Desarrollo de interfaces de usuario para una herramienta de telecontrol y automatización industrial

Proyecto de Fin de Carrera

Autor: Ángel Merino Sastre
Alumno de la Facultad de Informática de la ULPGC.

Tutor: Dr. Alexis Quesada Arencibia

20 de mayo de 2014

Agradecimientos

Este trabajo de fin de carrera es el fruto de muchos años de estudio y esfuerzo. Años en los que me he ido formando. Años en los que he ido creciendo como persona y como profesional de la informática. Pero no sería justo decir que todo el esfuerzo ha sido solamente mío. Por ello quisiera agradecer en primer lugar a mi familia. A mi madre por estar siempre ahí y por no dudar ni un solo segundo de mí. A mi hermana por hacerme sentir orgulloso de ella. En segundo lugar mi tutor Alexis por su paciencia y disponibilidad, y por su comprensión a mis idas y venidas. Tercero a mi pareja Kristina por enseñarme el camino y hacerme feliz. También quisiera agradecer a la universidad por formarme y por todo lo vivido durante tantos años. Al programa Erasmus sin el cual no hubiera encontrado el sentido a todo esto.

“A tutti quanti”, Gracias.

Índice de contenidos

1. Introducción

En este documento se describe el proyecto de fin de carrera titulado “Desarrollo de interfaces de usuario para una herramienta de telecontrol y automatización industrial”. Este proyecto software se presenta como una actualización en profundidad de la aplicación de telecontrol industrial *Hecaton*, desarrolla y ampliada como consecuencia de los respectivos proyectos de fin de carrera de los alumnos Ignacio Solinis Camalich [1], Adrián Peñate Sánchez [2] y Andrés del Pino Bolaños [3]. El objetivo del mismo es llevar el software y en especial su interfaz web a tecnologías actuales que hagan más atractivo no solo su uso sino su posterior desarrollo y mantenimiento.

1.1. Origen del proyecto

La idea original de la aplicación *Hecaton* nace en 2006 como proyecto de fin de carrera del alumno Ignacio Solinis Camalich cuyo título es “Sistema de Telecontrol de Plantas Desaladoras” [1] como solución para el telecontrol de plantas desaladoras instaladas en el archipiélago canario. La idea se materializa en 2007 y la primera versión del mismo es entregada e incluso puesta en funcionamiento en una instalación en Fuerteventura.

En esta primera versión se estableció en núcleo de la aplicación y su diseño modular incorporando la lectura de sensores, alarmas, envío de SMS y la gestión de *logs*.

Posteriormente el alumno Adrián Peñate Sánchez en su proyecto “Un sistema de monitorización y telecontrol para las instalaciones del servicio de Alojamiento Universitario de la ULPGC” [2], realizó una ampliación de la aplicación para una tarea diferente. Demostrando así la versatilidad de la aplicación y el acierto de su diseño modular. En este proyecto se añadieron la posibilidad de actuar sobre la instalación así como un *Applet* para la visualización de la planta.

El último ciclo de desarrollo fue llevado a cabo por el alumno Andrés del Pino Bolaños. En su proyecto de fin de carrera “Herramienta de gestión para plantas de tratamiento de aguas y sistemas de energía eléctrica en paneles solares fotovoltaicos” [3], se añade la posibilidad de crear secuencia de acciones así como las secciones como principales características.

Durante este tiempo la aplicación ha mantenido las mismas herramientas software desde su versión inicial. Desde el servidor de aplicaciones *JBoss AS 4.0.5*, la especificación *JEE 5* hasta la versión 1 de *Struts* para la generación de la interfaz. Durante estos años la tecnología ha cambiado y de alguna manera la aplicación no ha sido adaptada a la misma frenando sus posibilidades. Desde esta lógica reflexión nace la motivación para este proyecto de fin de carrera. Un nuevo ciclo de desarrollo que actualice *Hecaton*, de modo que se convierta en un software que usa métodos de desarrollo y herramientas utilizados hoy en día.

1.2. Estructura de la memoria

Esta memoria de proyecto de fin de carrera, debido a extensión y contenido, ha sido estructura de diferentes bloques, con el fin de hacer al lector más cómoda su lectura. En las siguientes líneas se detalla la estructura del documento:

En el primer bloque, introducimos al lector al proyecto. En él se describe qué es y cómo funciona la aplicación *Hecaton*. Posteriormente se enumeran los objetivos y oportunidades de mejoras que se han abordado en este trabajo. Posteriormente se definen las líneas de actuación y los recursos utilizados para llevar a cabo las tareas propuestas.

- **Capítulo1. Introducción**: Descripción de la aplicación y sus orígenes. Posibilidades de mejoras observadas y objetivos que se abordan en este proyecto. Introducción a las metodologías de desarrollo utilizadas para realizar el trabajo.

- **Capítulo 2. Análisis**: Análisis de aplicación. Descripción de las tareas a realizar. Se introduce al lector a las tecnologías y dispositivos usados por la aplicación. Finalmente se ofrece una visión de los casos de uso llevados a la última versión del software.

Ya en el segundo bloque, discutiremos sobre el diseño de las mejoras introducidas en la aplicación. Cada iteración cuenta con una descripción del trabajo realizado. Se enumerar las soluciones adoptadas y el cómo y porqué se han abordado de la manera elegida. Después, identificamos los detalles más importantes de la implementación.

- **Capítulo 3. Diseño**: Aquí se pone de manifiesto el trabajo realizado y las decisiones tomadas. Para cada iteración se ofrece una argumentación precisa de las soluciones adoptadas y del trabajo realizado.

- **Capítulo 4. Implementación**: Se ofrece los apuntes más destacables relacionados con el trabajo de codificación. La intención es poner de manifiestos las vicisitudes y detalles del trabajo de desarrollo software realizado.

En el tercer bloque se presenta al lector las conclusiones y reflexiones acerca del trabajo realizado. Se ofrecen las impresiones más destacadas después de implementar y probar la actualización del software. Junto a ello, se incluye los posibles trabajos futuros desde la visión del desarrollador.

- **Capítulo 5. Resultados y conclusiones**: En esta sección se presentan los resultados del trabajo realizado así como las conclusiones una vez terminadas las tareas.

- **Capítulo 6. Trabajos futuros**: Descripción de las líneas de trabajos más interesantes para los posibles nuevos desarrollos de la aplicación.

Finalmente, se incluye un bloque de contenidos extra donde encontramos los anexos a esta memoria. Se trata de información que describe el funcionamiento del sistema y otros documentos importantes generados a partir del trabajo realizado.

- **Anexo A. Manual de uso de la aplicación**: Actualización del manual de usuario que describe el funcionamiento completo de la aplicación.

- **Anexo B. Manual para el desarrollador**: Guía para continuar el desarrollo de la aplicación. Describe cómo utilizar la información del DVD adjunto y comenzar a trabajar con la aplicación.

- **Anexo C. Casos de uso completos**: Tablas que contienen la información detallada referente a los casos de uso.

- **Capítulo 7. Bibliografía**: Resumen de las referencias bibliográficas principales utilizadas para el desarrollo del trabajo así como para la redacción de esta memoria.

1.3. Descripción de la aplicación

Hecaton permite controlar de manera remota instalaciones de todo tipo. Hace posible realizar una monitorización completa de todos los parámetros requeridos a través de una serie de sensores distribuidos en los puntos de interés de la misma. Estos se conectan a través de módulos de adquisición de datos a un equipo informático que hace las veces de servidor a través del cual es posible consultar su estado. Así mismo se permite actuar sobre la planta haciendo uso de actuadores que pueden ser manipulados manualmente o ser programados a través de alarmas o secuencias de acciones.

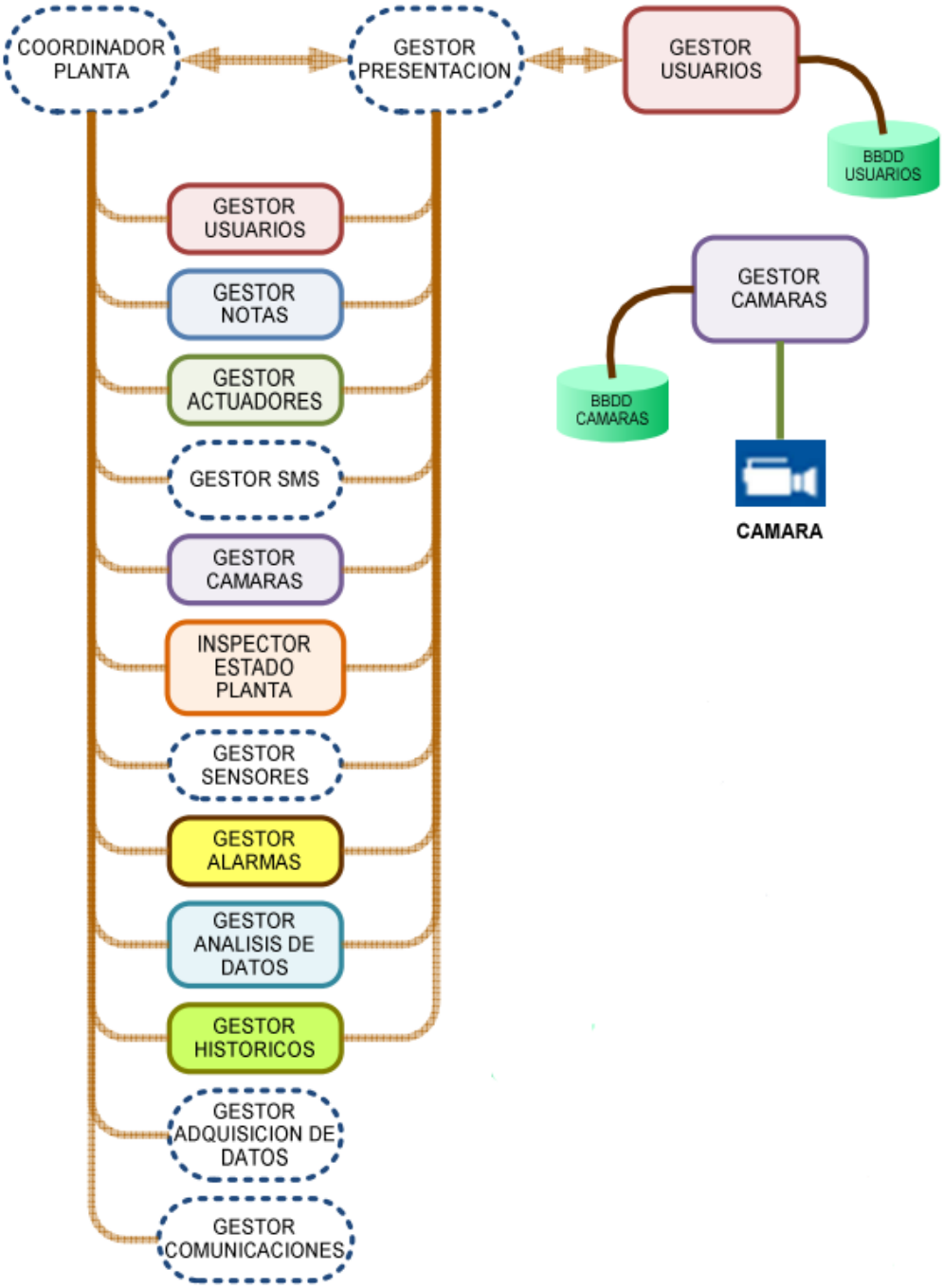

Ilustración 1: Estructura arquitectónica de *Hecaton* 1.

Los datos del estado de la instalación son procesados y almacenados para ser fácilmente accesibles a través de una interfaz web. El usuario tendrá la posibilidad de configurar libremente su instalación y la aplicación de manera que ésta se adapte a todo tipo de escenarios. Además disfrutará de la posibilidad de monitorización a través de la red lo cual ahorra costes de desplazamiento y contratación de personal técnico. Al ser posible actuar

utilizando la aplicación no será necesario estar presente físicamente para realizar el mantenimiento de la instalación o planta.

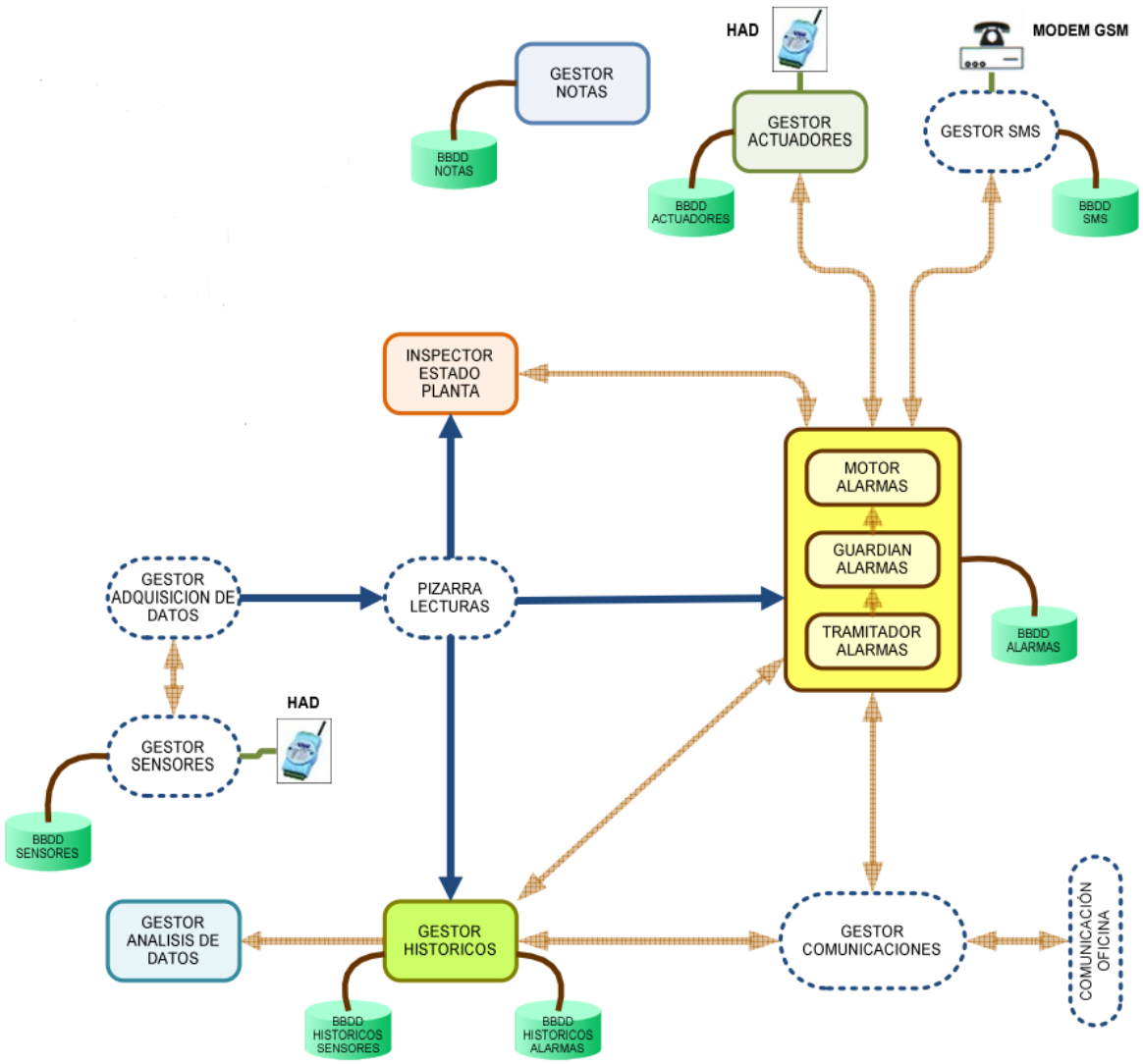

Ilustración 2: Estructura arquitectónica de *Hecaton 2*.

También se ha desarrollado un sistema de alertas que utiliza la red *GSM* para el envío de *SMS* que adviertan de alguna situación anómala previamente programada. Esta característica y la capacidad de monitorizar y actuar sobre toda la instalación pudiendo personalizar con gran detalle el funcionamiento del sistema muestran las virtudes y el potencial de este proyecto software. Hacer hincapié en el uso de herramientas de código abierto que reducen así mismo los costes de desarrollo ofreciendo una solución atractiva en lo económico para el cliente final.

La aplicación consta de los siguientes módulos:

- Módulo de usuarios
- Módulo de sensores
- Módulo de actuadores
- Módulo de alarmas
- Módulo de gestión de históricos (*logs*)
- Módulo de SMS
- Módulo de Secuencias
- *Applet* de visualización de históricos

1.4. Estado de la aplicación

Hecaton fue desarrollado utilizando la plataforma de desarrollo *J2EE* (*Java 2 Enterprise Edition*) y enteramente escrito en Java. Se utilizó el servidor de aplicaciones *JBoss SA* como servidor para contener la aplicación [1]. Como se indica en la literatura, se optó por la utilización de software libre que mantuviera los costes de desarrollo bajos ahorrando el precio de licencias de software propietario. Además Java nos ofrece independencia sobre la arquitectura subyacente.

Hecaton cuenta con las siguientes tecnologías. Indicaré también las versiones de las mismas:

- *JEE* versión de especificación 5 (Mayo de 2006) JSR244
    - *Enterprise application technologies*:
        - *Enterprise Java Beans (EJB) 3.0 JSR220*
        - *Java Persistence API (JPA) 1.0 JSR220*
        - *Common Annotations for the Java Platform 1.0 JSR250*
        - *Java Message Service API (JMS) 1.1 JSR914*
        - *Java Transaction API (JTA) 1.1 JSR907*
    - *Web application technologies*
        - *Java Servlet 3.0 JSR154*
        - *JavaServer Pages (JSP) 2.1 JSR245*
        - *JavaServer Pages Standard Tag Library (JSTL) 1.2 JSR52*

- Servidor de aplicaciones *JBoss 4.0.5 GA* (Octubre de 2006)
- Herramienta de desarrollo web *Jakarta Struts 1.3.5* (2006)
- Sistema de gestión de BBDD *MySQL*

Como se puede observar prácticamente las versiones del software que soportan la aplicación datan de más de 8 años y algunas de ellas han llegado al final de su vida (*EOL*) como *Struts 1* [4] que dejó de ser oficialmente soportado por *Apache* en abril de 2013.

Aunque es cierto que el software suele ser más fiable cuanto mayor tiempo lleve en funcionamiento, la tecnología no deja de avanzar. Las oportunidades de mejoras que pueden ser aplicadas hoy en día no tienen nada que ver con la informática de 2006. El software debe evolucionar para no morir fruto del avance tecnológico. A la hora de afrontar el análisis del proyecto fue incluso complicado encontrar documentación fiable que me permitiera comprender el código, fruto del uso de versiones tan antiguas.

Más allá de la tecnología usada, la aplicación en su última versión presentaba una organización caótica. La estructura del código fuente era difícilmente comprensible a simple vista lo que lleva a tiempos de desarrollo y corrección de errores muy altos. Por otro lado su interfaz luce exactamente igual que desde su primera versión, esto es, igual que cualquier sitio web de 2006.

1.5. Objetivos

Si bien en la propuesta de este proyecto se detallaron los objetivos del mismo, éstos han evolucionado durante la realización del mismo. El análisis detallado del código fuente y de las herramientas utilizadas no permite añadir más funcionalidad a la aplicación sin realizar una actualización y puesta al día en profundidad. Se trata de un hándicap común cuando se trabaja con un código ya existente donde el mantenimiento no ha sido cuidado en profundidad. Por ello y sin perder de vistas los objetivos originales se plantea refactorizar los mismos para que el resultado final sea una aplicación actualizada y estable. Además se pretende agilizar y facilitar

los posibles futuros desarrollos que partiendo de una aplicación base actual, puedan potenciar su funcionalidad y probar su funcionamiento en el mercado.

Los cambios realizados supondrán la actualización de parte del núcleo de la aplicación, desde el modelo de datos hasta el cambio de herramientas de desarrollo, si bien no será alterado el modelo de aplicación modular. Los objetivos se detallan en la siguiente lista:

1. Actualizar la aplicación y sus herramientas a las versiones estables actuales. La mayor parte de las herramientas de desarrollo usadas en este proyecto datan de 2006 o antes. Algunas no son ya mantenidas de manera oficial lo que deja el software desprotegido ante cualquier bug que se pudiera presentar. Además el hecho de que se utilicen herramientas antiguas limita las posibilidades de mejora y expansión ya que no se aprovechan las innovaciones tecnológicas aparecidas durante estos años. Por esta razón es necesario en primer lugar actualizar el servidor de aplicaciones *JBoss*, lo cual obliga a actualizar el código a la versión 6 de la especificación de JEE. Este paso conlleva una refactorización del código de gran calado pero necesaria. Así mismo es de urgente necesidad actualizar el *framework* web *Struts 1* ya que su desarrollo se detuvo en 2008 y el software ha llegado al final de su vida.

2. Introducción de una herramienta de gestión y construcción de proyectos. En nuestro caso al tratarse de una aplicación desarrollada en java utilizaremos *Maven*. El objetivo perseguido es poner orden a la caótica organización del código fuente, así como de sus dependencias. Además se agiliza y automatiza el proceso de compilación, enlace y despliegue de la aplicación. Hasta ahora todas estas tareas se han desarrollado de manera manual, lo cual dificulta y ralentiza de manera enorme el proceso de desarrollo para aplicaciones *J2EE*. También se quiere dar respuesta a la gestión de las dependencias, tarea que en la versión actual se realiza añadiendo las librerías al proyecto de manera manual. Con vistas al futuro, una herramienta como *Maven* se hace imprescindible para el desarrollo de software actual donde el testeo y la integración continua del desarrollo del producto marcan tendencia y casos de éxito.

3. Realizar una actualización de la interfaz web desde la cual se gestiona la aplicación. Se pueden observan enormes posibilidades de mejora con la aplicación de tecnologías web actuales como *AJAX* o librerías como *JQUERY* que permite que el diseño de la

interface sea acorde con las características dinámicas de las aplicaciones creadas con las tecnologías actuales de desarrollo web. Por lo tanto un proceso de re-ingeniería de la vista del sistema se hace más que necesaria persiguiendo su mejora, actualización y potenciando el atractivo de la aplicación para su potencial uso para clientes en entornos reales y productivos.

4. Potenciar la idea misma de telecontrol creando una interfaz para dispositivos móviles como son teléfonos móviles o tabletas. El acceso a redes a través de los sistemas de telefonía móvil ha supuesto un boom en los últimos años y la idea de poder gestionar una pequeña instalación industrial mediante estos dispositivos genera un gran atractivo para los potenciales usuarios. Se pretende además actualizar el viejo sistema de alertas *SMS*, el cual ha quedado obsoleto ante el empuje de la banda ancha móvil. Las potenciales funcionalidades que ofrecen los actuales dispositivos móviles abren un gran marco de posibilidades.

5. Finalizar el desarrollo de la tele vigilancia mediante cámaras web de las instalaciones. Este módulo se encuentra ya ideado en la aplicación pero no ha sido completamente desarrollado. Con la mejora del interface se pretende implementar esta funcionalidad e incluirla en el catálogo de servicios que ya ofrece la aplicación.

6. Uno de los puntos más importantes de este proyecto es la instalación y prueba, en un entorno real de la aplicación software para validar su viabilidad durante un largo periodo de uso. Se pretende realizar un test en una pequeña instalación donde podamos ver el sistema funcionando y así poder depurarlo en un escenario real, además de auditar y documentar el funcionamiento del mismo. La idea es presentar un informe donde se refleje la viabilidad de la aplicación y se compruebe que los requisitos originales de la aplicación se cumplen.

1.6. Metodología

Como se desprende de los párrafos anteriores este proyecto de fin de carrera es un ejercicio de refactorización software. A diferencia de mis compañeros, mi objetivo principal no ha sido añadir funcionalidad sino optimizar y actualizar la aplicación respetando la funcionalidad original. La refactorización software aparece dentro de muchas metodologías software y especialmente en las más actuales y exitosas como es la Programación Extrema.

1.6.1. Refactorización

En realidad la refactorización software se realiza desde que se conoce la programación. Es un proceso natural en el desarrollo de tecnología e innovación. Responde a la pregunta: Bien, funciona. Ahora ¿Cómo podemos hacer lo mismo mejor? El libro de Martin Fowler "Refactoring" [5] es una referencia clásica. Una de las definiciones más concisas sobre el concepto puede encontrarse en el mismo: "Es una disciplina técnica para la restructuración en un bloque de código ya existente, alterando su estructura interna pero sin cambiar su comportamiento externo."

Aunque la refactorización de código se ha llevado a cabo de manera informal durante años, la tesis doctoral de William F. Opdyke [24] es el primer trabajo conocido que examina específicamente esta técnica. Todos estos recursos proporcionan un catálogo de métodos habituales de refactorización. Un método de refactorización tiene una descripción de cómo aplicar el método e indicaciones sobre cuándo debería (o no debería) aplicarse. La refactorización es un concepto tan importante que ha sido identificado por David A. Wheeler como "una de las más importantes innovaciones en el campo del software" [6].

