### 3.4.10. Aprobación de la petición reabierta

En la imagen 3.49 se nos muestra la pantalla correspondiente al estado de Aprobación de la petición reabierta. Tiene los mismos campos que el apartado 3.4.2 ya que debe ser analizada de nuevo en caso de ser confirmada, su siguiente estado será Evaluación de riesgos. Tiene las siguientes acciones:


Imagen 3.49: Estado ChangeRequestReopenActivity

- Volver: Análoga al resto de apartados.
- Modificar: Permite modificar la petición reabierta, pero se queda en el mismo estado.
- Confirmar: Guarda los datos que se hayan modificado y transita al nuevo estado Evaluación de riesgos.
- Cancelar: Cancela la petición y esta acaba su flujo de ejecución en el estado final Petición cancelada.

En el caso de una petición de tipo Hardware nos encontraríamos con la misma configuración, haciendo el cambio del campo Nombre del Módulo por el de Artículo.

#### 3.4.11. Petición cancelada

Este estado será el final de nuestra petición de cambio si ha sido cancelada. En este estado los campos no serán editables ya que la petición ha sido cancelada y ya no se utilizará más.

# 4. Validación

Terminado el desarrollo del módulo y submódulos, se han tenido que validar el correcto funcionamiento del mismo. En este capitulo resumiremos como se han encarado las pruebas y alguno de los aspectos más importante a tener en cuenta.

## 4.1. Validación del módulo y los submódulos

Mientras se han desarrollado los diferentes submódulos y el módulo general, se han ido validando a medida que se iban finalizando submódulos y estados del módulo general. Las validaciones se han centrado básicamente en la interficie gráfica. Se han tenido en cuenta estos aspectos:

- Accesos a los recursos según los permisos definidos.
- Acceso a estados del flujo de ejecución según los roles definidos.
- Visibilidad de las tablas de registros y sus diferentes opciones de filtraje.
- Controlar que los flujos de ejecución sean correctos, por ejemplo que cuando se rechaza una petición vaya al estado que toca.
- Validación de campos obligatorios en los formularios de los diferentes submódulos así como en todos los estados del flujo de ejecución del módulo general.
- Comprobación que se almacenan correctamente los datos introducidos por el usuario.
- Aspectos de estética que tiene que seguir la interficie gráfica.

Estas validaciones se realizan mediante un navegador web, conectándonos al servidor donde está desplegado eSengo®

# 5. Conclusiones

Para darle más funcionalidad al ERP eSengo® se han creado los submódulos, ya descritos anteriormente,y el módulo de petición de cambios para poder tener más valor añadido. Con ello se han cumplido la mayoría de objetivos:

- Gracias a las reuniones mantenidas así como a la documentación consultada se ha podido entender como funciona un Gestor de Cambios que sea consistente con los estándares de ITIL. Así pues se han utilizado los diferentes roles, además de desarrollar el modulo de Peticiones de Cambio que basa su flujo de ejecución en el que define el estándar de ITIL.
- El hecho de haber realizado tanto los submódulos como los módulos ha ayudado a aprender sobre como utilizar la plataforma SherpaBeans y se ha aprendido a manejar algunas de las funcionalidades que ofrece el ERP eSengo®.

- Se ha aprendido a crear permisos mediante la plataforma SherpaBeans, estos se definen en la clase *ChangeManagerFeatures*, donde aparte de crear los permisos de acceso a cada submódulo, también es posible definir a que acciones del submódulo da acceso cada permiso.
- Se han cumplido todos los requerimientos que definió en un principio la empresa: añadir *dashboards*, gestor con diferentes estados, etc. Finalmente la empresa se desmarcó bastante del proyecto, dejando bastante libertad para añadir algún requerimiento extra, como por ejemplo la pestaña de Tiempo de Respuesta.

Estos objetivos se han cumplido satisfactoriamente y esto ha provocado que tengamos un modulo de gestión de peticiones intuitivo y bastante amigable al usuario. No ha sido fácil ya que, tanto SherpaBeans como eSengo® es software en constante cambio, esto supuso que a media implementación del proyecto se tuviera que adaptar el mismo a los cambios que la empresa consideró necesario, esto supuso la perdida de mucho tiempo ya que se tuvieron que rehacer varias clases y entidades, además de añadir los *dashboards* en todos los submódulos.