Ilustración 3: Evolución como ejemplo de refactorización.

Usualmente, la refactorización aplica una serie de pequeñas refactorizaciones estandarizadas, cada una de las cuales cambia el código fuente de un programa preservando el comportamiento del software y además respetando los requerimientos funcionales.

La refactorización esta normalmente motivada por ciertos indicadores llamados en inglés "*Code smell*". Son síntomas de que el código fuente de un programa pueda contener problemas más profundos. Normalmente no se trata de errores o "bugs". No son técnicamente incorrectos y no hacen que el software no funcione. Sin embargo indican debilidades en el diseño que pueden ralentizar el desarrollo o incrementar el riesgo de fallos en el futuro. Un ejemplo de esto podría ser un método que es demasiado largo o está duplicado. Una vez reconocidos estos problemas, pueden ser solventados mediante la refactorización del código fuente o transformados para eliminar tales indicadores de debilidades. En nuestro ejemplo, una función larga puede ser reescrita como una o más subrutinas. La duplicidad de código puede ser eliminada y generalizada para que pueda ser compartida.

A continuación mostramos una lista de estos indicadores o "*Code smells*" [7]:

- Código duplicado
- Métodos demasiado largos
- Clases demasiado largas
- Demasiados parámetros
- Clases que usan demasiados métodos de otra clase

- Clases que dependen de los detalles de implementación de otras clases
- Clases que sobrescriben métodos de las clases que derivan
- Clase “vaga” o clase que hace muy poco
- Identificadores demasiado largos
- Identificadores demasiado cortos
- Uso excesivo de literales
- *Callbacks* que ejecutan demasiadas acciones
- Estructuras condicionales complejas

Estos conceptos son también conocidos en muchos casos como anti-patrones de diseño y muy utilizados por programadores que utilizan técnicas ágiles.

Existen dos categorías generales de beneficios de aplicar refactorización:

1. Mantenibilidad.
2. Extensibilidad

Resulta más fácil corregir un error si el código fuente es de lectura sencilla donde extraer la intención del programador es fácil de captar. Por otra parte si un código se encuentra bien estructurado es más sencillo extender sus capacidades.

A continuación se ofrece una lista ilustrativa de algunas de las técnicas de refactorización más comunes. Algunas son aplicables a algunos lenguajes o tipos de lenguajes y no a otros. En la bibliografía [5] podemos encontrar listas más completas y desarrolladas de estas.

- Técnicas que permiten mayor abstracción
    - Encapsulación de campo - fuerza el acceso a un campo mediante métodos “*set*” y “*get*”
    - Generalizar tipos
    - Reemplazar comprobación de tipo con el patrón Estado o Estrategia.
    - Reemplazar condiciones con polimorfismo
    -
- Técnicas para la separación de código en fragmentos más pequeños

    - Separación de métodos en componentes
    - Extraer nuevas clases de otras más largas
    - Extraer métodos de otros más largos

- Técnicas para la mejora de nombres y localización de código
    - Mover métodos o campos - Por ejemplo a otras clases si es más apropiados
    - Renombrar métodos o campos
    - Mover métodos o campos a super clase
    - Mover a subclases.

Una de las primeras tareas es la creación de test automáticos que permitan garantizar que una sección de código que se desea refactorizar conserve su funcionamiento original. Lamentablemente en este proyecto no se ha introducido el uso de test. La tarea de desarrollo de test para este proyecto debido a su extensión sería demasiado grande. Llevaría un proyecto de fin de carrera entero. Además un entorno de test sólido no sólo se basa en escribir test unitarios. Es necesario automatizarlos e incluirlos en el proceso de compilación y construcción. A día de hoy tal proceso es manual en el estado actual de la aplicación. También es conveniente pasar a una metodología de integración continua que garantice que cada cambio realizado en el código no ha alterado el funcionamiento del mismo ni generado errores. Las herramientas utilizadas hasta ahora en este proyecto no ayudan a tal tarea. Uno de mis objetivos ha sido actualizar la aplicación sin perder de vista la introducción de estas técnicas. Pero en lo referente a la validación de la funcionalidad, usaré la funcionalidad actual, es decir, el software actualizado debe realizar las mismas tareas y acciones que el anterior y cumplir los casos de uso de los anteriores proyectos.

Por otro lado no estamos tratando en este proyecto solamente de refactorizar el código fuente. También queremos actualizar las herramientas de desarrollo y los marcos de trabajo, así como desarrollar una nueva interfaz. Lo que es cierto es que se han tenido en cuenta los principios de la refactorización software para la realización de este proyecto. Allí donde ha sido posible y necesario se ha refactorizado el código. También se incluirán en este texto aquellos casos en los que si bien no se ha refactorizado el código, si se han detectado posibilidades de mejoras. Es importante incluir en la sección de trabajos futuros la descripción de tales tareas para que un nuevo ciclo de desarrollo de la aplicación pueda abordarlos.

1.6.2 Desarrollo iterativo e incremental.

Las fases de desarrollo de este proyecto de fin de carrera han seguido un enfoque de desarrollo iterativo y creciente. Si bien en un primer momento se identificó la interfaz de la aplicación como la piedra central del proyecto, durante el análisis, diseño e implementación de esta tarea se han encontrado muchas otras tareas cruciales que se han ido añadiendo como iteraciones del desarrollo.

Ilustración 4: Desarrollo iterativo e incremental.

La idea principal detrás de la mejora iterativa es desarrollar un sistema de programas de manera incremental, permitiéndole al desarrollador sacar ventaja de lo que se ha aprendido a lo largo del desarrollo anterior, incrementando, versiones entregables del sistema. El aprendizaje viene de dos vertientes: el desarrollo del sistema, y su uso (mientras sea posible). Los pasos claves en el proceso son comenzar con una implementación simple de los requerimientos del sistema, e iterativamente mejorar la secuencia evolutiva de versiones hasta que el sistema completo esté implementado. En cada iteración, se realizan cambios en el diseño y se agregan nuevas funcionalidades y capacidades al sistema. También se prueban y se valida cada iteración mediante la realización de test y presentación al cliente. Si bien en este caso, este paso será llevado a cabo por mí mismo [8].

Básicamente este modelo se basa en dos premisas:

- Los usuarios nunca saben bien que es lo que necesitan para satisfacer sus necesidades.
- En el desarrollo, los procesos tienden a cambiar.

Ventajas del desarrollo incremental:

- En este modelo los usuarios no tienen que esperar hasta que el sistema completo se entregue para hacer uso de él. El primer incremento cumple los requerimientos más importantes de tal forma que pueden utilizar el software al instante.
- Los usuarios pueden utilizar los incrementos iniciales como prototipos y obtener experiencia sobre los requerimientos de los incrementos posteriores del sistema.
- Existen muy pocas probabilidades de riesgo en el sistema. Aunque se pueden encontrar problemas en algunos incrementos, lo normal es que el sistema se entregue sin inconvenientes al usuario.
- Puesto que los servicios de más alta prioridad se entregan primero, y los incrementos posteriores se integran en ellos, es inevitable que los servicios más importantes sean a los que se les hagan más pruebas. Esto significa que es menos probable que los usuarios encuentren errores de funcionamiento del software en las partes más importantes del sistema.

Ventajas del desarrollo iterativo:

- En el desarrollo de este modelo se da la retroalimentación muy temprano a los usuarios.
- Permite separar la complejidad del proyecto, gracias a su desarrollo por parte de cada iteración o bloque.
- El producto es consistente y puntual en el desarrollo.
- Los productos desarrollados con este modelo tienen una menor probabilidad de fallar.
- Se obtiene un aprendizaje en cada iteración que es aplicado en el desarrollo del producto y aumenta las experiencias para próximos proyectos.

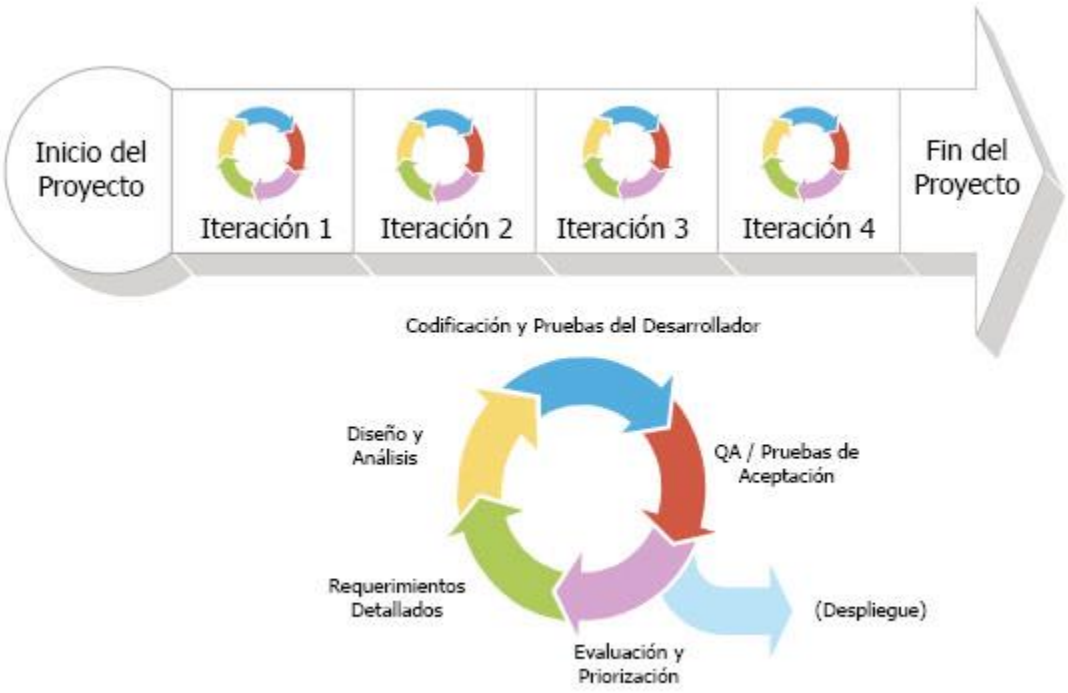

Ilustración 5: Esquema del modelo de desarrollo iterativo e incremental

Debilidades de este modelo de desarrollo:

- La entrega temprana de los proyectos produce la creación de sistemas demasiados simples que a veces se ven un poco monótonos a los ojos del personal que lo recibe.
- La mayoría de los incrementos se harán en base de las necesidades de los usuarios. Los incrementos en sí ya son estipulados desde antes de la entrega del proyecto, sin embargo hay que ver cómo se maneja el producto para ver si necesita otros cambios además de los estipulados antes de la entrega del proyecto.
- Los incrementos no deben constar de muchas líneas de código ya que la idea de los incrementos es agregar accesorios al programa principal (o funcional), para que este tenga una y mil formas de desenvolverse en su tarea; llenar los incrementos de muchas líneas de código provocaría que se perdiera la objetividad o base de lo que se trata el desarrollo incremental.
- Requiere de un cliente involucrado durante todo el curso del proyecto.
- El trato con el cliente debe basarse en principios éticos y colaboración mutua, más que trabajar cada parte independientemente, defendiendo sólo su propio beneficio.

- La entrega de un programa que es parcial pero funcional puede hacer vulnerable al programa debido a la falta de robustez en su sistema, provocando que agentes ajenos puedan interferir con el correcto funcionamiento del programa en sí.
- Infunde responsabilidad en el equipo de desarrollo al trabajar directamente con el cliente, requiriendo de profesionales sobre el promedio.
- Sufre fuertes penalizaciones en proyectos en los cuales los requerimientos están previamente definidos, o para proyectos "todo/nada" en los cuales se requiere que se completen en un 100% el producto para ser implementado (por ejemplo, licitaciones). Otro punto muy importante es asegurarnos de que el trabajo se pueda cumplir tomando en cuenta los costos que podamos usar en nuestros propios recursos.

2. Análisis

2.1. Análisis de la aplicación

Después de observar la aplicación, ver su funcionamiento y analizar tanto el código, como las herramientas utilizadas y la documentación es hora de pensar cómo es posible mejorarla. Partimos de la base de una aplicación que funciona, con un núcleo de funcionamiento bien diseñado y programado. Tenemos una muy amplia funcionalidad y un diseño modular que permite su ampliación. Pero la aplicación no ha sido actualizada desde su primera versión. Aunque se ha añadido funcionalidad de manera exitosa, el código y si dificultad ha aumentado sin control.

En primer lugar considero que se debe cambiar la organización del código fuente. Los archivos fuentes recibidos desde el último proyecto tienen una organización caótica y es muy complicado entender a qué parte de la aplicación pertenece cada cosa. En el código fuente he encontrado muchos ficheros, clases y librerías que no forman parte del código o pertenecen a antiguos test. Creo que la organización del código fuente es indispensable para obtener software de calidad que sea fácil de mantener y ampliar. También es conveniente introducir alguna herramienta de control de las librerías usadas. A día de hoy ni siquiera existe una carpeta donde estén recogidas todas las librerías sino que se encuentran desperdigadas por el proyecto. Además es necesario añadirlos manualmente a las rutas del proyecto para utilizarlas.

Por otro lado el proceso compilación y despliegue es tedioso, lento e ineficiente. Descrito por el anterior alumno Andrés del Pino Bolaños, es necesario compilar el código, generar los ficheros comprimidos a mano, copiarlos en el servidor de aplicaciones y finalmente arrancar el mismo para comprobar cada cambio realizado en el código. En necesario mejorar este proceso para acelerar el desarrollo del software.

La actualización de la interfaz es uno de los objetivos principales de este proyecto de fin de carrera. Las tecnologías web han avanzado enormemente en los últimos años, y es

recomendable aprovechar estos avances para mejorar la interfaz de la aplicación. Creo que es necesario un cambio en profundidad en el diseño y en la organización del sitio web. Es necesario analizar los casos de uso que generaron la funcionalidad de la interfaz y realizar una tarea de re-ingeniería. Nos es necesario cambiar la funcionalidad sino reorganizarla y unificarla. También es importante utilizar herramientas actuales que permitan dar un paso hacia los dispositivos móviles. Necesitamos introducir un diseño web que sea "*Responsive design*" (concepto complicado de traducir al castellano). Es decir, que nuestro sitio web se adapte a todo tipo de dispositivos y pantallas para que pueda ser utilizado por dispositivos tales como teléfonos móviles o tabletas digitales. Es muy interesante introducir estas mejoras ya que permitirán tener el control sobre la instalación en todo momento y de manera interactiva.

Un aspecto ya comentado es la necesidad de actualizar el software utilizado. Desde el servidor de aplicaciones hasta el *framework Struts* utilizado para la vista están obsoletos, al final de su vida y sin mantenimiento oficial que proteja de posibles fallos. Ocurre lo mismo con la librería que utiliza el módem *GSM*.

Por último quiero añadir a este apartado detalles que han aparecido durante la realización de este proyecto y que he considerado indispensable corregir, si bien no estaban contemplados en el análisis inicial. Se han detectado errores de diseño en el modelo de datos. En algunas entidades no se ha usado ningún tipo de identificador numérico. Si bien en algunas entidades se ha optado por claves compuestas que generan un identificador numérico a través de un código hash, en otros se ha optado por ristras de caracteres, nombres de los objetos reales, como claves primarias. En ocasiones una combinación de nombres ha sido empleada como clave. Si bien es conveniente para el entendimiento del modelo de datos, su manipulación en él es tediosa e ineficiente. Sí nuevas relaciones emergen durante el uso del software, la asociación de tales claves con otros objetos es complicada frente a la sencillez de claves numéricas. Por otro lado el uso de relaciones uno a muchos se ha sido usado erróneamente. Se han manipulados objetos ya existente añadiendo claves dentro de los mismo en lugar de relaciones. Como ejemplo podemos citar las secciones. La pertenencia a una sección de un sensor está indicada por un campo sección dentro del sensor. Tal campo debe ser inicializado como nulo si el sensor no pertenece a una sección y debe ser manipulado aunque en la mayor parte de los casos la sección es irrelevante.

Es también necesario la separación y definición de un módulo para los equipos de adquisición de datos. Actualmente toda la lógica de negocio relacionada con estos se encuentra embebida dentro de la gestión de sensores, lo cual no tiene sentido. Esta decisión se tomó en la primera versión del proyecto y se arrastra desde entonces. Es necesario que los dispositivos de adquisición de datos sean tratados como elementos independientes y completos como lo son lo sensores o los actuadores. Actualmente se ofrece funcionamiento para un solo tipo de hardware de adquisición de datos pero resulta lógico que esto se amplíe. Si no tenemos definido un módulo independiente para el soporte de este hardware, cualquier cambio o ampliación resultará complicada. Separando los módulos de adquisición de los sensores obtendremos las ventajas ya descritas.

2.2. Plan de actuación

Si bien en el apartado anterior se ha intentado exponer qué cambios son necesarios y el porqué de cada uno de ellos, es hora de definir de manera concreta qué cambios se han realizado, cómo y qué herramientas se han usado. También es importante aclarar que acciones, sobre todo de refactorización, se han llevado a cabo durante cada iteración

2.2.1. Fase 1

El primer problema a resolver es la organización del código. Es crucial que la estructura de la aplicación esté definida y sus fuentes representen el diseño modular de la misma y sigan las especificaciones *Java EE*. Para ello se introducirá la herramienta "*Maven*". *Maven* es una herramienta para la gestión y construcción de proyectos en java. El uso de arquetipos o prototipos (del inglés "*Archetypes*", plantillas de proyectos) persigue la estandarización de proyectos *Java* que permitirá organizar los fuentes de nuestras aplicaciones de una manera reconocida, documentada y probada por toda una comunidad de desarrolladores. *Maven* basa su configuración en ficheros *xml* para definir tareas como la compilación del ciclo de vida del desarrollo. Además, *Maven* está preparada para trabajar en red y posee una potente utilidad de gestión de dependencias (librerías). De esta manera resolvemos otro de los problemas encontrados durante el análisis. *Maven* es casi la herramienta "de facto" para el desarrollo de proyectos en *Java* [9]. Posteriormente se ofrecerá al lector una descripción más detallada de *Maven* así como del resto de las herramientas software aquí presentadas.

2.2.2. Fase 2

En segundo lugar se pretende actualizar el servidor de aplicaciones *JBoss* de su actual versión, la 4.0.5, a su última versión estable, la 7.1.1. Entre las ventajas de esta decisión se encuentra que el servidor en su versión 7 se encuentra completamente certificado para la especificación 6 de *Java EE*, además de una gran mejora en la gestión de memoria. Aunque un cambio de versiones tan amplio supondrá muchos cambios a nivel de código. Estos cambios son en su mayoría motivados por cambio en las especificaciones como la actualización de los "EJB" a su versión 3.1, el cambio en el sistema de mensajería en *JBoss* al proyecto "*HornetQ*" o del sistema caché a "*Infinispan*", entre otros. Estos requerimientos se abordarán en profundidad más adelante.

2.2.3. Fase 3

Uno de los cambios más profundos tiene que ver con el cambio en la interfaz. Después de dejar sentado que no es posible continuar con *Struts 1* se ha decidido actualizar a *Struts 2*. *Struts* es un *framework* de desarrollo web basado en el modelo MVC (Modelo-Vista-Controlador). Entre las ventajas de esta decisión encontramos: el manejo de acciones como hilos, la dependencia de los *servlets* (las acciones no están acopladas al contenedor), soporte para la inyección de dependencias, la introducción de los interceptores o el soporte para web dinámica con *AJAX* [4] Todo ello acompañado con su diseño basado en *plugins* que nos permitirá utilizar por ejemplo tecnologías como objetos JSON para el envío de información al usuario, lo que posteriormente nos permitirá fácilmente construir aplicaciones móviles reutilizando todo el código existente.

Si bien hemos enumerado muchas de las ventajas de la actualización a *Struts 2*, existen inconvenientes. *Struts 2* no es una actualización de *Struts 1*, es un proyecto separado, procedente del *framework WebWork 2*. Por lo tanto son incompatibles. Se requiere un cambio profundo en todo el código para llevar a cabo tal actualización. Llegaría a ser de un coste similar cambiar a cualquier otro *framework* de desarrollo web para Java como *JavaServer Faces* o *Maverick*. Sin embargo requerimos de un rediseño de la vista como se ha mencionado anteriormente. Muchos de los casos de uso serán renovados y por lo tanto cambirán. Debido a esto, la actualización del *framework* para la vista de la aplicación aparte del tiempo extra

necesario para familiarización con el *Struts 2* requerirá del rediseño de muchas de las clases controladoras (Acciones). Apache, la organización que soporta el proyecto *Struts* (así como *Maven*), ofrece abundante documentación que facilita el paso del antiguo *Struts 1* a su versión 2.

También se procederá en esta fase a la creación del módulo para los dispositivos de adquisición de datos, separándolo del módulo de sensores. Al mismo tiempo se cambiarán los aspecto relacionados con el modelo de datos descritos anteriormente, así como las relaciones uno a muchos y los identificadores.

2.2.4. Fase 4

En este paso se pretende definir el estilo final de la aplicación. Utilizando *Bootstrap* se buscará un diseño apropiado que unifique la interface. Conviene adaptarse a un diseño estándar que no introduzca demasiadas modificaciones o "*hacks*" a través de *CSS*. Esto puede entorpecer enormemente un cambio de diseño a posteriori. Se busca obtener una interfaz limpia, basada en colores suaves y agradables a la vista.

También es necesario cambiar el tipo de dato de mapeo de las ristras de caracteres en la base de datos para las alarmas y secuencias. Éstas están definidas como ristras de hasta 256 caracteres. Si las reglas o pasos son extensos (y no necesariamente demasiado), no pueden ser añadidas.

Por último se procederá a la actualización de la librería *SmsLib* lo cual nos permite utilizar otro controlador de Java para el puerto serie como *RxTx*. Hasta hoy se viene utilizando la versión 2.1.2 de la librería que solo admite *JavaComm* como driver. Como consecuencia de ello el módem sólo puede ser utilizado por la aplicación en sistemas Windows con arquitectura x86.

2.3 Descripción de las tecnologías utilizadas.

2.3.1 Plataforma *Java Enterprise Edition*.

La plataforma *Java Enterprise Edition* (*Java EE*) es un conjunto de especificaciones que facilitan el desarrollo y despliegue de aplicaciones empresariales multicapa. *Java EE* ofrece un conjunto de especificaciones y técnicas que proporcionan soluciones completas, seguras, estables y escalables para el desarrollo, despliegue y gestión de aplicaciones de múltiples niveles de funcionalidad basadas en servidores. Se reduce el costo y complejidad de desarrollo, lo cual resulta en servicios que se pueden desplegar y extender fácilmente [12].

Los componentes *Java EE* se despliegan y son administrados por contenedores conocidos como servidores de aplicaciones. También se explorará y utilizará los servicios estándar proporcionados por dichos servidores para el despliegue de aplicaciones *JEE*.

Ilustración 6: Conocido logotipo de *Java* actualizado por su nuevo propietario

El modelo de aplicación *Java EE* nace con el lenguaje de programación java y la máquina virtual *Java*. La portabilidad, seguridad y productividad que estas proveen forman la base de este modelo de aplicación.

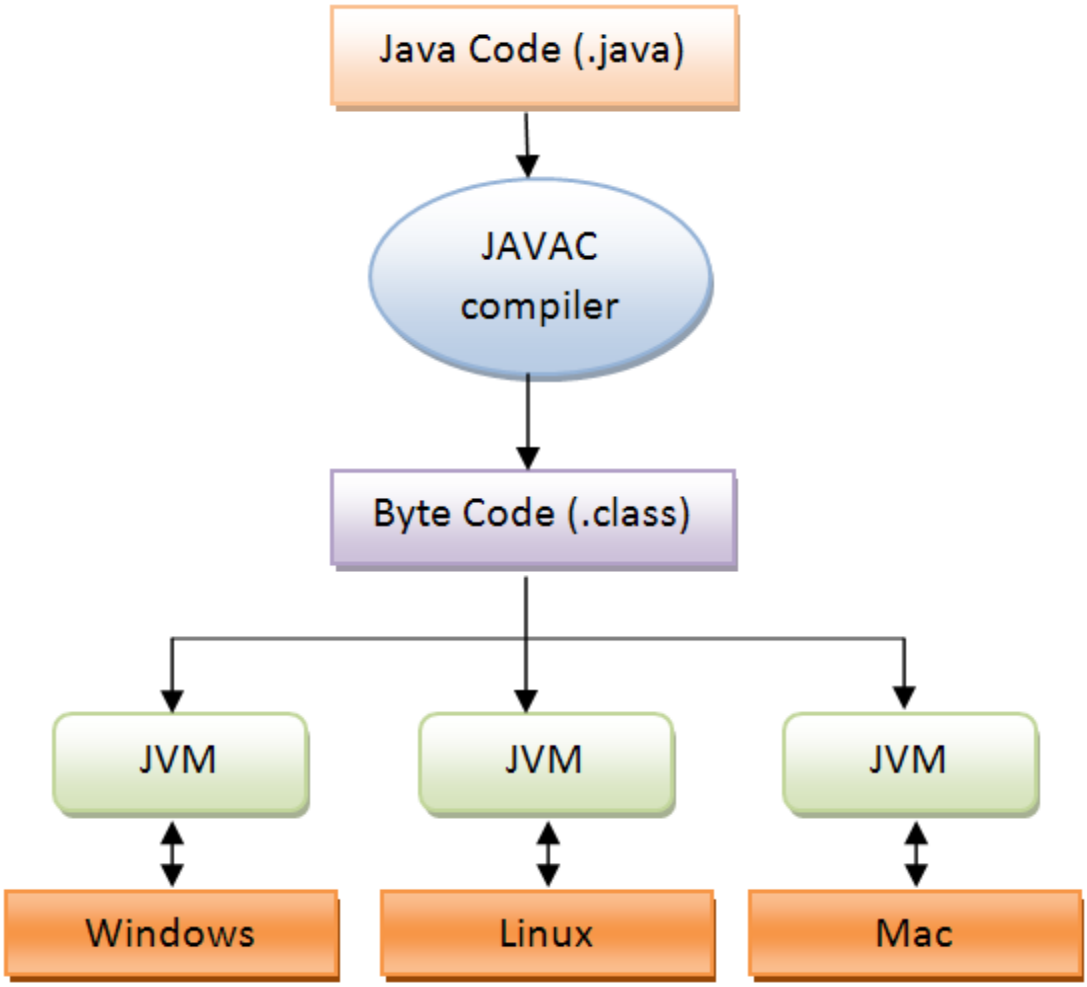

Ilustración 7: Modelo de compilación de *Java*

Cabe destacar que *Java EE* no es una herramienta o lenguaje de programación en sí. Se trata como hemos dicho de un conjunto de especificaciones que permiten soluciones para el desarrollo, despliegue y gestión de aplicaciones multicapa centradas en servidor. ¿Qué quiere decir esto? Esto quiere decir que cualquier fabricante puede tomar todas o varias de estas especificaciones y desarrollar un producto final que las cumpla. *Sun* creó una implementación de todas estas especificaciones (*Glashfish*) pero no es la única. En este proyecto se trabaja con *JBoss* que no es más que una implementación distinta que cumple estas especificaciones.

Tecnologías más destacadas de la plataforma *Java EE* en su versión 7 [11]:

- ***Enterprise JavaBeans (EJB 3.1)***

Se trata de cuerpos de código que poseen campos y métodos que implementan módulos de la lógica de negocio. Se puede pensar en un "*Enterprise Bean*" como un bloque de edificios que puede ser usado solo o con otros para ejecutar la lógica de negocio en un servidor de aplicaciones *Java EE*. Los *EJB* se clasifican como de sesión o como orientados a mensajes [18]:

- Un *bean* de sesión representa el transcurso de una conversación con un cliente. Cuando el cliente finaliza la ejecución de una tarea, la sesión y sus datas desaparecen.
- Un "*message-driven bean*" combina características de un *bean* de sesión y un interlocutor de mensajes o "*message-listener*". Permite a un componente de negocio recibir mensajes de manera asíncrona.

- ***JavaSever Pages (JSP 2.2)***

JavaServer Pages permite introducir fragmentos de código de *servlets* directamente como documentos de texto. Una página *jsp* es un fichero de texto que contiene 2 tipos de texto:

- Datos estáticos que pueden ser expresados como cualquier otro formato basado en texto como *HTML* o *XML*.
- Elementos *JSP* que determinan cómo se debe construir el contenido dinámico de la página.

- ***JavaServer Pages Standard Tag Library (JSTL 1.2)***

La tecnología *JSTL* encapsula funciones comunes a toda aplicación *JSP*. En lugar de mezclar etiquetas ("*tags*") de diferentes proveedores de aplicaciones *JSP*, una única y estándar colección de etiquetas es usada. Tal estandarización permite desplegar aplicaciones *JSP* en cualquier contenedor de aplicaciones que soporte *JSTL*. Además la implementación de estas etiquetas está optimizada. En otras palabras, las etiquetas representan pedazos de código JSP que ayudan a reducir y comprender las páginas *JSP*.

JSTL posee etiquetas de control e iteradores que alteran el control de flujo, etiquetas para la manipulación de ficheros *XML*, internacionalización, acceso a base de datos y muchas otras funciones comunes.

- ***Java Persistence API (JPA 2.0)***

La *API* de persistencia permite salvar la diferencia entre los concepto de modelos orientados a objetos y las bases de datos relacionales basadas en tablas. Los objetos y los datos que estos contienen solamente existen mientras se ejecutan. La persistencia en un entorno orientado a objetos, se refiere a la capacidad de dichos objetos de mantener sus datos entre sucesivas ejecuciones de un programa. Los *EJB* de entidad dan soporte a la persistencia gracias al contenedor de *Enterprise Java Beans*. Su uso se basa en el uso de anotaciones sobre clases java planas o normales (conocidas como *POJO*). *JPA* transforma el mundo de los objetos al mundo de las claves primarias y secundarias mapeando los objetos en la base de datos correspondiente. Utiliza posteriormente el conector Java de bases de datos (*JDBC*) correspondiente para lanzar las consultas ("*Queries*") correspondientes [19].

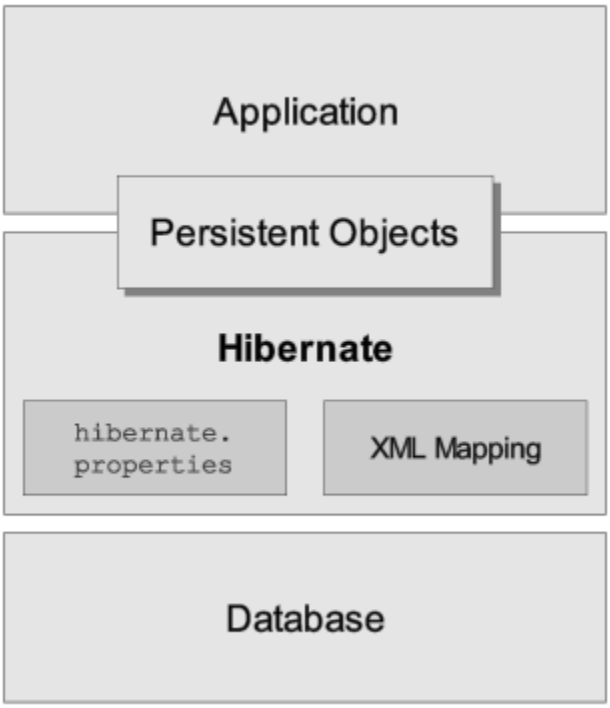

Ilustración 8: Esquema de funcionamiento de *Hibernate*

Existen diferentes implementaciones de esta especificación (*Hibernate*, *topLink*, *eclipseLink*, etc). En este proyecto se usa y se seguirá usando *Hibernate*. *Hibernate* es una herramienta de Mapeo objeto-relacional (*ORM*) desarrollada por *RedHat* que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (*XML*) o anotaciones en los *Beans* de las entidades que permiten establecer estas relaciones.

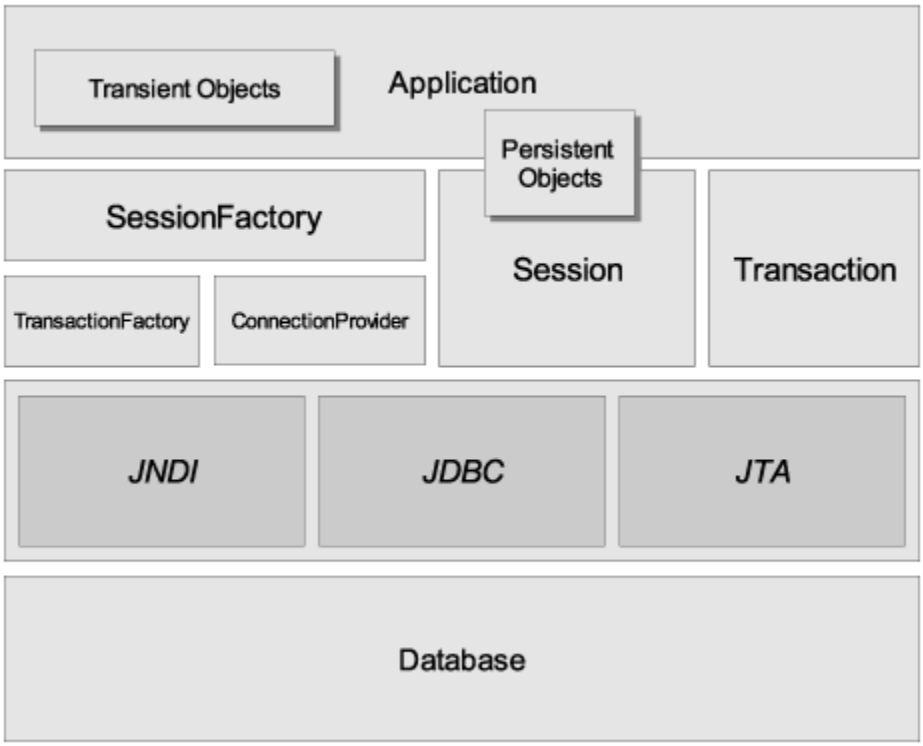

Ilustración 9: Modelo de persistencia *JPA*

- ***Java Transaction API (JTA 1.1)***

La *API Java Transaction* provee una interfaz estándar para demarcar las transacciones. *JTA* se asegura de la integridad de los datos y su validez cuando nos encontramos en situaciones en que se quiere asegurar su unicidad en operaciones de guardado, actualización o eliminación. Permite asegurar la atomicidad de una o varias instrucciones que trabajan con datos como por ejemplo las operaciones sobre las bases de datos.

- ***Contexts and Dependency Injection for the Java EE Plataform. (CDI 1.0)***

  CDI define un conjunto de servicios contextuales, que hacen fácil para los desarrolladores el uso de los *Beans* entre las diferentes capas de la aplicación. *CDI* también tiene otros usos como el de permitir a los desarrolladores gran flexibilidad para integrar diferentes tipos de componentes en un acoplamiento flexible pero que asegura los tipos de datos.

  Explicado de forma más sencilla, *CDI* permite el uso de diferentes clases dentro de un contenedor de aplicaciones sin la necesidad de indicar su localización exacta dentro del mismo ("*lookup*") y garantizando que el tipo de datos usado sea el correcto.

- ***Dependency Injection for Java (DI 1.0)***

  Dependency Injection for Java define un conjunto estándar de anotaciones y una interfaz para el uso de clases inyectables.

- ***Java Message Service API (JMS 1.1)***

  El servicio de mensajería *Java* permite a los componentes de una aplicación *Java EE* crear, enviar, recibir y leer mensajes. Permite una comunicación distribuida de bajo acoplamiento, fiable y asíncrona [20].

- ***Java Naming and Directory Interface API (JDNI 1.2)***

  Provee funcionalidad de nombres y directorios, permitiendo a las aplicaciones acceso a múltiples servicios de nombres como *LDAP*, *DNS* o *NIS*. La *API* "*JNDI*" Provee métodos estándar para ejecutar operaciones sobre directorios de nombres tales como la asociación de atributos con objetos y la búsqueda de objetos usando sus atributos. Usando *JDNI* una aplicación *Java EE* puede guardar y recuperar todo tipo de objetos *Java*, permitiendo a la aplicación coexistir con muchos tipos de aplicaciones y sistemas.

JDNI provee a las aplicaciones cliente, los *EJB* y a los componentes web acceso al contexto de nombres *JDNI*. Este permite a un componente ser personalizado sin la necesidad de acceder o cambiar el código fuente del componente.

- ***Java Database Connectivity API (JDBC 4.0)***

  El *API* de conectividad con base de datos java permite invocar sentencias *SQL* desde programas *Java*. Esta es utilizada siempre que se accede a la base de datos dentro de las aplicaciones *Java EE*.

Existen muchas otras especificaciones en *Java EE* de las cuales unas son usadas en este proyecto y otras no. En este resumen se pretende dar una visión de aquellas más relevantes para el software que tenemos entre manos.

Hemos dicho que la plataforma *Java EE* está destinada a desarrollar aplicaciones distribuidas con una arquitectura multicapa. Esto quiere decir que podemos separar el desarrollo de la aplicación en diferentes capas según su función. Las aplicaciones *Java EE* suelen ser consideradas aplicaciones de tres capas porque se distribuyen en tres localizaciones, ordenadores clientes, el sistema donde se ejecuta el servidor de aplicaciones, y el sistema donde reside la base de datos.

- **La capa del cliente (*Client-tier*)** que es la capa destinada a mostrar la interfaz gráfica de usuario. Las aplicaciones *Java EE* pueden ser una aplicación Java Swing normal, o una aplicación Web renderizada en un navegador. Esta capa se ejecuta en el ordenador cliente.

- **La capa de la lógica de negocio (*Business-tier*)** y la capa de la lógica de presentación (*Web-tier*). Estas capas se ejecutan en el servidor de aplicaciones.

- **La capa de los datos (*Data-tier*)** que es la capa destinada a la gestión de los datos. Esta capa puede separarse a su vez en una o más capas.

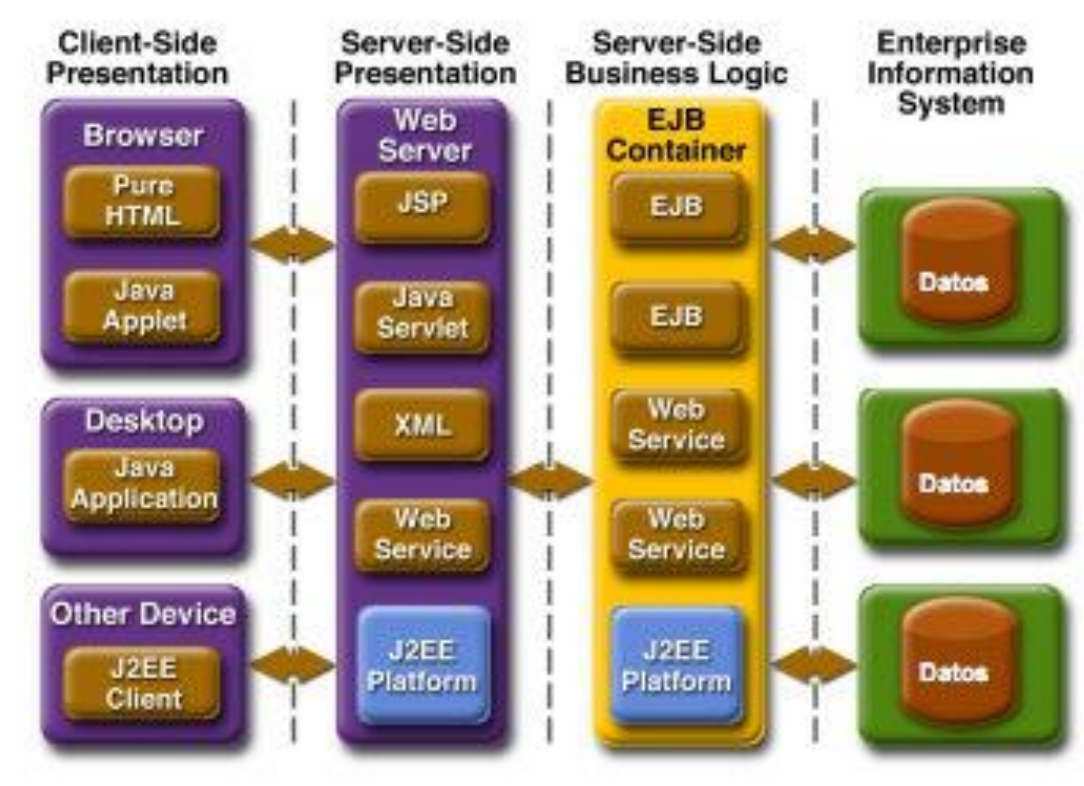

Ilustración 10: Modelo multicapa *Java EE*

Las aplicaciones empresariales se ejecutan en un servidor de aplicaciones. Una aplicación empresarial Java EE está formada por un conjunto de módulos donde cada módulo es un conjunto de uno o más componentes que se ejecutan en el mismo contenedor.

Un componente no es más que una unidad de software, puede ser un componente web como una página *JSP* o un *servlet*, un componente *EJB*, etc. Estos componentes se ejecutan dentro de su correspondiente contenedor dentro del servidor de aplicaciones.

El contenedor no es más que un entorno de ejecución que gestiona los componentes, por eso, los componentes deben de cumplir el contrato que establece el contenedor. Ese contrato no es más que un conjunto de métodos que debe implementar el componente y que permite al contenedor interactuar con él.

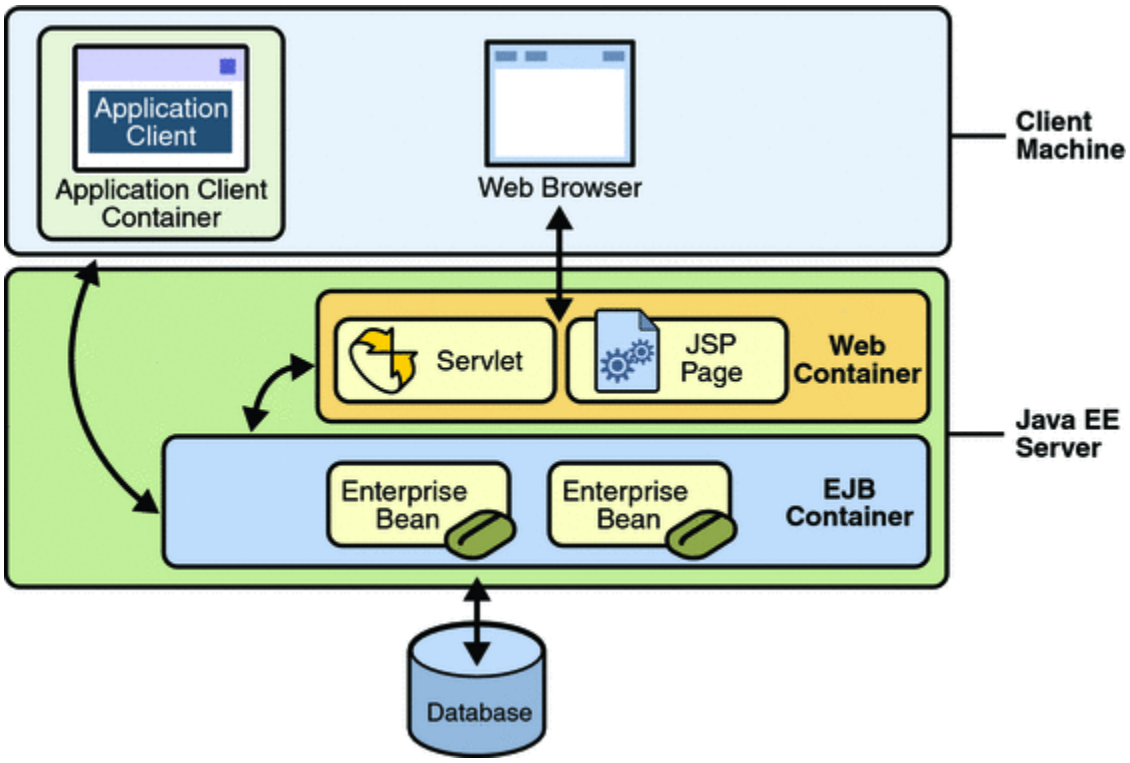

Ilustración 11: Capas del servidor de aplicaciones *Java EE*

Además, cada contenedor proporciona una serie de servicios que el componente puede utilizar. El contenedor es el encargado de gestionar el ciclo de vida de los componentes, realizar la reserva de recursos, etc. Algunos de estos servicios son servicios declarativos, esto quiere decir que algunos servicios se declaran en vez de programarse. La declaración se realiza mediante descriptores de despliegue. Cada módulo dispone de un descriptor de despliegue. El descriptor de despliegue no es más que un archivo *XML* que describe cómo se deben desplegar esos componentes en el contenedor del servidor de aplicaciones [13].

Los módulos que forman una aplicación empresarial pueden ser de tres tipos:

- **Archivos *JAR* (*Java Archive*)**: Los archivos *JAR* permiten agrupar distintos archivos .java en uno solo. Es el empleado para empaquetar componentes *EJBs*.

- **Archivos *WAR* (*Web Application Archive*)**: Los archivos *WAR* permiten empaquetar en una sola unidad aplicaciones web completas (*servlets*, páginas *JSPs*, contenido estático como imágenes y otros recursos Web).

- **Archivos *EAR* (*Enterprise Application Archive*)**: Los archivos *EAR* son archivos desplegables en servidores de aplicaciones *JEE*. Contienen archivos *WAR* y *EJBs* empaquetados en ficheros *JAR*.

  Por lo que podríamos decir que existen tres tipos de aplicaciones Java EE:

- Aplicaciones Web *JAVA*.
- Objetos distribuidos *EJBs*.
- Aplicaciones empresariales que engloba a las dos anteriores, aplicaciones web *JAVA* y objetos distribuidos *EJBs*.

2.3.2. Servidor de aplicaciones *JBoss*.

JBoss es un servidor de aplicaciones *Java EE* de código abierto implementado totalmente en Java. Al estar basado en Java, *JBoss* puede ser utilizado en cualquier sistema operativo para el que esté disponible la máquina virtual de *Java*. *JBoss Inc*., empresa fundada por Marc Fleury y que desarrolló inicialmente *JBoss*, fue adquirida por *Red Hat* en abril del 2006. A día de hoy el proyecto ha cambiado de nombre debido a la confusión entre la versión de pago *JBoss EAP* y el proyecto libre. Desde 2013 ha pasado a llamarse *WildFly*

Ilustración 12: Logotipo *JBoss 7*

El proyecto se nutre de una red mundial de colaboradores. Los ingresos de la empresa están basados en un modelo de negocio de servicios. *JBoss* implementa todo el paquete de especificaciones de *JEE*.

JBoss AS es el primer servidor de aplicaciones de código abierto, preparado para la producción y certificado *JEE 6*, disponible en el mercado, ofreciendo una plataforma de alto rendimiento para aplicaciones de *e-business*. Combinando una arquitectura orientada a

servicios *SOA*, con una licencia *GNU* de código abierto, *JBoss AS* puede ser descargado, utilizado, incrustado y distribuido sin restricciones por la licencia [14].

JBoss ha lanzado toda una serie de proyectos paralelos al servidor de aplicaciones que implementa las especificaciones *JEE*. Entre los más destacados y usados en este proyecto nos encontramos:

- ***Infinispan***

Este software implementa una caché para el acceso frecuente a objetos java con el propósito de mejorar el rendimiento de la aplicación. La caché puede ser replicable y transaccional. Puede ser replicada en una o varias máquinas virtuales *Java* y a través de la red. Puede ser transaccional debido a que la especificación *JTA* puede ser configurada de tal manera que cada operación de cache puede ser una transacción. *Infinispan* sustituye al proyecto *JBoss Cache* [15].

- ***HornetQ***

HornetQ es un software de mensajería asíncrona de código abierto perteneciente al proyecto *JBoss*. Procede del conocido *JBoss Messaging* 2.0. *HornetQ* implementa la especificación *JMS* y puede ser utilizado independientemente, es decir, sin necesidad del servidor de aplicaciones y usado por todo tipo de aplicaciones [16].

Posee las siguientes características:

- Soporta *STOMP* y *AMQP 1.0* protocolos para clientes entre idiomas
- 100 % compatible con *JMS*
- Hasta 8,2 millones de mensajes por segundo
- *Clustering* para la escalabilidad y la fiabilidad
- Soporta arquitectura Maestro/Esclavo para tolerancia a fallos
- Puede conectar a otros servidores *HornetQ* y otros servidores compatibles *JMS*
- Desvía el tráfico sin necesidad de modificar el código de aplicación
- Paginación para soportar los mensajes cuyo tamaño excede la memoria *RAM* disponible

- Soporte de transacciones *JTA*
- Configuración simple a través de *XML*

JBoss tiene además su propia implementación de los EJB y es el desarrollador principal de motor de persistencia *Hibernate* ya mencionado anteriormente.

2.3.3. *Tomcat*

Tomcat es un servidor web con soporte de *servlets* y *JSPs*. *Tomcat* no es un servidor de aplicaciones, como *JBoss*. Incluye el compilador *Jasper*, que compila *JSPs* convirtiéndolas en *servlets*. El motor de *servlets* de *Tomcat* a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. Hoy en día *Tomcat* es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad [17].

2.3.4. *Maven*

Maven es una herramienta de software para la gestión y construcción de proyectos Java creada por Jason van Zyl, de *Sonatype*, en 2002. Es similar en funcionalidad a *Apache Ant* (y en menor medida a *PEAR* de *PHP* y *CPAN* de *Perl*), pero tiene un modelo de configuración de construcción más simple, basado en un formato *XML*. Estuvo integrado inicialmente dentro del proyecto *Jakarta* pero ahora ya es un proyecto de nivel superior de la *Apache Software Foundation* [9].