## 5.1. Posibles ampliaciones

Una vez acabado el desarrollo del módulo así como de los submódulos, se nos ocurren posibles ampliaciones para el futuro y así añadir más funcionalidad al gestor.

- Tanto la plataforma SherpaBeans como el ERP eSengo® son elementos en cambio constante, así pues una de las ampliaciones sería adaptar el código a la versión más nueva de SherpaBeans utilizando los últimos elementos que ofrezca eSengo® y que nos faciliten la implemntación, o que haga la aplicación más amigable al usuario.
- Otra posible ampliación sería la de enlazar el último estado de la petición de cambio , *Esperar planificación del cambio*, al flujo normal de eSengo®, para así, mediante algún estado más poder desglosar la petición en diferentes tareas y asignarlas a los responsables de hacerlas.
- Poner todos los estados del flujo de ejecución de una petición en *dashboards*, para tener más opciones visuales de personalización.
- Añadir más tipos de petición de cambio, por ejemplo cambios en documentos o en personal, para hacer el módulo aun más funcional y genérico.

# 6. Bibliografía

A continuación se exponen todas las referencias bibliográficas de la memoria:


[1] http://gbeaubouef.wordpress.com/tag/cloud-computing/
[2] http://www.informatica-hoy.com.ar/software-erp/Conceptos-basicos-del-ERP-Enterprise-Resource-Planning.php
[3] http://www.isencia.com
[4] http://www.esengo.es
[5] http://www.isencia.be/services/sherpabeans
[6] http://itil.osiatis.es/Curso_ITIL/Gestion_Servicios_TI/gestion_de_cambios/introduccion_objetivos_gestion_de_cambios/conc
[7] http://itilv3.osiatis.es/transicion_servicios_TI/gestion_cambios/ introduccion_objetivos.php
[8] http://www.eclipse.org
[9] http://java.sun.com/javaee/technologies/javaee5.jsp
[10] https://www.isencia.com/website/app/technologies?flowSrc=0
[11] http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html
[12] http://www.pcmag.com/encyclopedia_term/0,2542,t=application+programming+interface&i=37856,00.asp
[13] http://sociedadinformacion.fundacion.telefonica.com/DYC/SHI/seccion=1188&idioma=es_ES&id=2009100116300061&activo=4
[14] http://wicket.apache.org/meet/features.html
[15] http://www.hibernate.org/
[16] Java Persistence with Hibernate - Christian Bauer, Gavin King
[17] http://struts.apache.org/
[18] http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html
[19] http://tapestry.apache.org/
[20] http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications
[21] http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/Servlets.html
[22] http://www.java-samples.com/showtutorial.php?tutorialid=576
[23] http://docs.oracle.com/javase/1.4.2/docs/guide/security/jaas/tutorials/GeneralAcnOnly.html
[24] http://quark.humbug.org.au/publications/ldap/ldap_tut.html
[25] http://www.javaworld.com/jw-09-2002/jw-0913-jaas.html

## Summary

In this memory we discuss about the implementation of a change request manager based on ITIL's standard definitions. This manager is included in eSengo, an ERP developed by Isencia S.L. Although there are an explanation about the developed sub-module that gives functionality to the manager. Also we introduce some features about eSengo ERP and SherpaBeans platform, based on JavaBeans and developed by Isencia, also there is an enumeration of technologies that uses SherpaBeans.

## Resum

Aquesta memòria tracta sobre l'implementació d'un gestor de peticions de canvi, basat en els estàndard definits per ITIL. Aquest gestor s'inclou a l'ERP eSengo, desenvolupat per Isencia S.L. També es fa una breu explicació sobre els submòduls necessaris i els estats del fluxe de execucció del gestor. A més a més s'expliquen característiques sobre l'ERP eSengo i la plataforma basada en JavaBeans, SherpaBeans, també desenvolupada per Isencia, també s'en enumeren les teconologíes utilitzades per aquesta plataforma.

## Resumen

Esta memoria trata sobre la implementación de un gestor de peticiones de cambio, basado en los estándares definidos por ITIL. Este gestor se implementa dentro del ERP eSengo, desarrollado por Isencia S.L. También se hace una breve explicación sobre los submódulos que han sido necesarios y los estados del flujo de ejecución del gestor. Además se explican características sobre el ERP eSengo y sobre la plataforma basada en Java Beans, SherpaBeans, también desarrollada por Isencia, aparte de enumerar las tecnologías que esta plataforma utiliza.