Ilustración 13: Logotipo *Maven*

Maven utiliza un *Project Object Model* (*POM*) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Una característica clave de *Maven* es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar *plugins* de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos *Open Source* en *Java*, de Apache y otras organizaciones y desarrolladores. Este repositorio y su sucesor reorganizado, el repositorio *Maven 2*, pugnan por ser el mecanismo *de facto* de distribución de aplicaciones en Java, pero su adopción ha sido muy lenta. *Maven* provee soporte no sólo para obtener archivos de su repositorio, sino también para subir artefactos al repositorio al final de la construcción de la aplicación, dejándola al acceso de todos los usuarios. Una caché local de artefactos actúa como la primera fuente para sincronizar la salida de los proyectos a un sistema local.

Maven está construido usando una arquitectura basada en *plugins* que permite que utilice cualquier aplicación controlable a través de la entrada estándar. En teoría, esto podría permitir a cualquiera escribir *plugins* para su interfaz con herramientas como compiladores, herramientas de pruebas unitarias, etcétera, para cualquier otro lenguaje.

La filosofía general de *Maven* es la estandarización de las construcciones generadas por seguir el principio de Convención sobre Configuración, a fin de utilizar existentes modelos en la producción de software. Esta estrategia necesariamente restringe ampliamente la variabilidad, lo que se refleja en la exhortación de *Maven* a adherirse a su modelo de proyecto. Mientras que *Maven* es adecuado para nuevos proyectos, los proyectos complejos ya existentes pueden ser simplemente no adaptables para que utilicen *Maven*. La falta de restricciones de la convención de capas del proyecto fue hecha de alguna manera más configurable con el lanzamiento de *Maven 2*.

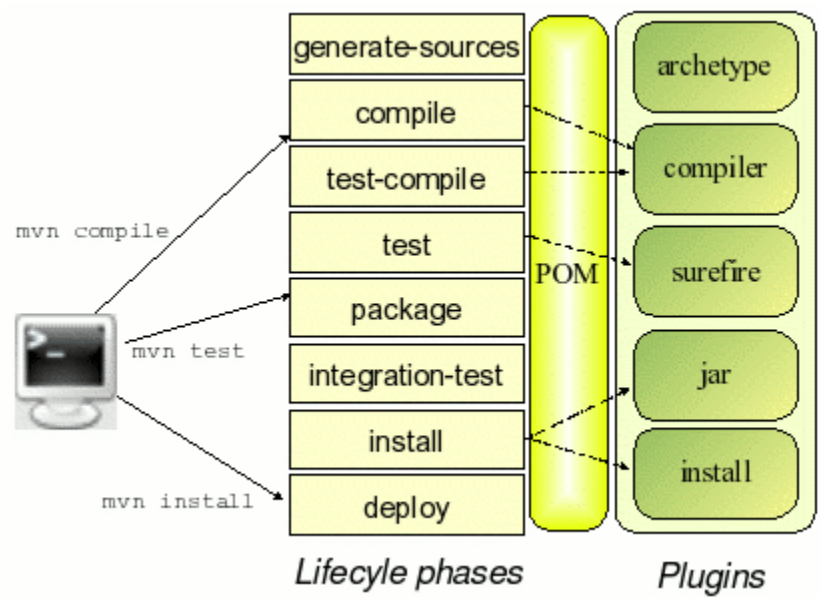

Ilustración 14: Esquema de funcionamiento de *Maven*

Maven está construido alrededor de la idea de reutilización, y más específicamente, a la reutilización de la lógica de construcción. Como los proyectos generalmente se construyen en patrones similares, una elección lógica podría ser reutilizar los procesos de construcción. La principal idea es no reutilizar el código o funcionalidad (como *Apache Ant*), sino simplemente cambiar la configuración o también código escrito. Esa es la principal diferencia entre Apache *Ant* y *Apache Maven*: el primero es una librería de utilidades y funciones buenas y útiles, mientras que la otra es un framework configurable y altamente extensible [10].

Aunque *Maven* es configurable, históricamente el proyecto *Maven* ha enfatizado seriamente que los usuarios deben adherirse a su concepto de un modelo de proyecto estándar tanto como sea posible.

Las partes del ciclo de vida principal del proyecto *Maven* son:

1. *compile*: Genera los ficheros *.class* compilando los fuentes *.java*
2. *test*: Ejecuta los test automáticos de *JUnit* existentes, abortando el proceso si alguno de ellos falla.
3. package: Genera el fichero *.jar* con los *.class* compilados
4. *install*: Copia el fichero .jar a un directorio de nuestro ordenador donde *Maven* deja todos los *.jar*. De esta forma esos *.jar* pueden utilizarse en otros proyectos *Maven* en el mismo ordenador.

5. *deploy*: Copia el fichero *.jar* a un servidor remoto, poniéndolo disponible para cualquier proyecto *Maven* con acceso a ese servidor remoto.

Cuando se ejecuta cualquiera de las órdenes *Maven*, por ejemplo, si ejecutamos **mvn install**, *Maven* irá verificando todas las fases del ciclo de vida desde la primera hasta la de la orden, ejecutando sólo aquellas que no se hayan ejecutado previamente.

También existen algunas metas que están fuera del ciclo de vida que pueden ser llamadas, pero *Maven* asume que estas metas no son parte del ciclo de vida por defecto (no tienen que ser siempre realizadas). Estas metas pueden ser añadidas al ciclo de vida a través del *Project Object Model* (*POM*). Estas son:

1. *clean*: Elimina todos los *.class* y *.jar* generados. Después de esta orden se puede comenzar un compilado desde cero.
2. *assembly:assembly*: Genera un fichero *.zip* con todo lo necesario para instalar nuestro programa java. Se debe configurar previamente en un fichero *xml* qué se debe incluir en ese *zip*.
3. *site*: Genera un sitio web con la información de nuestro proyecto. Dicha información debe escribirse en el fichero pom.xml y ficheros *.apt* separados.
4. *site-deploy*: Sube el sitio web al servidor que hayamos configurado.
5. etc.

2.3.5. *Struts 2*

Struts es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón *MVC* bajo la plataforma *Java EE* (*Java Enterprise Edition*). *Struts* se desarrollaba como parte del proyecto *Jakarta* de la *Apache Software Foundation*, pero actualmente es un proyecto independiente conocido como *Apache Struts*. Permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en las que *Java Enterprise* esté disponible lo convierten en una herramienta altamente disponible.

Ilustración 15:*Struts2* como fusión de dos *frameworks*.

Struts2 es un nuevo *framework* (anteriormente conocido como *WebWork 2*) que introdujo algunas mejoras sobre *Struts1*, de cara a simplificar las tareas más comunes en el desarrollo de aplicaciones web, así como mejorar su integración con *AJAX*, *REST*, *JSON*, etc. *Struts* se basa en el patrón de arquitectura de software *Modelo-Vista-Controlador* (*MVC*) el cual se utiliza ampliamente y es considerado de gran solidez. De acuerdo con este Framework, el procesamiento se separa en tres secciones diferenciadas llamadas el modelo, las vistas y el controlador. El modelo *MVC* se implementa en *Struts* usando un *servlet* como controlador para las peticiones del usuario. De este modo cada petición del usuario se relaciona con una acción. Cada acción se encarga de llevar a cabo la funcionalidad de la *URL*, accediendo a la sesión y petición *HTTP*, los parámetros del formulario y otros servicios; con lo que responde con el modelo de objeto *POJO*. Finalmente la acción devuelve un resultado, que está relacionado con una página *JSP* (u otro tipo como *JSON*, *Tiles* o *Streams*) que se envía al usuario. El uso de pila interceptores que se asocia a cada acción permite definir las reglas de acceso a las mismas, desde el control de la petición *HTTP* y sus parámetros hasta el control de acceso de usuarios y sus roles. Los interceptores son *POJOs* que implementan una misma interface, con lo cual pueden ser definidos por el desarrollador añadiendo la funcionalidad que necesite.

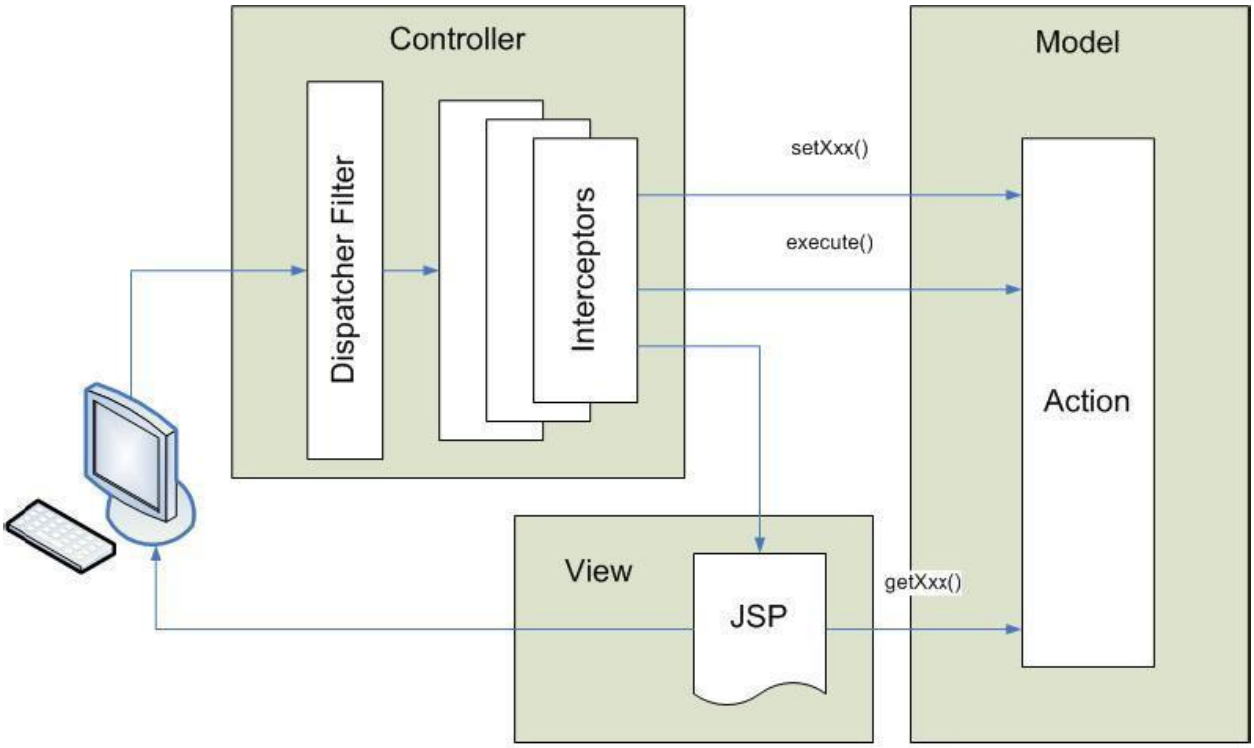

Ilustración 16: Modelo MCV de Struts2.

Algunas de sus características destacables son:

- Funcionamiento basado en acciones.
- Amplia comunidad de usuarios y desarrolladores.
- Las acciones pueden ser *Plain Old* Java *Objects* (*POJOs*), pudiendo ser ejecutadas por separado.
- Permite configuración mediante ficheros *XML* y/o mediante anotaciones dentro del código.
- Preparado para trabajar con diferentes *framework*, tanto de la capa de negocio como *EJB* o *Spring* hasta *SiteMesh*, *Tiles*, *Freemarker* o *Velocity*, *frameworks* de gestión de la vista mediante plantillas.
- Utiliza *OGNL* como lenguaje de expresiones, facilitando el acceso y la modificación del modelo desde la vista.
- Ofrece etiquetas para acceder a componentes preestablecidos *AJAX*.
- Existe infinidad de *plugins* que permiten extender el número.

Precisamente en este proyecto se han utilizado diferentes *plugins* de *Struts2* que han posibilitado la integración con diferentes tecnologías. Entre ellos están:

- ***Struts2 CDI plugin*** - Permite la inyección de dependencias directamente en las acciones de *Struts*.

- ***Struts2 Convention plugin*** - Ofrece el uso de anotaciones para configurar *Struts 2*. De esta manera las clases acción contienen su configuración reduciendo drásticamente la el tamaño del fichero struts.xml. Se facilita enormemente la comprensión de la vista.
- ***Struts2 JSON plugin*** - Añade el "*result*" de tipo *JSON*.

- ***Struts2 JQuery plugin*** - Define el uso de una colección de etiquetas para añadir funcionalidades de la librería *JQuery* que hacen uso de los objetos de datos declarados en las acciones. Desde llamadas *AJAX* hasta la biblioteca de funciones *JQuery UI* donde se incluyen *JGrid* (tablas) o *JChart* (gráficas).

- ***Struts2 Tiles2 plugin*** - Define el "*result*" de tipo *Tiles*.

- ***Struts2 - Bootstrap plugin*** - Añade una librería de etiquetas que permite trabajar con el popular *framework* creado por Twitter. Mediante este *plugin* podemos añadir plantillas basadas en *Bootstrap*. Esto permite incluir diseños "*responsive design*" que se adaptan a diferentes dispositivos.

- ***Struts2 conversation plugin*** - Permite el uso de la funcionalidad "*wizards*" salvando datos en la sesión entre diferentes peticiones *HTML*.

2.3.6. *MySQL*

En este proyecto se utiliza el popular sistema de gestión de bases de datos *MySQL*. Subsidiaria desde 2008 de *Sun Microsystem* y a su vez, de *Oracle* desde abril de 2009, *MySQL* es un sistema de gestión de bases de datos relacional, multihilo y multiusuario. Se ofrece bajo la *GNU GPL* para cualquier uso compatible con esta licencia, pero para aquellas empresas que quieran incorporarlo en productos privativos deben comprar una licencia específica que les permita este uso. Está desarrollado en su mayor parte en *ANSI C*.

Ilustración 17: Logotipo de *MySQL*.

Es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional *MyISAM*, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones web existe baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a *MySQL* ideal para este tipo de aplicaciones. Sea cual sea el entorno en el que va a utilizar *MySQL*, es importante monitorizar de antemano el rendimiento para detectar y corregir errores tanto de *SQL* como de programación.

En los casos en los que la aplicaciones requieren de transacciones seguras, se utiliza el motor *InnoDB*. *InnoDB* es un mecanismo de almacenamiento de datos de código abierto para la base de datos *MySQL*, incluido como formato de tabla estándar en todas las distribuciones de *MySQL* a partir de las versiones 4.0. Su característica principal es que soporta transacciones de tipo *ACID* y bloqueo de registros e integridad referencial. *InnoDB* ofrece una fiabilidad y consistencia muy superior a *MyISAM*, la anterior tecnología de tablas de *MySQL*, si bien el mejor rendimiento de uno u otro formato dependerá de la aplicación específica.

MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas bajo petición.

Las siguientes características son implementadas únicamente por *MySQL*:

- Permite escoger entre múltiples motores de almacenamiento para cada tabla. En *MySQL 5.0* éstos debían añadirse en tiempo de compilación, a partir de *MySQL 5.1* se pueden añadir dinámicamente en tiempo de ejecución:
    - Los hay nativos como *MyISAM*, *Falcon*, *Merge*, *InnoDB*, *BDB*, *Memory*/*heap*, *MySQL Cluster*, *Federated*, *Archive*, *CSV*, *Blackhole* y *Example*.
    - Desarrollados por *partners* como *solidDB*, *NitroEDB*, *ScaleDB*, *TokuDB*, *Infobright* (antes *Brighthouse*), *Kickfire*, *XtraDB*, *IBM DB2*). *InnoDB* estuvo desarrollado así pero ahora pertenece también a *Oracle*.
    - Desarrollados por la comunidad como *memcache*, *httpd*, *PBXT* y *Revision*.
- Agrupación de transacciones, reuniendo múltiples transacciones de varias conexiones para incrementar el número de transacciones por segundo.

2.3.7. Otras Librerías y utilidades

2.3.5.1. *Tiles 2*

Tiles permite a los desarrolladores definir páginas con fragmentos que pueden ser ensamblados dentro de páginas completas en tiempo de ejecución. Estos fragmentos o azulejos (en inglés "*Tiles*") pueden ser usados como simples elementos incluidos que permiten reducir la duplicidad entre elementos comunes en un diseño web. A sus vez estos fragmentos pueden también ser incluidos dentro de otros para desarrollar una serie de plantillas reutilizables. Estas plantillas definen el desarrollo de un estilo de diseño que es a la vez consistente y atractivo en toda la aplicación web. Se trata también de un proyecto *Apache*.

2.3.5.2. *JAMOD*

Es una librería que implementa el protocolo *ModBus*, que en este proyecto se utiliza para acceder al hardware de adquisición de datos. Se presenta como una librería totalmente orientada objetos que facilita la obtención de las lecturas de los sensores y actuadores conectados al correspondiente módulo a través de *TCP/IP*. Además es gratuita y de código abierto.

2.3.5.3. *SMSLIB*

SmsLib es una librería java que provee una *API* de mensajería *SMS* universal, lo cual permite enviar y recibir mensajes a través de módems *GSM*. De código abierto, se distribuye bajo los términos de la licencia *Apache V2*.

2.4 Recursos hardware

Este proyecto conserva el hardware utilizado en los proyectos anteriores. Si bien se planteó en un principio estudiar la inclusión de módulos de adquisición de datos que trabajan a través de conexiones inalámbricas, se descartó este estudio por la prioridad de otras actuaciones sobre la aplicación.

2.4.1. Equipo servidor

En cuanto al sistema informático requerido por la aplicación me remito a los proyectos de fin de carrera anteriores. En realidad cualquier equipo comprado a día de hoy donde exista soporte para la máquina virtual *Java* será posible correr la aplicación (independientemente de su arquitectura). La única restricción aparece por el uso del módem *GSM* que se conecta a través de un puerto serie. El resto de software y hardware es perfectamente soportado por múltiples plataformas. Java ofrece un pobre soporte para estos dispositivos (algo anticuados de por sí). En las versiones anteriores de la aplicación se imponía el uso de arquitecturas *x86* de Intel y sistemas *Windows* que permiten la comunicación a través del controlador *JavaComm v2* de Java. Con la actualización realizada en este proyecto se utiliza el controlador *RxTx* para el cual existen versiones de archivos binarios para arquitecturas x86, su extensión amd64 e *Itanium ia64*. Las dos primeras están disponibles para sistemas operativos *Windows* y *Linux*.

2.4.2. Módulos de adquisición de datos

2.4.2.1. Módulo *ADAM 6017*

Las características más destacadas del módulo de adquisición de datos *ADAM 6017* son:

- 8 Entradas analógicas diferenciales.
- 2 Salidas digitales.
- Tipo de entrada: mV, V, mA.
- Rangos de entrada: ±150 mV, ±500 mV, ±1 V±5 V, ±10 V, 0-20 mA, 4-20 mA.
- Precisión: ±0,1% en voltaje, ±0,2% en corriente
- Interfaz *Ethernet* 10/100Mbps.
- Soporta protocolos *Modbus/TCP*, *TCP/IP*, *UDP* y *HTTP*.
- Consumo 2W a 24V en continua.
- Precisión datos: 16bits
- Ratio de muestreo: 10 muestras/segundo.
- Temperatura de operación: industrial -10º - 70º.

Ilustración 18: Módulo *ADAM 6017*.

2.4.2.2. Módulo *ADAM 6022*

Las características más destacadas del módulo de adquisición de datos *ADAM 6022* son:

- 2 circuitos con:
    - 3 entradas analógicas diferenciales.
    - 1 salida analógica.
    - 1 entrada digital.
    - 1 salida digital.
- Tipo de entrada: mV, V, mA.
- Rangos de entrada: ±150 mV, ±500 mV, ±1 V±5 V, ±10 V, 0-20 mA, 4-20 mA.
- Precisión: ±0,1% en voltaje y en corriente
- Interfaz *Ethernet* 10/100Mbps.
- Soporta protocolos *Modbus/TCP*, *TCP/IP*, *UDP* y *HTTP*.
- Consumo 2W a 24V en continua.
- Precisión datos: 16bits en analógico y 12bits en digital.
- Ratio de muestreo: 10 muestras/segundo.
- Temperatura de operación: industrial -10º - 50º.

Ilustración 19: Módulo *ADAM 6022*.

2.4.2.3. Módulo *ADAM 6022*

El módulo *ADAM 6510* es un *hub* para uso industrial con 5 puertos.

Ilustración 20: Módulo *ADAM 6510*.

2.4.3. Módem *GSM Wavecom Fastrack M1306B*

Ilustración 21: Módem GSM Wavecom Fastrack M1306B.

Este módem de banda dual 900/1800 MHz ofrece conectividad a redes móviles GSM/GPRS a través de una tarjeta de operador móvil. Esto permite el envío y recepción de mensaje *SMS* controlado por todo tipo de aplicaciones mediante su comunicación por puerto serie *RS232*. Ofrece ratios de entre 300 y 115200 baudios, usando por defecto 9600 bits/s.

2.5. Revisión de los casos de uso

En este proyecto no se modificarán notablemente los casos de uso ya que no se propone ningún tipo de cambio en la funcionalidad de la aplicación. De tal manera que la referencia a la documentación de los proyectos anteriores y una pequeña reseña a las mínimas modificaciones realizadas bastaría. Sin embargo considero necesario realizar una actualización de los casos de uso. La lectura y análisis de los casos de uso de la documentación anteriores no es sencilla, ni facilita la comprensión del sistema. Los casos de uso representan un análisis de la funcionalidad de un sistema software traducida desde el lenguaje humano o de la organización que se desea modelar. En numerosos proyectos muchas veces representa una colección de intenciones. ¿Qué queremos que el software haga? Aunque por diferentes motivos nunca se llegue a implementar la funcionalidad descrita, conviene actualizarlos para poder entender el sistema hasta el punto que haya alcanzado. A continuación se exponen estos casos de uso expresados en diagrama UML [21], [22]. También se han actualizados las tablas que contienen los casos de uso completos, que pueden ser consultadas en el Anexo C de esta memoria.

2.5.1. Actores del sistema

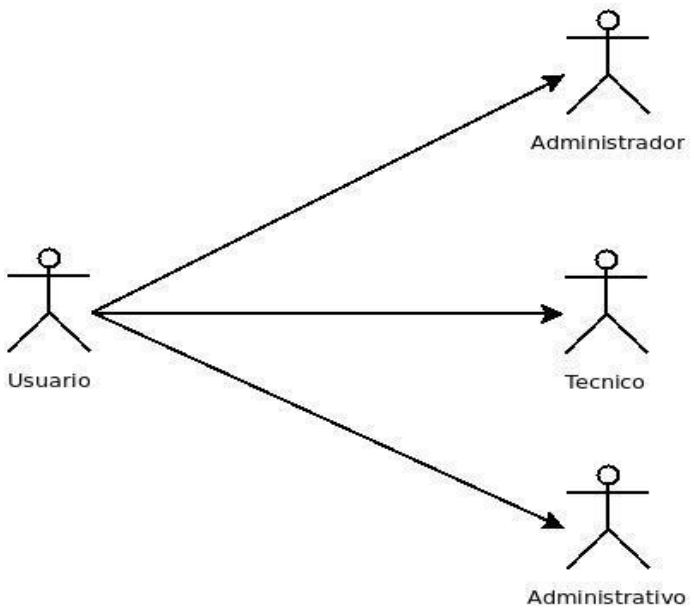

Ilustración 22: Generalización de actores.

2.5.1.1. Administrador

Es el encargado de revisar el trabajo de los otros técnicos, además también desempeña las funciones propias de un técnico. Utiliza todas las utilidades del software que utilizan los técnicos, además se ocupa de supervisar su trabajo.

2.5.1.2. Técnico

Se ocupan de recibir las alertas del sistema y llevan cabo la instalación de los sensores, los módulos de adquisición, los actuadores, etc. Para estas tareas usan las funcionalidades de los módulos de control. También son ellos los que definen las secuencias de acciones, alarmas y las acciones pertinentes en función de los valores de las lecturas de los sensores.

2.5.1.3. Administrativo

Se ocupan de vigilar las plantas. Son los actores que menos intervienen en la aplicación. Utilizan las consultas de los estados de los sensores y de la planta para realizar ese cometido. También manejan la documentación y la consulta de los datos de los usuarios del sistema.

2.5.2. Gestión de sensores

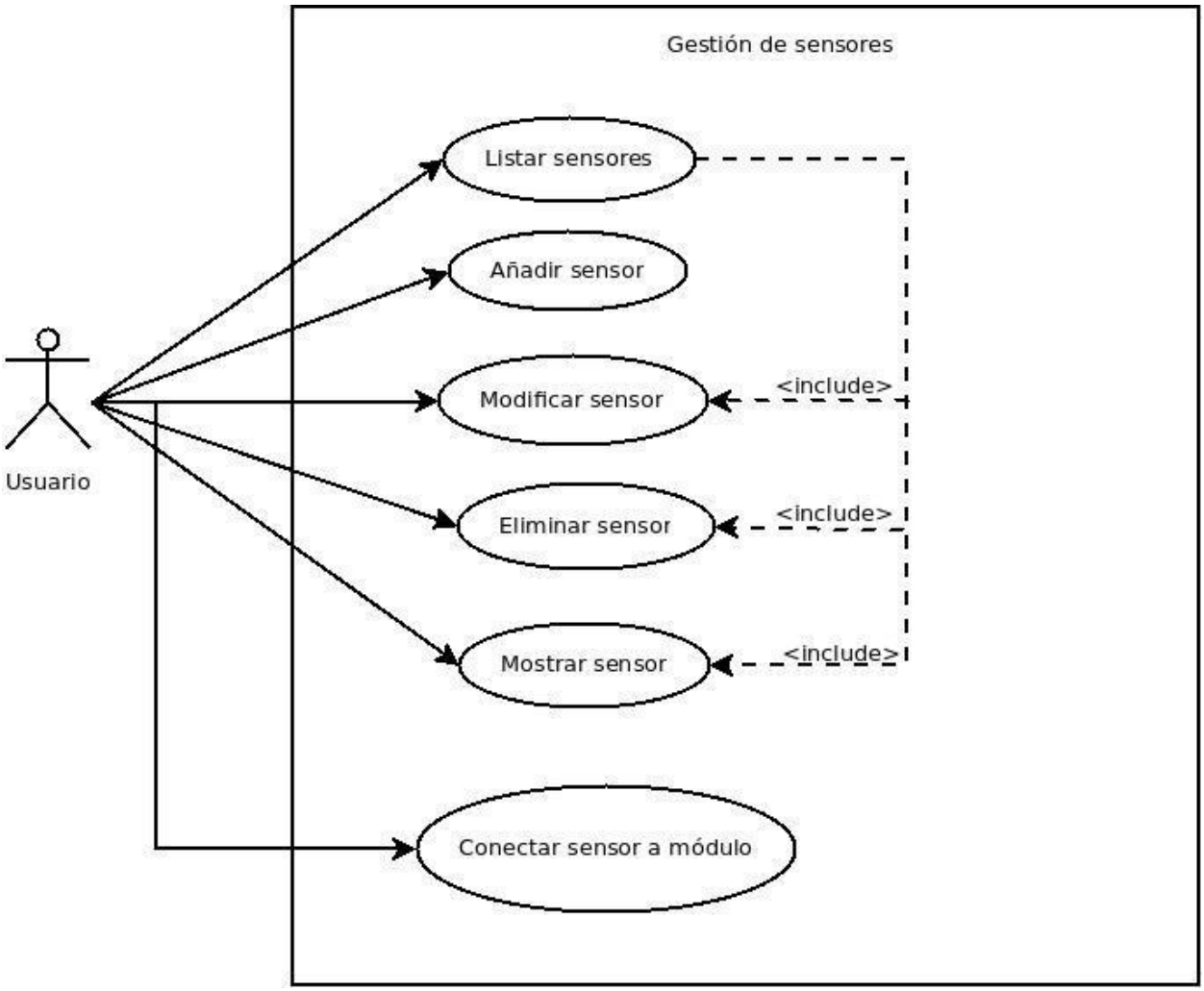

Ilustración 23: Diagrama de casos de uso del módulo de sensores.

2.5.3. Gestión de actuadores

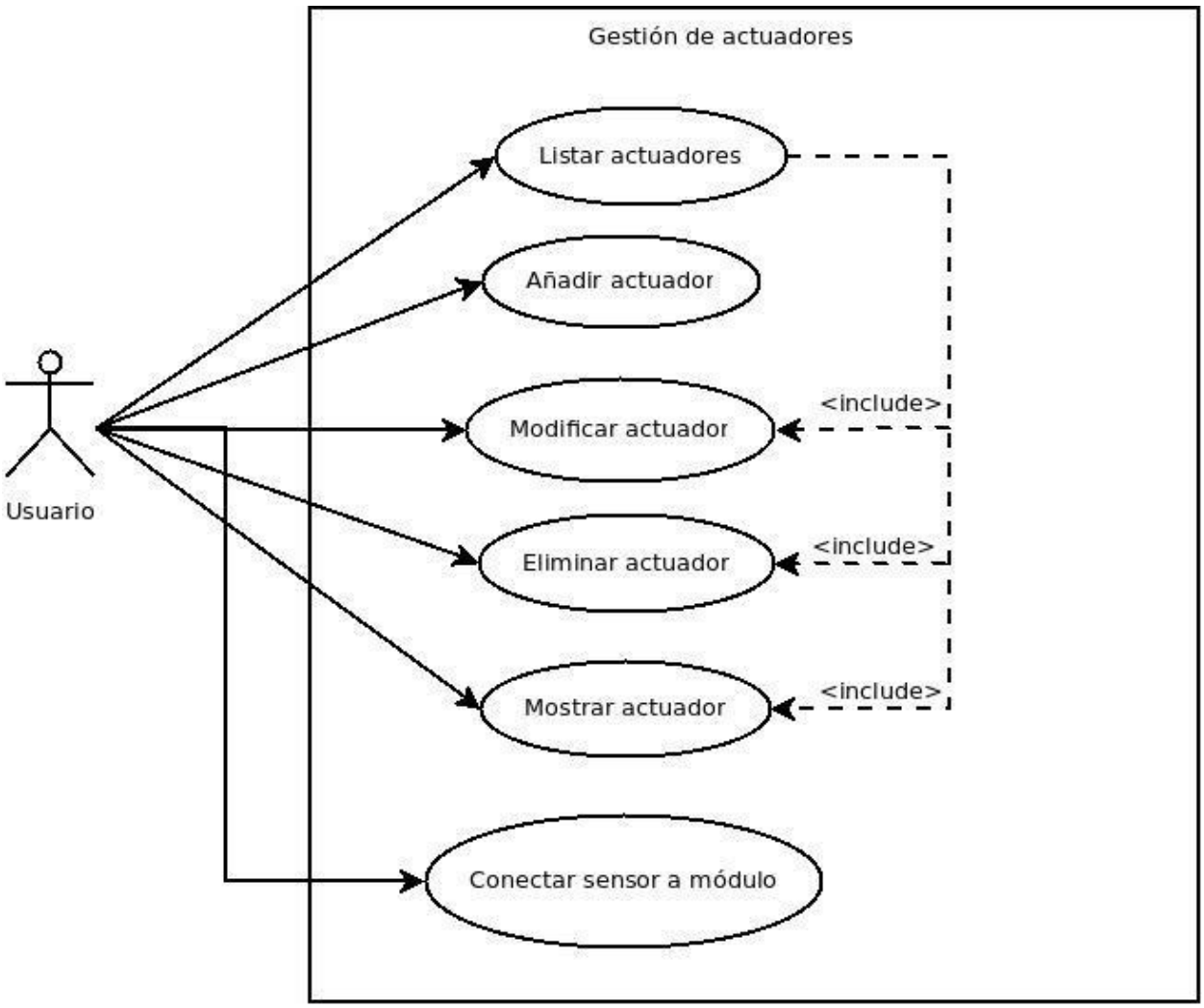

Ilustración 24: Diagrama de casos de uso del módulo de actuadores.

2.5.4. Gestión de módulos

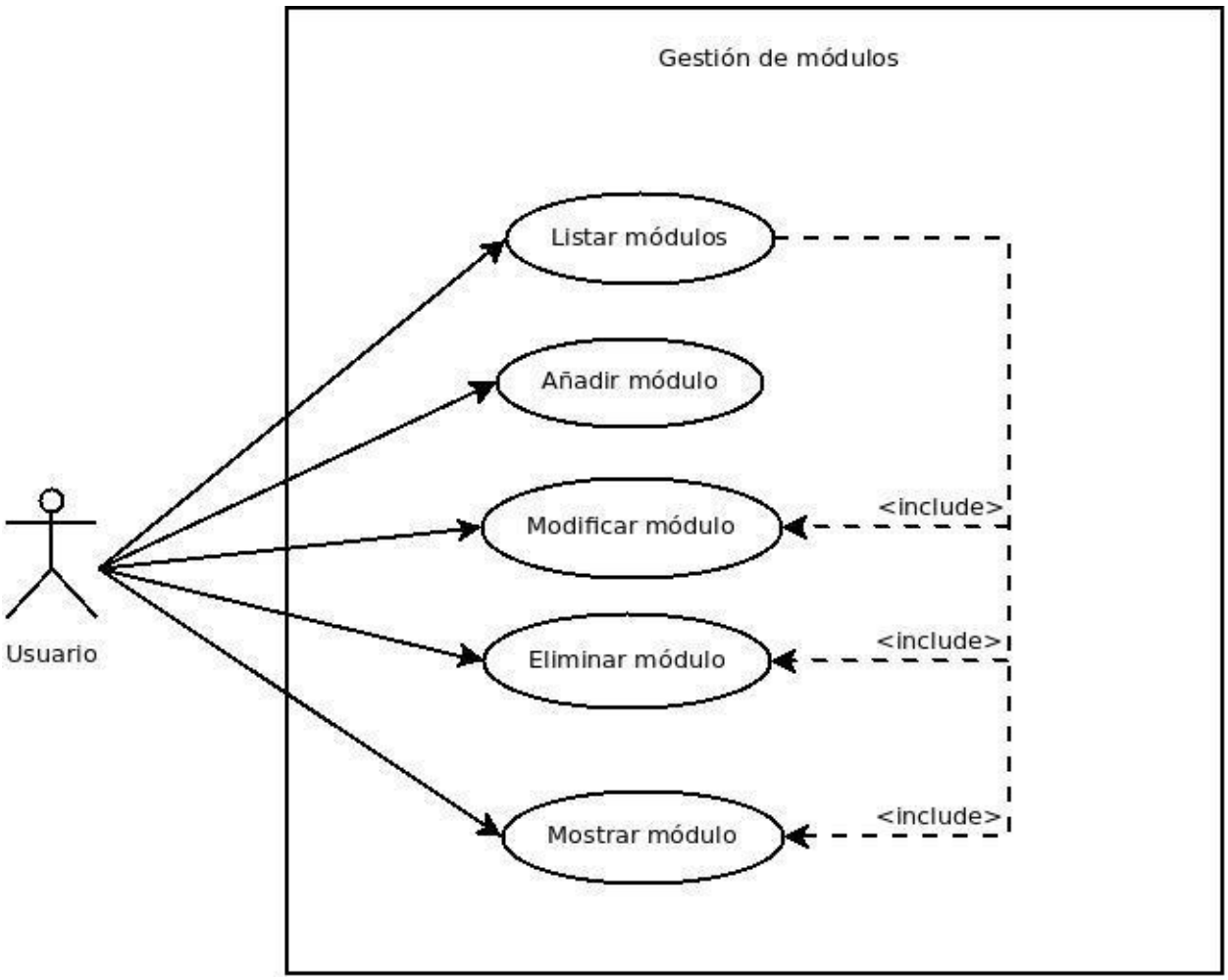

Ilustración 25: Diagrama de casos de uso de los módulos de adquisición de datos.

2.5.5. Gestión de diagramas

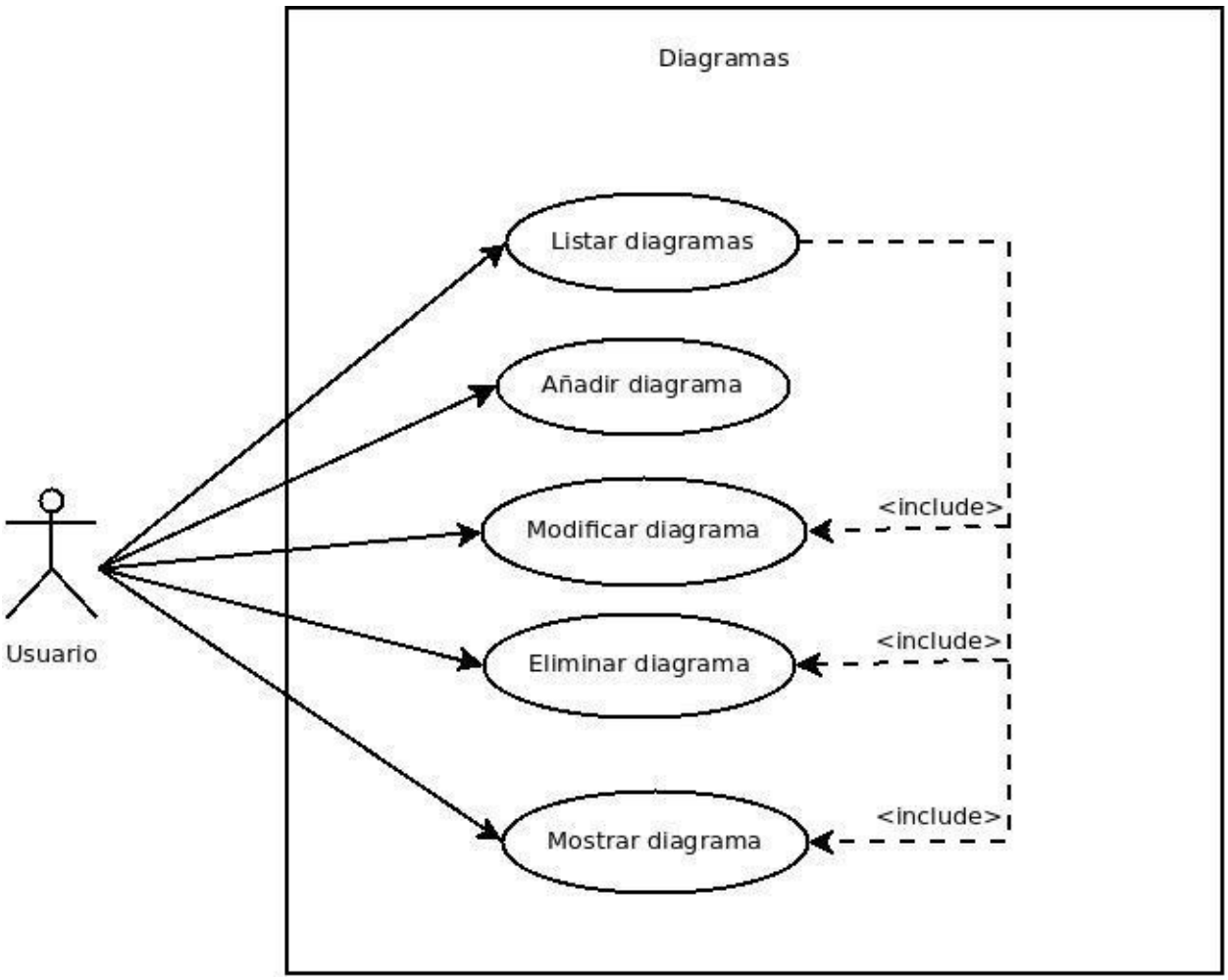

Ilustración 26: Diagrama de casos de uso del módulo de diagramas.

2.5.6. Gestión de usuarios

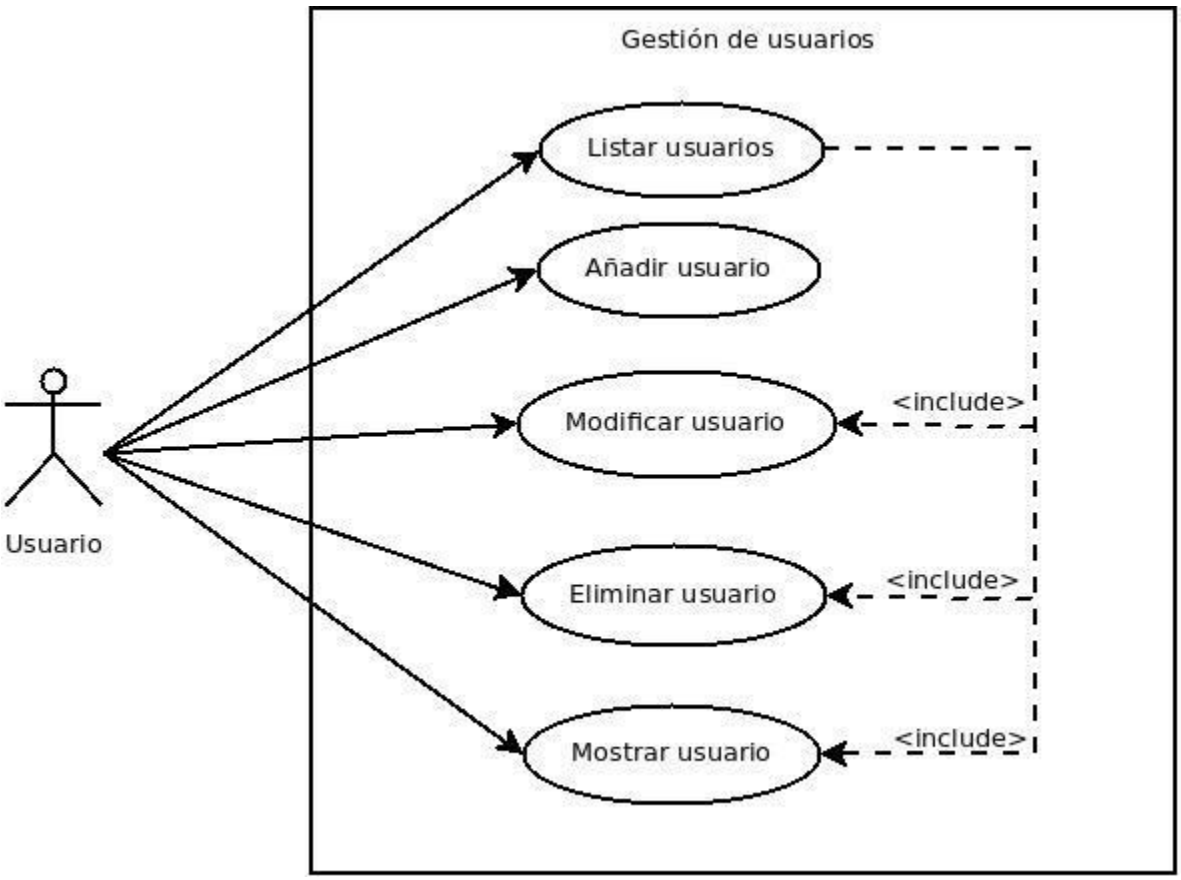

Ilustración 27: Diagrama de casos de uso del módulo de usuarios.

2.5.7. Gestión de alarmas

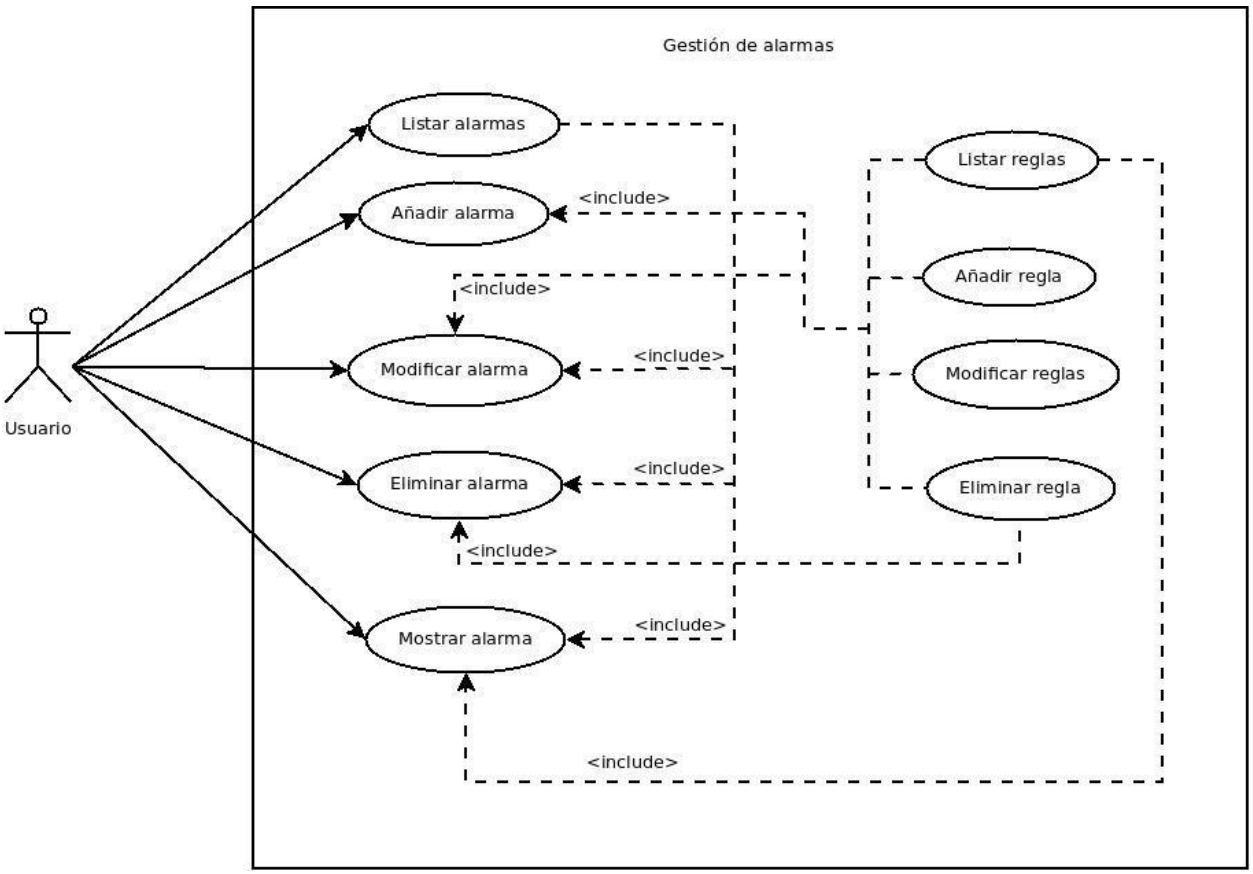

Ilustración 28: Diagrama de casos de uso del módulo de alarmas.

2.5.8. Gestión de planta

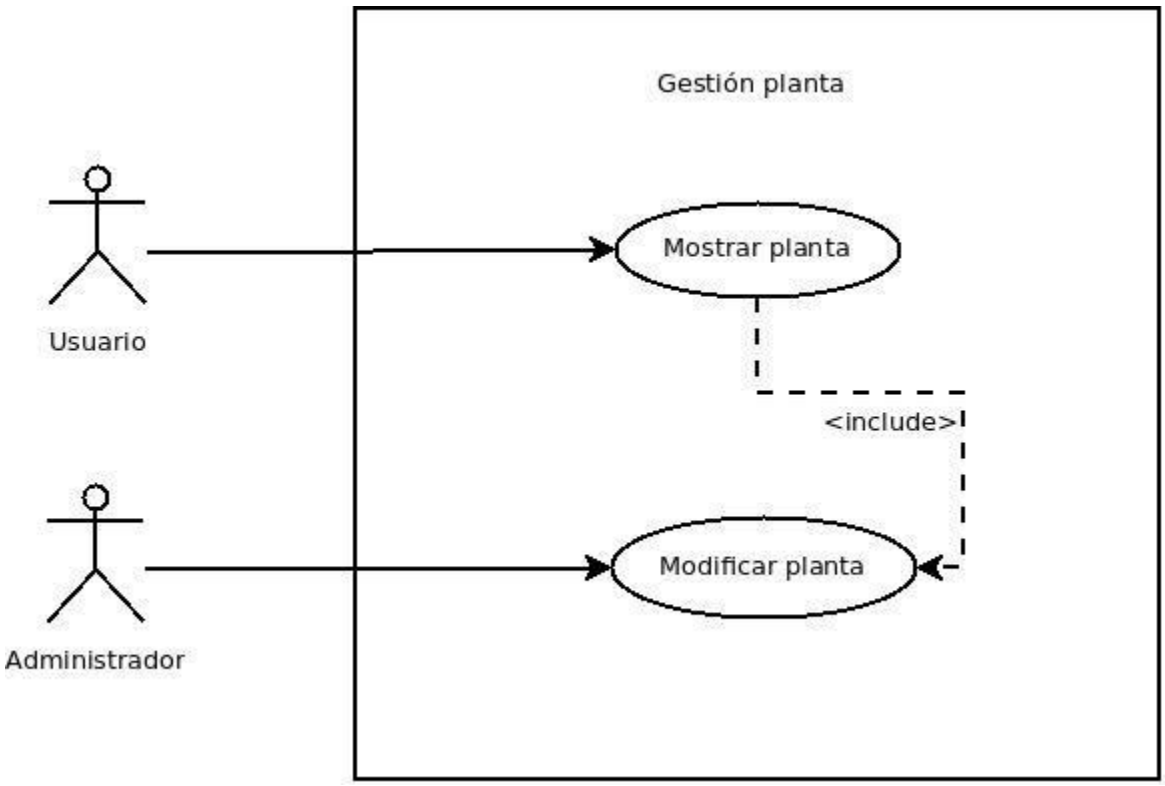

Ilustración 29: Diagrama de casos de uso del módulo de plantas.

2.5.9. Gestión de sms

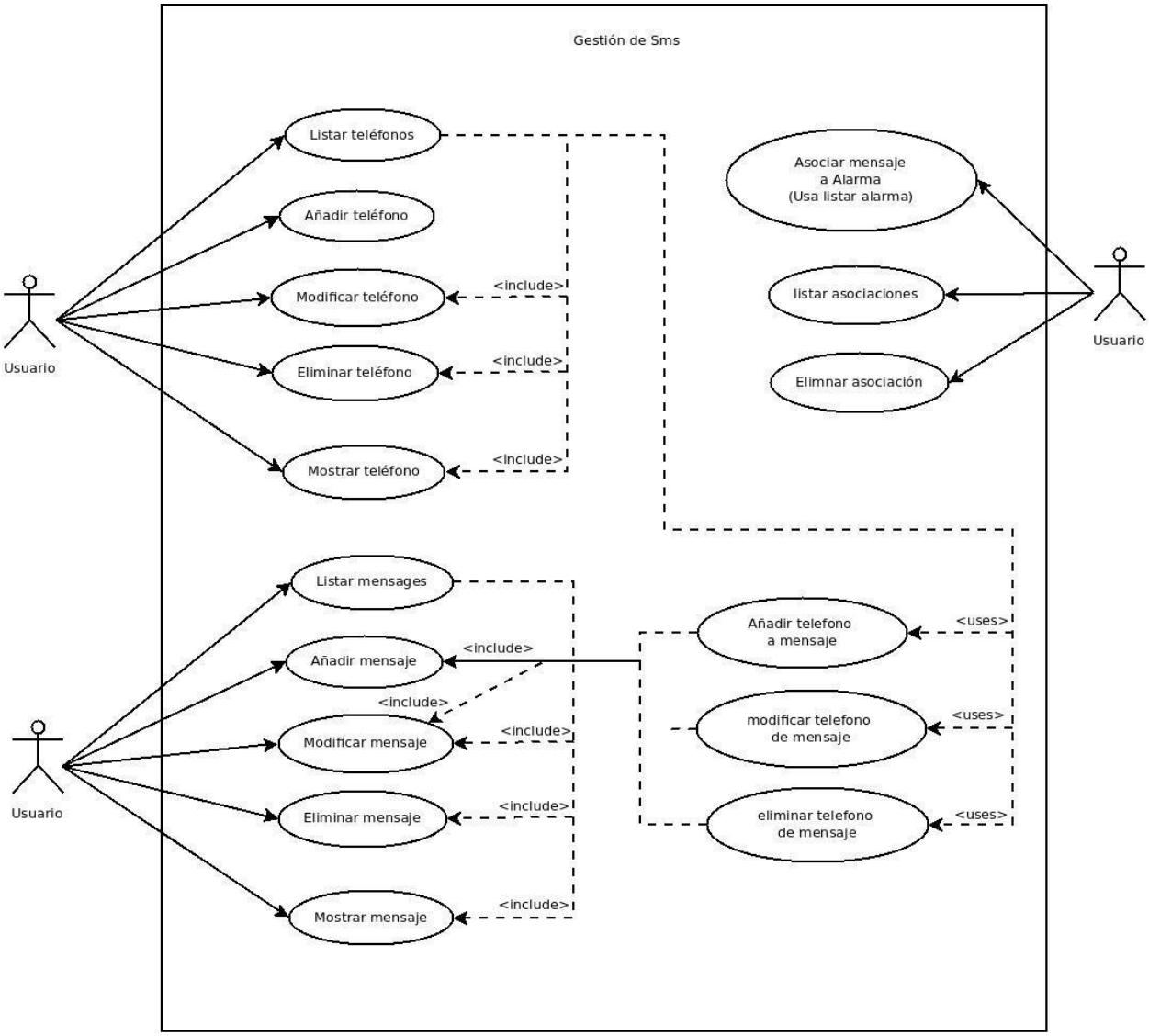

Ilustración 30: Diagrama de casos de uso del módulo de sms.

2.5.10. Gestión de secuencias

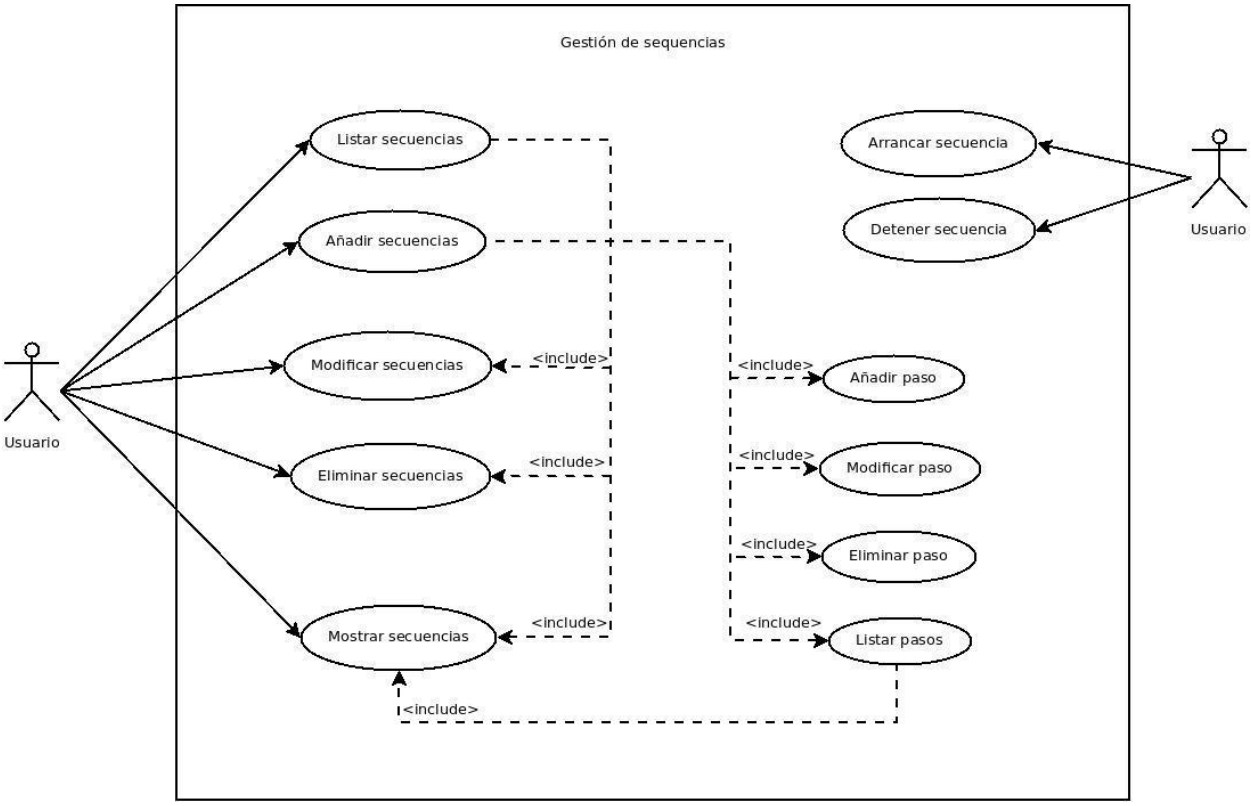

Ilustración 31: Diagrama de casos de uso del módulo de secuencias.

2.5.11. Gestión de históricos

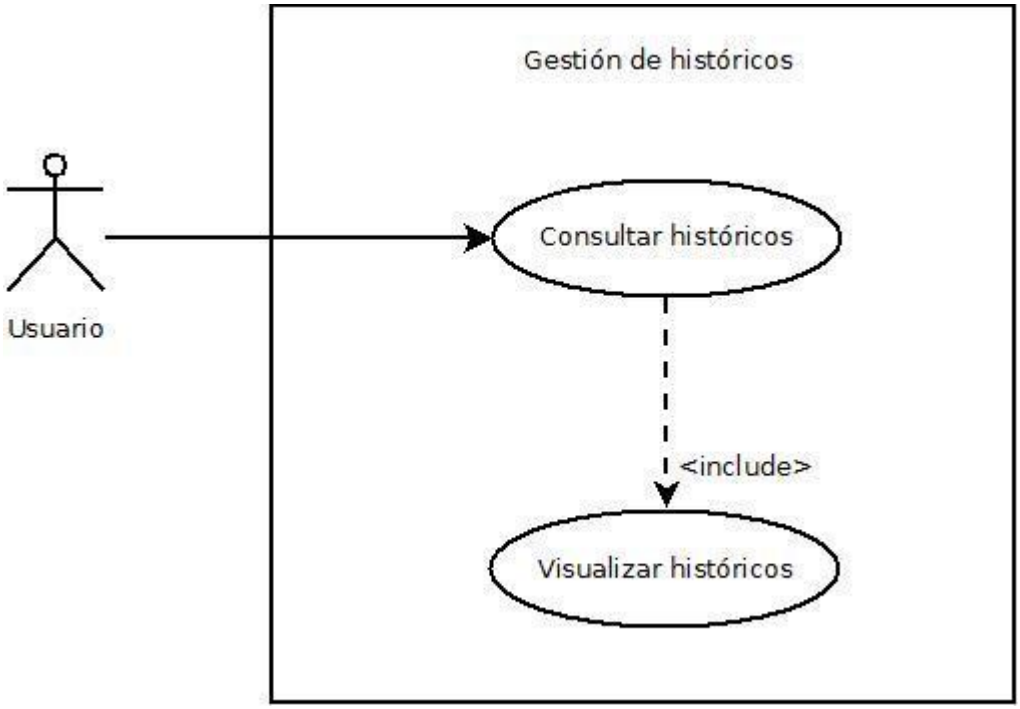

Ilustración 32: Diagrama de casos de uso del módulo de históricos.

3. Diseño

3.1. Iteración 1

Recordando lo escrito en la sección 2.2.1 que corresponde al análisis de la primera fase o iteración de este proyecto, nuestro primer paso será introducir *Maven* en la aplicación. *Maven*, al estar escrito en *Java* no necesita instalación. Podemos descargarnos su última versión desde su sitio web en la fundación Apache. En este proyecto ha sido usada la versión 3.1.1.

Una vez instalado conviene declarar la ruta mediante variables del sistema o indicar en nuestro *IDE* favorito donde se encuentra tal instalación. Esto lo indicaremos con detalle en la guía del desarrollador en anexo B. Cabe destacar que *Maven* puede ser utilizado a través de una consola de comando si es necesario.

Una vez *Maven* esté funcionando, usaremos el arquetipo "*jboss-javaee6-webapp-ear-archetype*", disponible en el repositorio global de *Maven* y mantenido por JBoss. Este arquetipo generará un proyecto Java para *JBoss AS 7.1* que consiste en un E*A*R (archivo usado en *Java EE* para empaquetar uno o varios módulos) que incluye un módulo *EJB-JAR* y otro *WAR*. Además añadiremos a nuestro *EAR* otro módulo *JAR* donde incluiremos las entidades de persistencia, siendo este un proyecto *JPA* independiente en sí mismo.

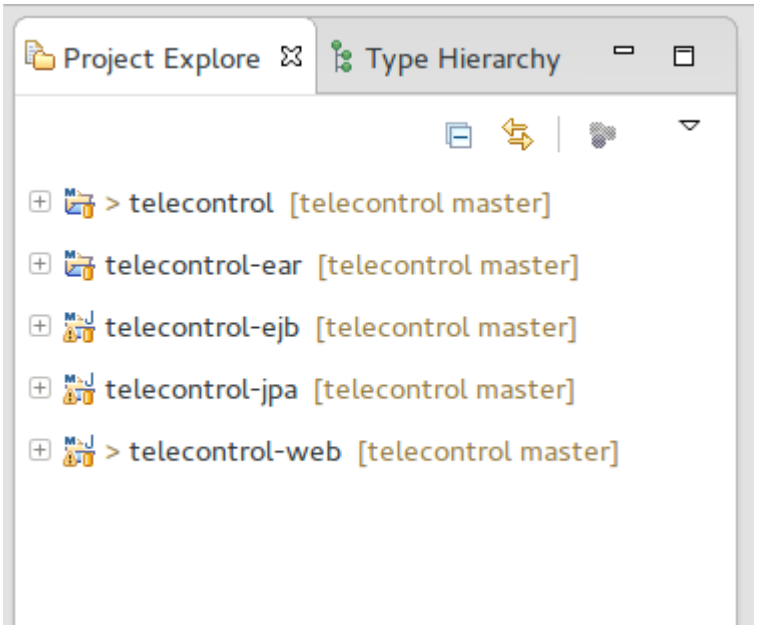

Ilustración 33: Disposición de los nuevos proyectos.

La cabecera del fichero POM (*Project Object Manager*) contiene las siguientes líneas:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>project</groupId>
    <artifactId>telecontrol</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>pom</packaging>
    <name>telecontrol application</name>

    <modules>
        <module>telecontrol-ejb</module>
        <module>telecontrol-web</module>
        <module>telecontrol-ear</module>
        <module>telecontrol-jpa</module>
    </modules>
```

Con el nuevo proyecto ya creado añadiremos el código fuente en la misma disposición modular que define la aplicación hasta ahora. El resultado después de cambiar nombres de los paquetes es:

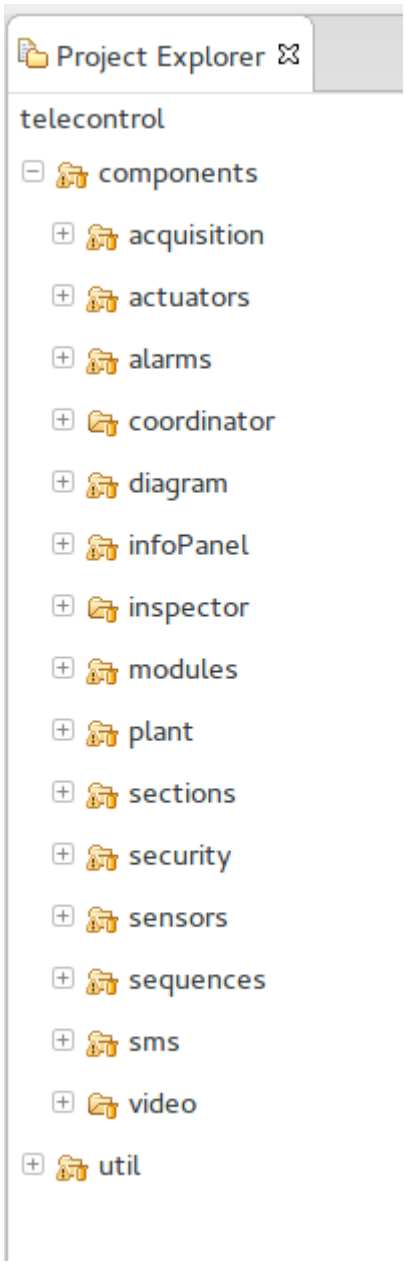

Ilustración 34: Organización modular del subproyecto *EJB-JAR*.

El módulo web tiene la siguiente disposición:

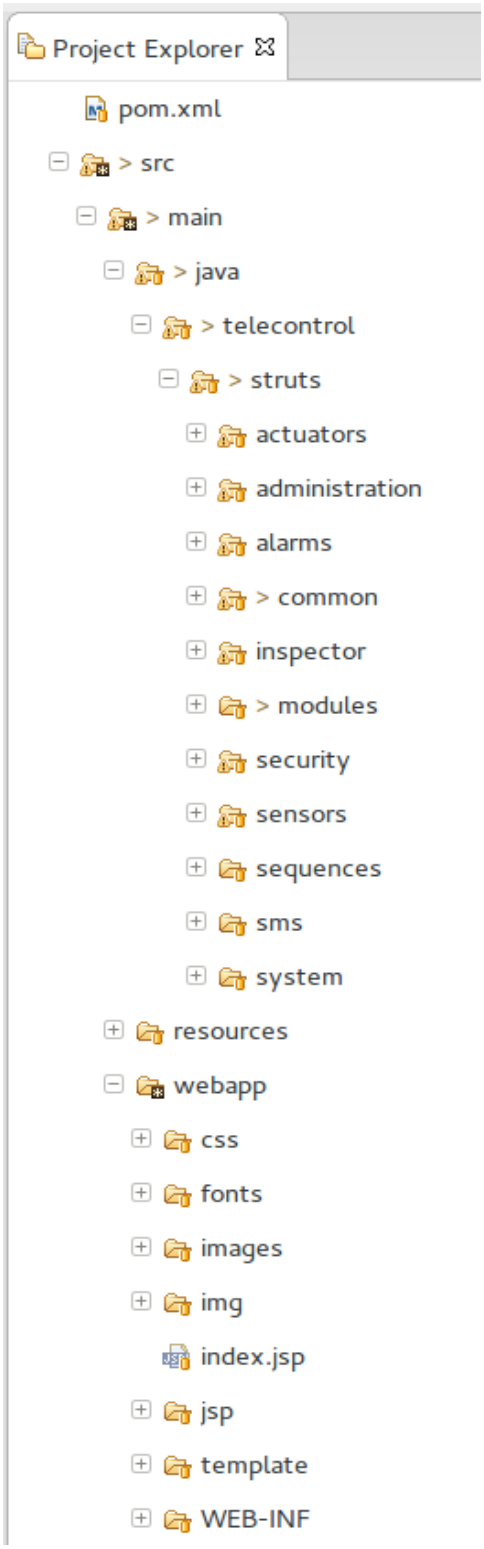

Ilustración 35: Organización modular del subproyecto *WAR*.

Necesitamos añadir a cada proyecto las dependencias necesarias. Necesitamos por lo tanto definir un repositorio local donde se almacenarán las librerías. Los repositorios de *Maven* funcionan de manera muy similar a los repositorios distribuidos de sistema de control de versiones *GIT*. En el fichero de configuración "*settings.xml*" podemos indicar con el *tag* "<localRepository>" donde se encuentra nuestro repositorio. Por defecto, *Maven* creará uno por nosotros, según nuestro usuario y sistema operativo.

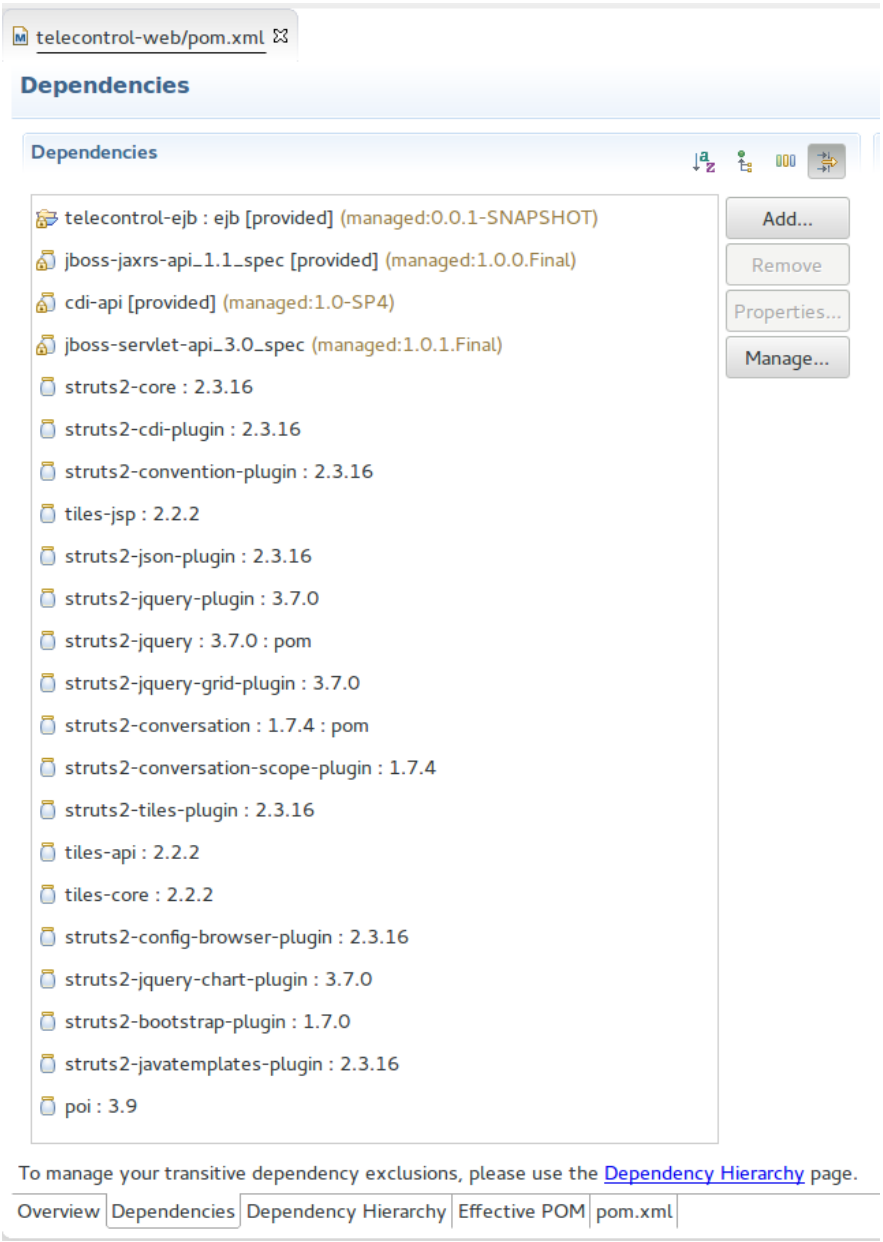

Ilustración 36: Interfaz gráfica para la gestión de dependencias.

También es necesario añadir aquellos repositorios remotos donde se encuentren las librerías que el proyecto requiere. El repositorio global de *Maven* contiene miles y miles de librerías pero no todas están presentes. En muchas ocasiones los desarrolladores prefieren añadir sus librerías a otros repositorios o crear los suyos propios. Para añadir estos repositorios remotos añadiremos también en el fichero de configuración tantas etiquetas “<repository>” como necesitemos.

El último detalle corresponde a aquellas librerías que no se encuentran en ningún repositorio de *Maven*. En este caso solamente tenemos que añadirlas usando la consola de comandos a nuestro repositorio local.

Una vez terminado y comprobado que *Maven* funciona y nuestro proyecto puede ser compilado, hemos resuelto todas las tareas planteadas en la primera iteración. Tenemos los fuentes del proyecto organizados en módulos. Disponemos de un sistema para generar los archivos binarios que ejecutará el servidor de aplicaciones. También poseemos un sistema que organiza y controla las dependencias de nuestro proyecto permitiéndonos añadir librerías con suma facilidad (sólo un comando es necesario o algunos clics si trabajamos con un entorno de desarrollo que integre *Maven*). Además nuestro proyecto queda añadido de manera automática a nuestro repositorio desde donde podemos gestionar el control de versiones y la distribución del mismo si lo deseáramos. *Maven* permite la interconexión entre repositorios de manera abierta o autenticada permitiendo a una organización mantener un control exhaustivo y distribuido a través de redes del código fuente.

También *Maven* permite definir nuestro sistema de construcción y despliegue. Podemos decirle dónde y cómo queremos que los paquetes *EAR*, *JAR* y *WAR* sean depositados tras el proceso de compilación. De esta manera será posible testear cualquier cambio que realicemos en nuestro código en nuestro servidor de aplicaciones.

Por último *Maven* permite la ejecución automatizada de tests como uno de sus pasos en el ciclo de vida del proyecto. Solamente sería necesario escribir nuestros tests y *Maven* se encargaría de ejecutarlos por nosotros y ofrecernos los resultados de los mismos. También nos permitirían utilizar un sistema de integración continua (como podría ser Jenkins) que construya el proyecto *Maven* desde sus ficheros de descripción *xml* y lleve un control sobre los cambios en nuestro software.

Nos encontramos ahora con un software que ha automatizado todo su ciclo de vida y que se corresponde con las herramientas y técnicas actuales de los procesos de desarrollo profesional esperados hoy en día.

3.2. Iteración 2

Una vez nuestro proyecto está totalmente integrado en *Maven*, el siguiente paso es actualizar el servidor de aplicaciones. Como se podría pensar, este cambio no debería suponer ningún tipo de problema, más allá de algunas nuevas especificaciones en los ficheros de configuración del servidor. Sin embargo el cambio va más allá. Nos movemos de una versión de *JBoss* de 2006 a otra de 2013.

El primer asunto a resolver no tiene que ver con el servidor *JBoss* directamente sino con la especificación de *Java EE* que pasa desde la versión 5 a 6. *Java EE* introduce nuevas funcionalidades para los componentes *EJB*, aunque no imponen cambios obligatorios. Por ejemplo, se podrían eliminar todas las interfaces de los beans de sesión, aunque considero positivo de momento conservarlos. En ellas podemos encontrar toda la funcionalidad del *bean* sin indagar en su implementación.

Como hemos dicho anteriormente la versión 6 introduce la inyección de dependencias *CDI*. Si bien esto no afectará a las capas de negocio y de datos, si supondrá un cambio en la capa web ya que elimina la necesidad de los “*lookup*” o la búsqueda de objetos dentro del contenedor. Si tenemos en cuenta también que la

nomenclatura del servicio de nombres *JNDI* ha cambiado, esto nos ahorrará muchos cambios. La nueva nomenclatura introduce los siguientes espacios de nombres:

- java:comp/ - El espacio de nombres pertenecientes al actual componente (p.e. EJB).
- java:module/ - Asociado al módulo actual.
- java:app/ - Asociado a la aplicación actual.
- java:global/ - Asociado a servidor de aplicaciones

JBoss 7 añade además los siguientes espacios de nombres a los cuatro definidos en la especificación de *JavaEE 6*:

- java:jboss/
- java:/

En los *beans* guiados por mensaje, y a modo de ejemplo deberemos realizar cambios como el mostrado a continuación en el componente de gestión de *sms*:

- *JBoss 4* + *JavaEE 5*:

```
@MessageDriven(activationConfig = {
            @ActivationConfigProperty(propertyName="destinationType",
propertyValue="javax.jms.Topic"),
            @ActivationConfigProperty(propertyName="destination",
propertyValue="desaladora/topic/alarms/AlarmsHandLer") })
```

- *JBoss 7* + *JavaEE 6*:

```
@MessageDriven(activationConfig = {
            @ActivationConfigProperty(propertyName="acknowledgeMode",
propertyValue="auto_acknowledge"),
            @ActivationConfigProperty(propertyName = "destinationType",
propertyValue = "javax.jms.Topic"),
```

```
            @ActivationConfigProperty(propertyName = "destination", propertyValue =
"java:/telecontrol/topic/alarms/AlarmsHandLer") })
```

En el caso de la búsqueda de las colas y topic la búsqueda de estos objetos se simplifica de este modo:

```
@Resource(mappedName = "java:/ConnectionFactory")
private TopicConnectionFactory topicConnectionFactory;

@Resource(mappedName = "java:/telecontrol/topic/blackBoardTopic")
private Topic blackBoard;
```

Este ejemplo no ha sido elegido al azar. Quiero llamar la atención en la propiedad “auto_acknowledge”. Esta propiedad fuerza el reconocimiento del mensaje cuando este llega a la cola o *topic*, aun cuando el manejador del mensaje no confirme su recepción. Debe también ser indicada cuando se envía el mensaje. En el proyecto anterior se indicaba la propiedad por defecto lo que hacía que los mensajes tuvieran carácter de transacción. Esto provoca que ante cualquier tipo de fallo (problemas de conexión con los módulos de adquisición de datos o base de datos) los mensajes quedarán a la espera de reconocimiento incluso después de parar el servidor *JBoss*. El sistema se sobrecarga hasta que recuperaba la conexión donde se procesaban todos los mensajes. Esta funcionalidad puede ser interesante, pero no en nuestro caso. Si perdemos, por decir una cifra, 10 lecturas durante un minuto, de nada nos sirve que se ejecuten más tarde 10 lecturas ya que las medidas serán las actuales, no las que hemos perdido anteriormente.

Jboss incorpora en su versión 7 *Infinispan*. Este viene a sustituir al sistema de caché *JBoss* Cache utilizado hasta ahora. En nuestra aplicación esto supondrá los siguientes cambios que van en línea con el nuevo sistema de espacio de nombres:

- *JBoss 4 + JBoss Cache*

```
PojoCacheMBean cache = ( PojoCacheMBean ) MBeanProxyExt.create( PojoCacheMBean.class,
"jboss.cache:service=DesaladoraPojoCache", server );
```

- *Jboss 7 + Infinispan*

```
@Resource(lookup="java:jboss/infinispan/container/telecontrol")
CacheContainer container;
Cache<Object, Object> cache;
```

El sistema de log en *Jboss 7* ha sido implementado con un servicio interno llamado *JBoss Logging*. Aunque es posible conservar el clásico *log4j* que ha sido usado hasta ahora, requiere de una configuración adicional. *JBoss Logging* usa internamente *log4j*. Por lo tanto el código se conservará igual, siendo el único cambio necesario la inclusión de la librería *JBoss Logging*.

Todos estos servicios requieren de una configuración que indique su funcionamiento en el servidor. En *Jboss AS 4* la configuración del servidor se encontraba dividida en varios ficheros y carpetas, dentro de la instancia donde la aplicación fuera ejecutada (*default* en este caso). *JBoss 7* centraliza toda la configuración en un solo fichero *xml*. Este se encuentra en la carpeta “/configuration” de la instancia del servidor. En nuestro caso usaremos una instancia “*standalone*” (autónoma, que no necesita conectarse con otros servidores) en la ruta “/<servidor-JBoss7>/standalone/configuration/standalone.xml”

En este fichero podemos configurar todos los aspectos del servidor y de los sub-sistema añadidos. En nuestro caso es especialmente interesante la configuración del sistema de log, de fuentes de datos y de mensajería *Java*.

- Subsistema de datos. - En él indicamos qué fuente de datos cargará el servidor para darle servicio a la aplicación. Observamos el uso del *MySQL* que incluye la configuración de acceso y el driver.

```
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
       <datasources>
              <datasource jta="true" jndi-name="java:jboss/datasources/MySQLDB"
      pool-name="MySQLDB" enabled="true" use-java-context="true" use-ccm="true">
                     <connection-url>jdbc:mysql://localhost:3306/</connection-url>
                     <driver>com.mysql</driver>
                     <security>
                            <user-name>root</user-name>
                            <password>1234</password>
                     </security>
                     <statement>
                      <prepared-statement-cache-size>100</prepared-statement-cache-
      size>
                        <share-prepared-statements>true</share-prepared-statements>
                    </statement>
              </datasource>
              <drivers>
                     <driver name="com.mysql" module="com.mysql"/>
              </drivers>
      </datasources>
</subsystem>
```

- Subsistema de mensajería *Java*. - Debemos declarar nuestras colas y *topics* de mensajes de manera que el servidor de aplicaciones las cree cuando arranca la aplicación.

```
<subsystem xmlns="urn:jboss:domain:messaging:1.1">
      <hornetq-server>
            <persistence-enabled>true</persistence-enabled>
                  <journal-file-size>102400</journal-file-size>
                  <journal-min-files>2</journal-min-files>
..........
<jms-destinations>
      <jms-queue name="AlarmsWarningsQueue">
            <entry name="telecontrol/queue/alarms/AlarmsWarnings"/>
      </jms-queue>
```

```
        <jms-queue name="UpdateCacheQueue">
                <entry name="telecontrol/queue/coordinator/UpdateCache"/>
        </jms-queue>
        <jms-topic name="blackBoardTopic">
                <entry name="telecontrol/topic/blackBoardTopic"/>
        </jms-topic>
        <jms-topic name="AlarmsHandLerTopic">
                <entry name="telecontrol/topic/alarms/AlarmsHandLer"/>
        </jms-topic>
</jms-destinations>
```

- Subsistema de *logging*. - Indica el funcionamiento de los registros del servidor. Podemos indicar el tipo de manejador (principalmente ficheros y consola) y la granularidad de los mismo.

```
<subsystem xmlns="urn:jboss:domain:logging:1.1">
        <console-handler name="CONSOLE">
                <level name="DEBUG"/>
                <formatter>
                        <pattern-formatter  pattern="%d{HH:mm:ss,SSS}  %-5p  [%c]  (%t)
                %s%E%n"/>
                </formatter>
        </console-handler>
        <periodic-rotating-file-handler name="FILE_P">
                <formatter>
                        <pattern-formatter  pattern="%d{HH:mm:ss,SSS}  %-5p  [%c]  (%t)
                %s%E%n"/>
                </formatter>
                <file relative-to="jboss.server.log.dir" path="server.log"/>
                <suffix value=".yyyy-MM-dd"/>
                <append value="true"/>
        </periodic-rotating-file-handler>
.........
<logger category="org.apache.struts2">
        <level name="ERROR"/>
                <handlers>
```

```
                <handler name="Telecontrol_struts2"/>
            </handlers>
            </logger>
        <logger category="net.wimpi.modbus">
            <level name="DEBUG"/>
            <handlers>
                <handler name="Telecontrol_modbus"/>
            </handlers>
        </logger>
        <logger category="smslib">
            <level name="ALL"/>
        </logger>
..........
        <root-logger>
            <level name="INFO"/>
            <handlers>
                <handler name="CONSOLE"/>
                <handler name="FILE_P"/>
            </handlers>
        </root-logger>
</subsystem>
```

Respecto al sistema de log es necesario añadir una nota sobre su importancia. El rastreo del funcionamiento del sistema así como de sus posibles errores es de vital valor para el mantenimiento del mismo. De otro modo sería imposible encontrar, trazar y aislar los problemas que pudieran aparecer. También indicar que no vale con registrar todo. Conviene investigar qué es necesario y qué no lo es, añadiendo la granularidad que la aplicación necesite.

Una vez la aplicación esté funcionando bajo *JBoss AS7* y cumpla con las especificaciones de *Java EE 6* podemos continuar con una nueva iteración.

3.3. Iteración 3

3.3.1. Introducción del módulo de gestión de hardware de adquisición de datos

Los módulos de adquisición de datos están incluidos en la lógica de negocio de los sensores. La idea es crear un módulo con funcionalidad similar al de gestión de sensores o actuadores que lo separe. En los siguientes diagramas se muestra el cambio en las clases:

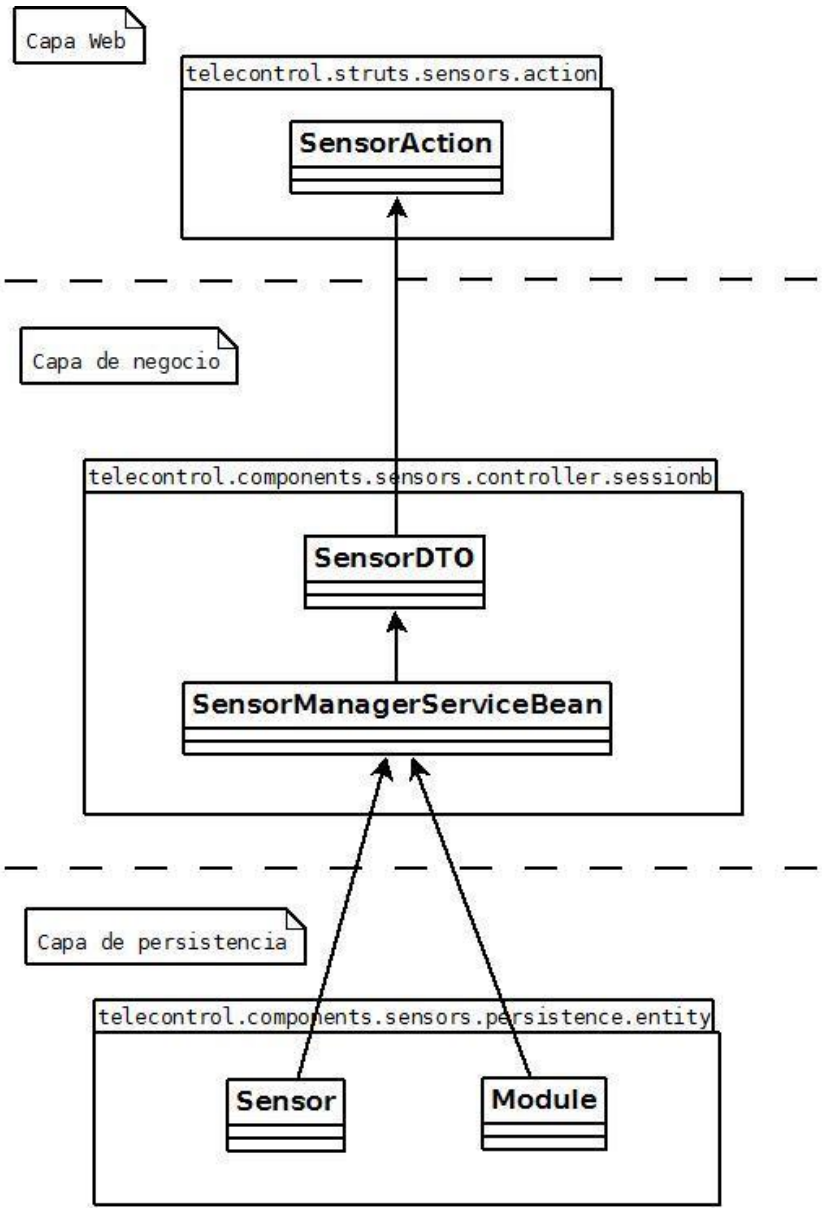

Ilustración 37: Diseño de clases sensor-módulo anterior.

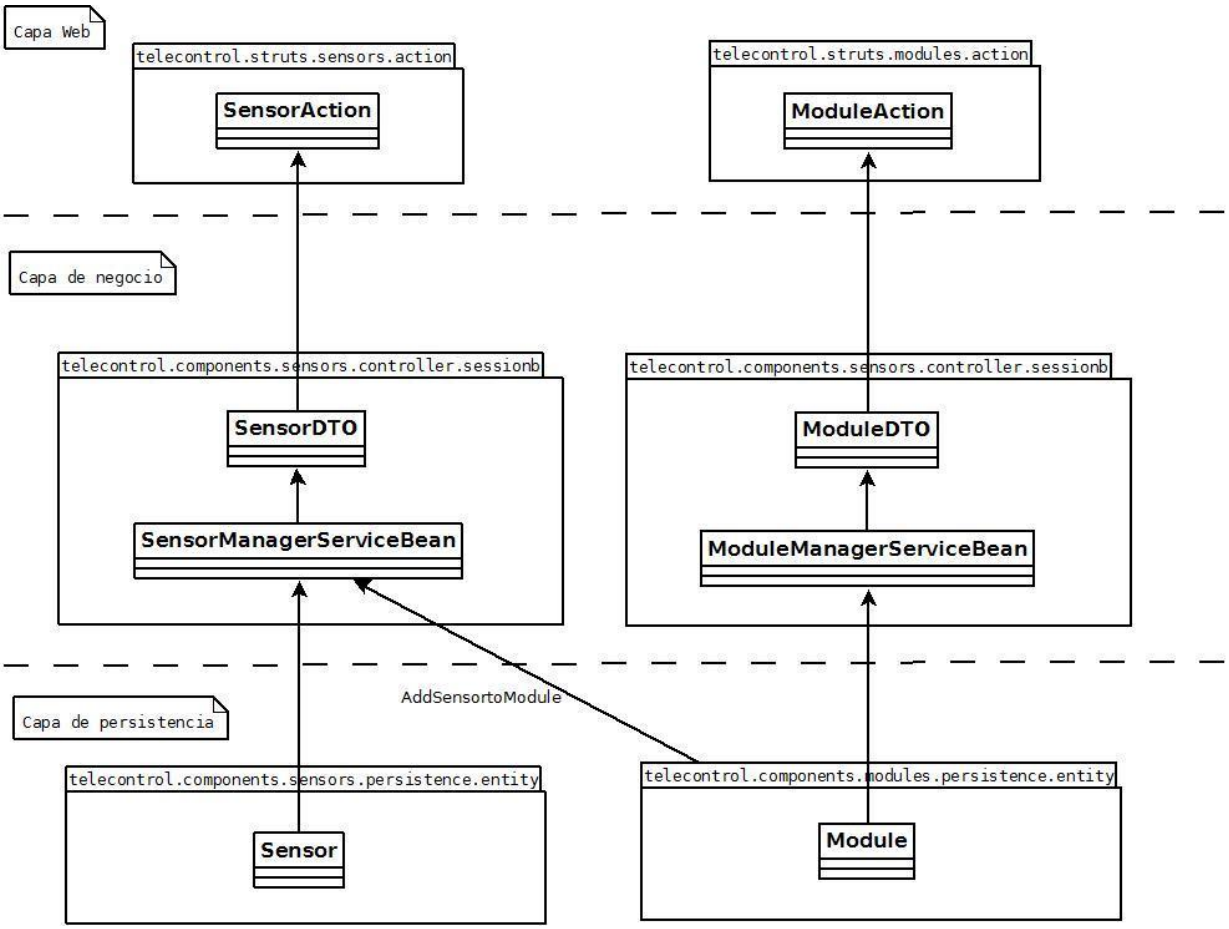

Ilustración 38: Nueva propuesta de diseño.

La refactorización separa parte de la funcionalidad del servicio de sensores al nuevo servicio de módulos. Las clases que antes invocaban a los sensores para tener acceso a los datos de los módulos deben ahora usar el servicio de módulos. A modo de ejemplo se incluye el método que relaciona los sensores (o actuadores) a una entrada (o salida) del dispositivo de adquisición de datos. Sería interesante además que el servicio de gestión de los módulos se fuera el responsable de la carga y utilización de los manejadores concretos para cada tipo de dispositivo. Ahora se usa siempre el mismo (“ModBusWrapperInterface” la clase envoltorio de la librería *JAMOD*), pero sería sencillo y conveniente añadir los métodos necesarios al servicio de módulos. De esta manera, se permitirá el uso en tiempo de ejecución de distintos protocolos y librerías.

3.3.2. Añadir los identificadores numéricos

En los objetos entidad de la aplicación se han omitido, en algunas ocasiones, el uso de identificadores numéricos. En algunos casos se han añadido y en otros no. También se observa que algunos identificadores numéricos corresponden con un código hash de las claves primarias que son ristras. Esta es una opción válida, pero no ha sido utilizada homogéneamente en el proyecto. A nivel de software, la identificación de un objeto por un número es mucho más sencilla que otro tipo de datos y/o la combinación de los mismos, como ocurre en la aplicación. A nivel de base datos, si se desea la unicidad de ciertos valores esto se expresó como restricciones semánticas adicionales (de unicidad por ejemplo). El cambio propuesto es utilizar claves numéricas en todos los objetos. A priori puede parecer un cambio sencillo pero requiere modificar el campo clave del objeto y por lo tanto la clave primaria de las tablas en la base de datos. Esto requiere cambiar cada línea de código donde se acceda a la clave del elemento en cuestión e indicar el acceso al id.

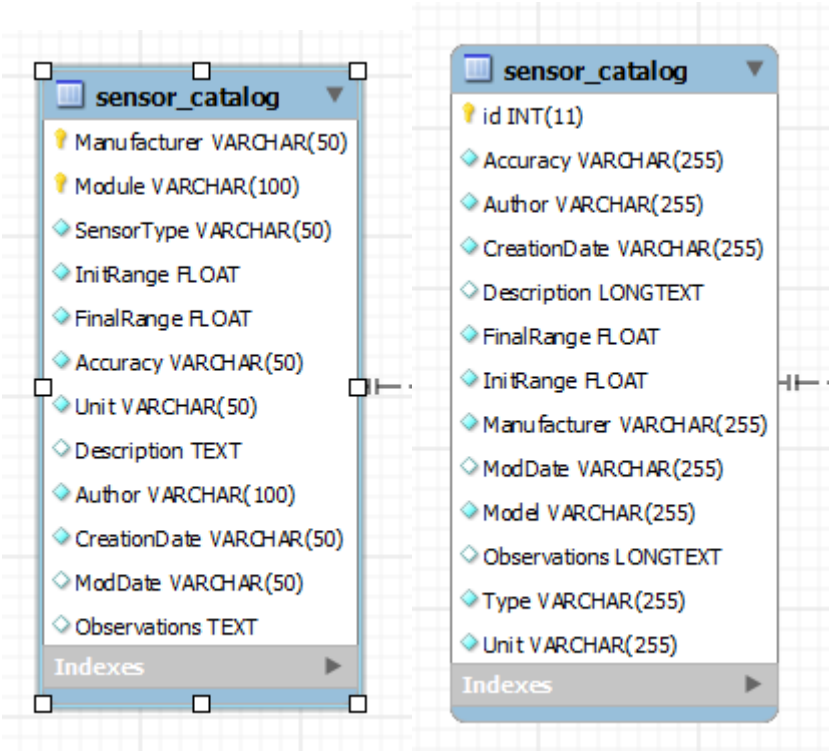

Ilustración 39: De par de claves ristra a identificador numérico.

Por lo tanto, debemos ir objeto a objeto inspeccionando donde es usado. Afortunadamente la mayorías de los *IDEs* como Eclipse nos ayudan en esta tarea

permitiéndonos cambios masivos de texto en diferentes documentos e indicándonos lo errores de sintaxis en el código.

SensorCatalog.java

```
36  */
37 @Entity
38 @Table(name = "sensor_catalog", schema = "telecontrol", uniqueConstraints =
39     @UniqueConstraint(columnNames = {"Manufacturer", "Model"})
40 )
41 @NamedQueries({
42     @NamedQuery(name = "SensorCatalog.findAll",
43         query = "SELECT s FROM SensorCatalog s ORDER BY s.id"),
44     @NamedQuery(name = "SensorCatalog.findByPrimaryKey",
45         query = "SELECT s FROM SensorCatalog s WHERE s.id = :id")
46     })
47
48 public class SensorCatalog implements Serializable {
49
50     private static final long serialVersionUID = 52712593325501488666L;
51
52     // Primary Key
53     @Id
54     @GeneratedValue(strategy=GenerationType.IDENTITY)
55     @Column(name = "id")
56     private Integer id;
57
58     @Column(name = "Manufacturer" , nullable = false)
59     private String manufacturer;
60     @Column(name = "Model" , nullable = false)
61     private String model;
62
63     // Rest of attributes
64     @Column(name = "Type", nullable = false)
65     private String type;
66     @Column(name = "InitRange", nullable = false)
67     private Float initRange;
68     @Column(name = "FinalRange", nullable = false)
69     private Float finalRange;
70     @Column(name = "Accuracy", nullable = false)
71     private String accuracy;
72
73     @Column(name = "Unit", nullable = false)
74     private String unit;
75     @Lob
76     @Column(name = "Description")
77     private String des;
```

Ilustración 40: Vista del objeto sensor con anotaciones JPA.

3.3.3. Cambios en las relaciones de las secciones.

El módulo de secciones se planteó como un modo de dividir una instalación que contuviera muchos elementos en diferentes partes más pequeñas. Un acierto para aquellas plantas que fueran demasiado grandes, donde la cantidad de elemento se vuelve inmanejable. Se añadió la restricción de que cada elemento sólo podía pertenecer a una sección. Tal decisión llevó a crear la relación entre el elemento (módulo, alarma, sensor, etc) y la sección como “muchos a uno” y en sentido contrario. Por lo tanto se tomó la decisión de añadir un campo a cada elemento que indicara la sección.

Esta decisión presenta un error de diseño. Primero cabe destacar que un elemento puede no estar asociado a una sección con lo cual el valor en tal campo no está definido. El campo sección no añade ninguna información vital al elemento. Se tuvo que modificar todas y cada una de las clases y tablas ya existente para añadir este cambio. Además y lo que es más importante, es un cambio en cierto sentido “irreversible”, que no admite cambios futuros. Si nosotros decidiéramos permitir que un elemento estuviera al mismo momento en más de una sección por exigencias del cliente, este diseño no sirve. También ocurre que si decidiéramos eliminar las secciones, por el motivo que fuera, debemos modificar todas las tablas.

JPA permite las relaciones uno a muchos pero como un ejercicio de retro compatibilidad. No añade lógica para el tratamiento de las operaciones en cascada (que hacer cuando una parte de la relación es borrada o modificada) para este tipo de relaciones. Por ello deja la implementación del comportamiento de este funcionamiento al programador.

La modificación propuesta es pasar a una relación muchos a muchos donde se crea una tabla relación entre el elemento y la sección. Si introducimos una restricción de unicidad entre la clave del elemento de esta tabla, conseguimos el comportamiento de una relación uno a muchos ya que no puede existir el mismo elemento relacionado con más de una sección.

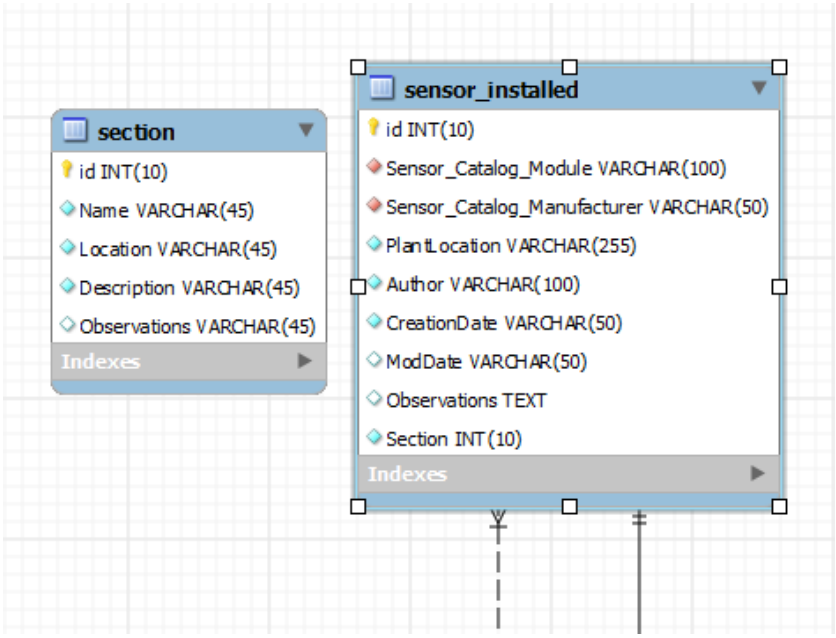

Ilustración 41: Antigua relación sensor-sección.

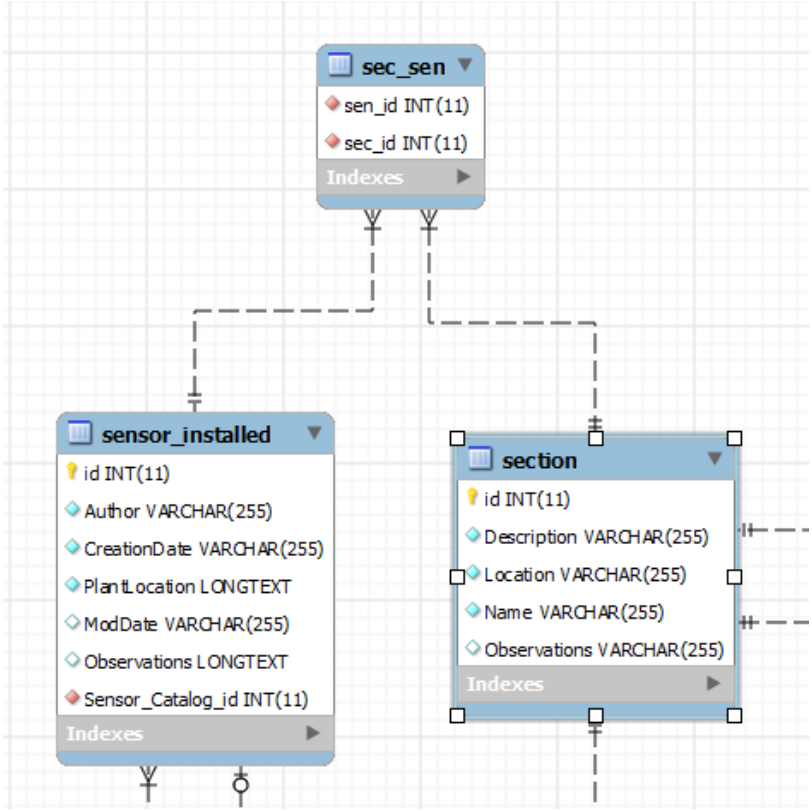

Ilustración 42: Nueva propuesta muchos a muchos.

Con este nuevo diseño bastaría con eliminar la restricción de unicidad para la clave del elemento, en este caso del sensor en la tabla “sec_sen” o sección-sensor, para permitir que un elemento esté en varias secciones. Además, si eliminamos las secciones, los elementos no se ven afectados. También las búsquedas se simplifican. Ahora si queremos todos los elementos de una sección solamente necesitamos buscar los id de estos, para luego, consultarlos uno a uno.

3.3.4. Actualización a *Struts 2* y cambio de diseño en la capa web

El mayor rediseño en la aplicación se concentra en esta sección. Debemos no solo actualizar a *Struts2* sino también proponer un cambio en el manejo de la interfaz que mejore la usabilidad de la misma.

3.3.4.1. Cambio de diseño en clases

Con el cambio a *Struts 2* toda la lógica del procesamiento web puede ser más fácilmente concentrada en una sola clase acción para cada uno de los diferentes módulos que componen la aplicación. Esto era posible también en *Struts 1*, añadiendo el método a ser llamado dentro de los ficheros de configuración *struts.xml*. Ahora, gracias a las anotaciones, toda la información de una acción se concentra en una sola clase. Las anotaciones permiten definir las acciones de *Struts* asociadas a un método. No solamente al método por defecto *execute()*. Por lo tanto pasamos de este esquema:

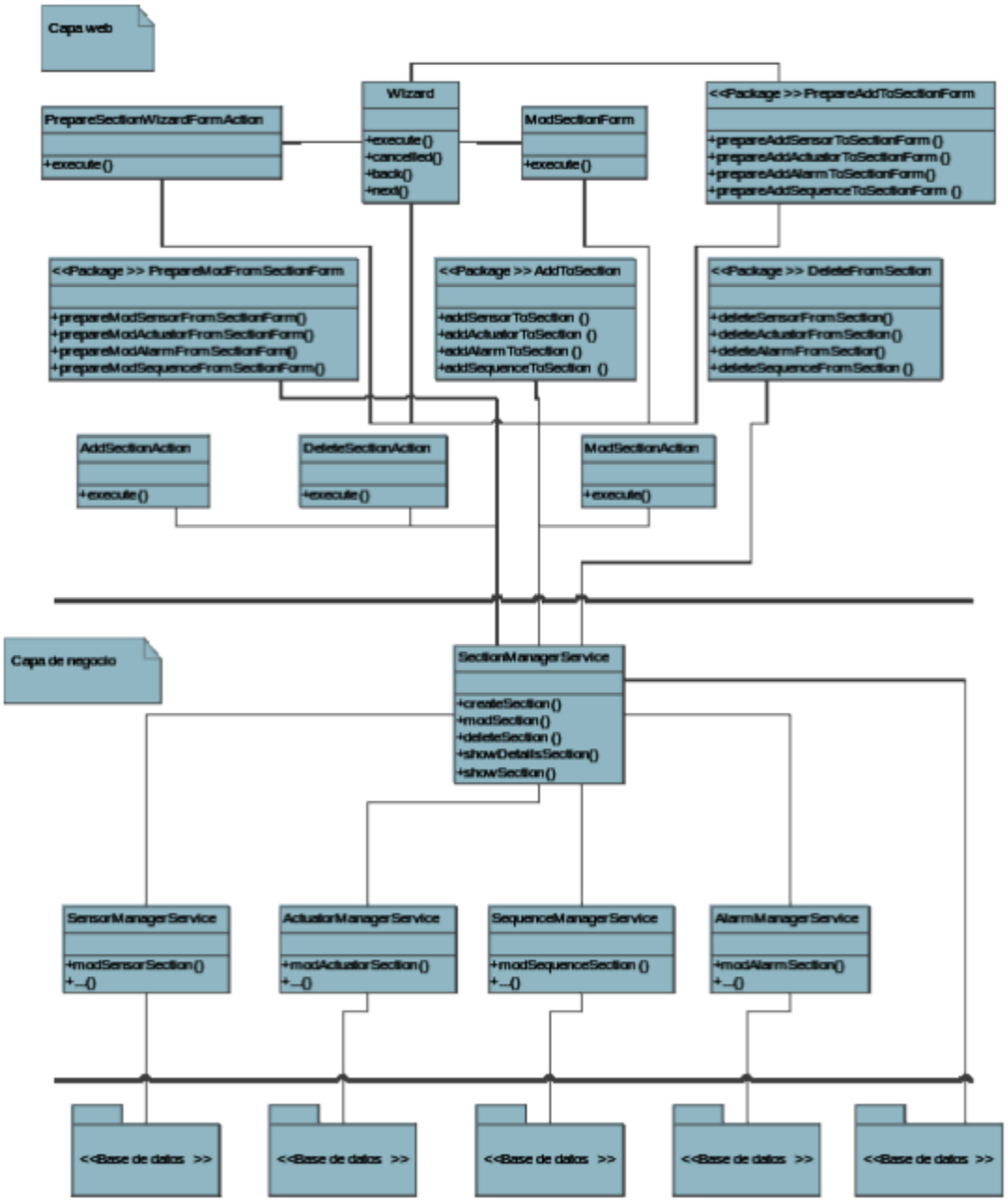

Ilustración 43: Diagrama de clases del gestor de secciones (5.3.5 [3]).

A esta nueva organización que será válida para todos los módulos:

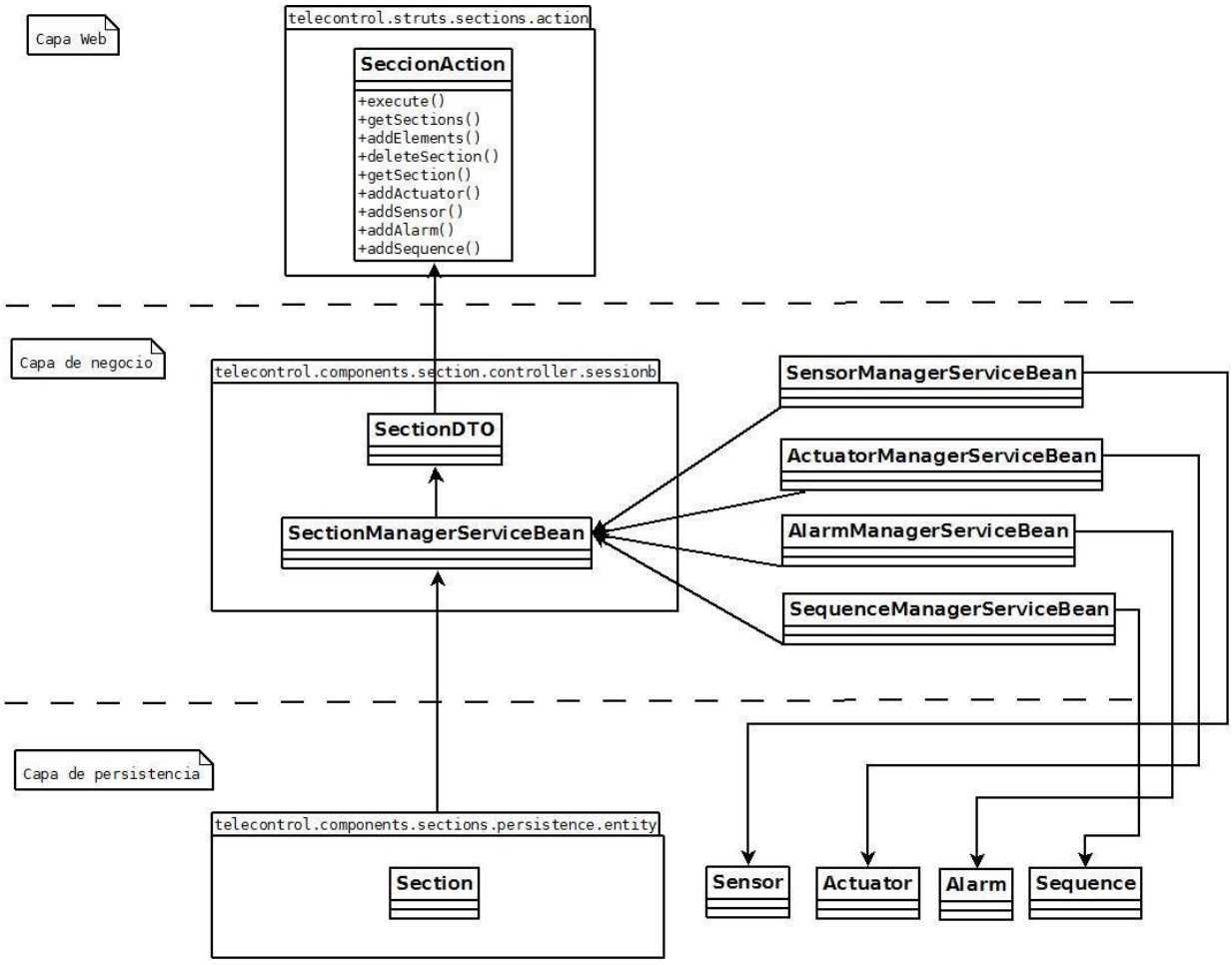

Ilustración 44: Nuevo diagrama de clases del gestor de secciones.

3.3.4.2. Actualización de los wizards.

En *Struts 2*, gracias a *OGNL*, no son necesarias las clases que gestionan los formularios ya que los datos son leídos por un interceptor. *OGNL* es el acrónimo de "*Object Graph Navigation Language*", un lenguaje de expresiones muy poderoso que nos permite leer valores de objetos Java. Este lenguaje nos permite leer valores y ejecutar métodos (que regresen algún valor) para mostrar los valores o resultados de los mismos en nuestras páginas *JSP* creadas usando las etiquetas de *Struts*. Además proporciona una conversión automática de tipos que permite convertir datos desde texto *HTTP* a objetos *Java*. Esto significa que desde los ficheros *JSP* tendremos acceso a los atributos de las acciones.

Los *Wizard* no existen en *Struts 2* pero es importante conservar esta funcionalidad en la nueva interfaz ya que nuestros formularios están compuestos por numerosos y variados elementos. En *Struts 2* se hace necesario de un *plugin* que conserve los datos entre distintas peticiones *HTML*. Básicamente lo que necesitamos es guardar estos datos en la sesión del *Servlet* y pasar un id que mantenga identificadas estas variables entre formulario y formulario.

Esto es posible gracias a "*Struts 2 Conversation Plugin*" (literalmente conversación en español). Este *plugin* permite mediante anotaciones definir qué acción inicia una conversación, para la cual genera un id asociado a un nombre. Asociamos también los campos que deben ser conservados a esta conversación. Mediante el uso de una librería de etiquetas específico, los formularios generados en *JSP* añaden la identificación de la conversación indicada en la acción, pudiendo así conservar los datos entre llamada y llamada.

SectionsAction.java

```
package telecontrol.struts.administration.sections.action;

import java.util.ArrayList;

@Namespace("/administration/sections")
@ParentPackage("telecontrol-default")
public class SectionsAction extends ActionSupport implements ModelDriven<SectionView>, SessionAware {

    private static final long serialVersionUID = 6041596332522313604L;

    static final Logger logger = Logger.getLogger("telecontrol.struts.administration.sections");

    @Inject
    private SectionManagerServiceLocal sectionManager;
    @Inject
    private SensorManagerServiceLocal sensorManager;
    @Inject
    private ActuatorManagerServiceLocal actuatorManager;
    @Inject
    private AlarmManagerServiceLocal alarmManager;
    @Inject
    private SequenceManagerServiceLocal sequenceManager;

    @ConversationField(conversations = "add")
    private SectionView sectionView = new SectionView();

    Map<String, Object> session;

    private List<SectionView> sectionList = new ArrayList<SectionView>();
    private List<SensorInstalledView> sensorsInstalledView = new ArrayList<SensorInstalledView>();
    private List<ActuatorInstalledView> actuatorsInstalledView = new ArrayList<ActuatorInstalledView>
    private List<AlarmCatalogDTO> alarmsInstalledView = new ArrayList<AlarmCatalogDTO>();
    private List<SequenceDTO> sequencesInstalledView = new ArrayList<SequenceDTO>();

    @Override
    @Actions({
        @Action(value="index", results={@Result(name=SUCCESS, location="mainLayout.administration.sec
      })
    public String execute() throws Exception {
        return SUCCESS;
    }

    @BeginConversation(conversations = "add")
    @Action(value="add", results={@Result(name=SUCCESS, location="/jsp/administration/sections/sectio
    public String add() throws Exception {
        return SUCCESS;
    }

    @ConversationAction(conversations = "add")
    @Actions({
        @Action(value="addSensor", results={@Result(name=SUCCESS, location="/jsp/administration/secti
        @Action(value="addActuator", results={@Result(name=SUCCESS, location="/jsp/administration/sec
        @Action(value="addAlarm", results={@Result(name=SUCCESS, location="/jsp/administration/sectio
        @Action(value="addSequence", results={@Result(name=SUCCESS, location="/jsp/administration/sec
    })
    public String addSensor() throws Exception {
        return SUCCESS;
    }

    @ConversationAction(conversations = "add")
    @Action(value="addElements", results={
```

Ilustración 45: Ejemplo de uso de las etiquetas *Struts2 conversation plugin*.

3.3.4.3. Esquema de vista

Nuestro nuevo interfaz estará basado en dos elementos principales: tablas y ventanas. Las tablas serán el elemento de entrada a cada uno de los módulos y las ventanas nos

permiten su modificación. Este diseño es posible gracias la combinación de *JQuery*, *AJAX* y *JSON*.

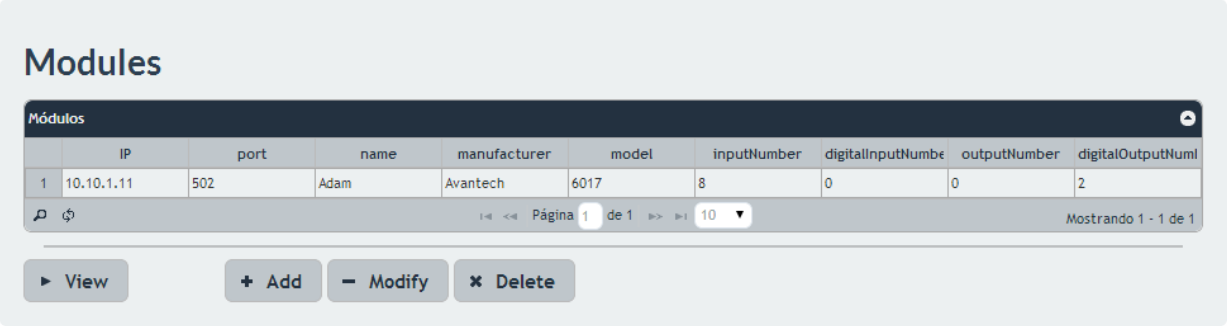

Ilustración 46: Ejemplo de las tablas.

Cada elemento de la aplicación tendrá la misma disposición. En la ventana principal veremos una tabla que lista los elementos y los botones que realizan las operaciones “añadir, modificar, eliminar y ver”. Será posible seleccionar un ítem de la tabla haciendo *click* sobre él o múltiples en caso de que sea necesario y/o posible. Todas las tablas añaden paginación con funcionamiento mediante *JavaScript*. Esto es especialmente interesante cuando disponemos de tablas realmente grandes como en los registros de históricos. Anteriormente era necesario recargar toda la página para ir mostrando todos los datos de una tabla.

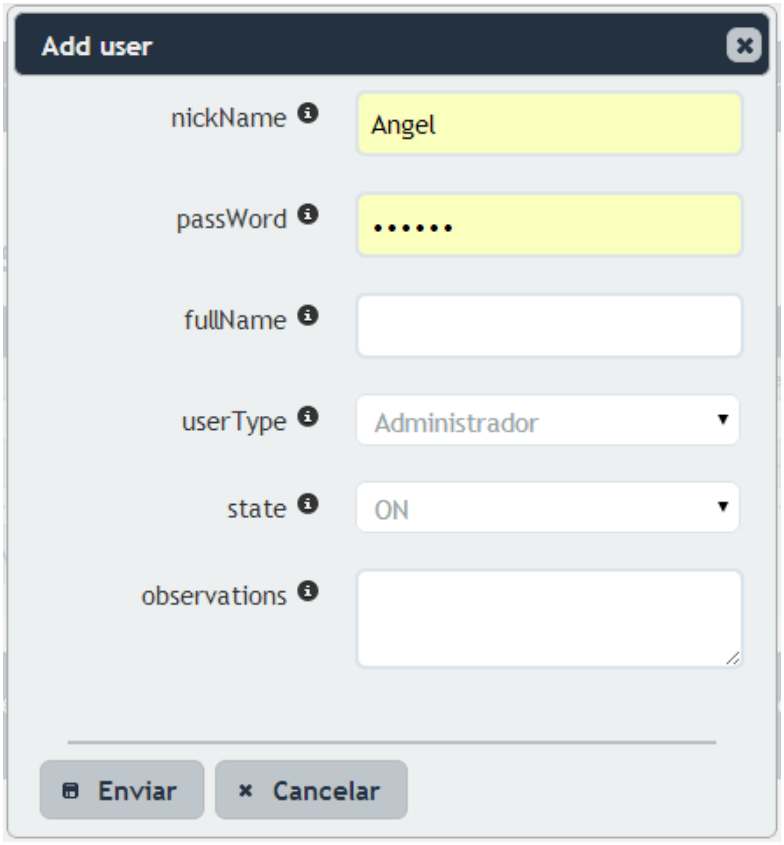

Ilustración 47: Ejemplo de las ventanas emergentes.

Las ventanas serán elementos emergentes dentro del mismo navegador. También son conocidos como “pop-up”. En ellas podremos realizar todas las operaciones definidas sobre los elementos pertenecientes a los módulos de la aplicación.

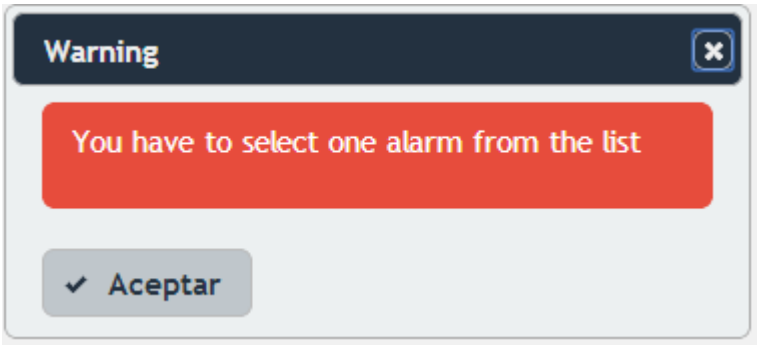

Ilustración 48: Retroalimentación en las ventanas.

En la versión anterior cada módulo dispone de un acceso por funcionalidad, esto es, existía una vista en los sensores para añadir, otra para modificar, otra para eliminar y para consultar. Este sistema de navegación resulta engorroso. Cada una de las vistas mencionadas contiene una tabla con los sensores sobre los que se puede realizar la acción, excepto añadir que muestra un formulario. La nueva disposición permite con un solo clic acceder a una única vista que ofrece una visión global de los datos. Además gracias al sistema de ventanas podemos mantener a la vista del usuario en la misma página, los datos de los elementos existentes en el sistema.

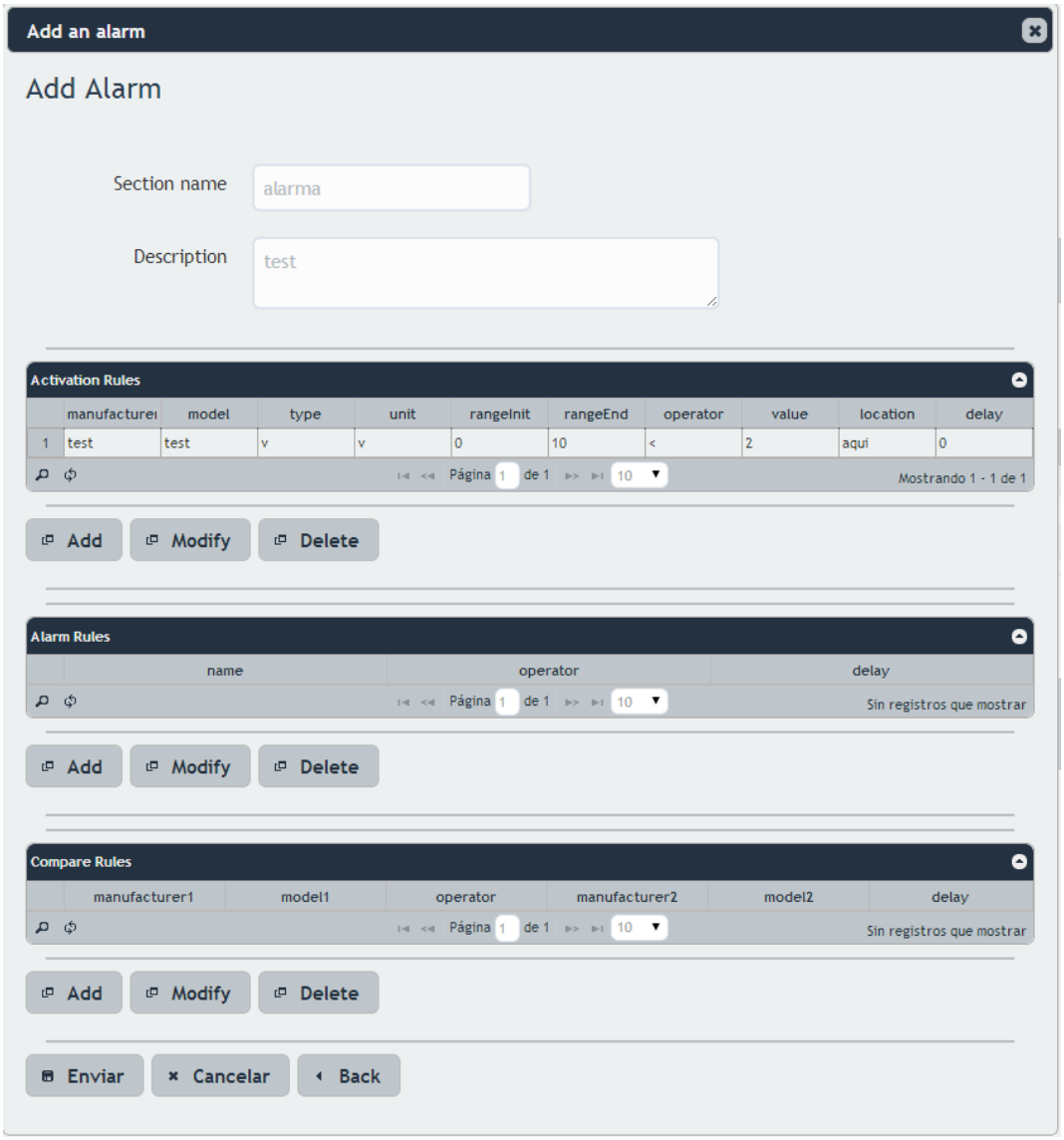

Ilustración 49: Combinación ventana-tablas.

Esta propuesta reduce el número de páginas en la aplicación así como el código *JSP*. La mayoría de elemento se repetirá de una página a otra permitiéndonos crear fragmentos *Tiles* que serán utilizados por múltiples páginas. Permite que su uso sea más sencillo para el usuario sin reducir en ningún momento la funcionalidad.

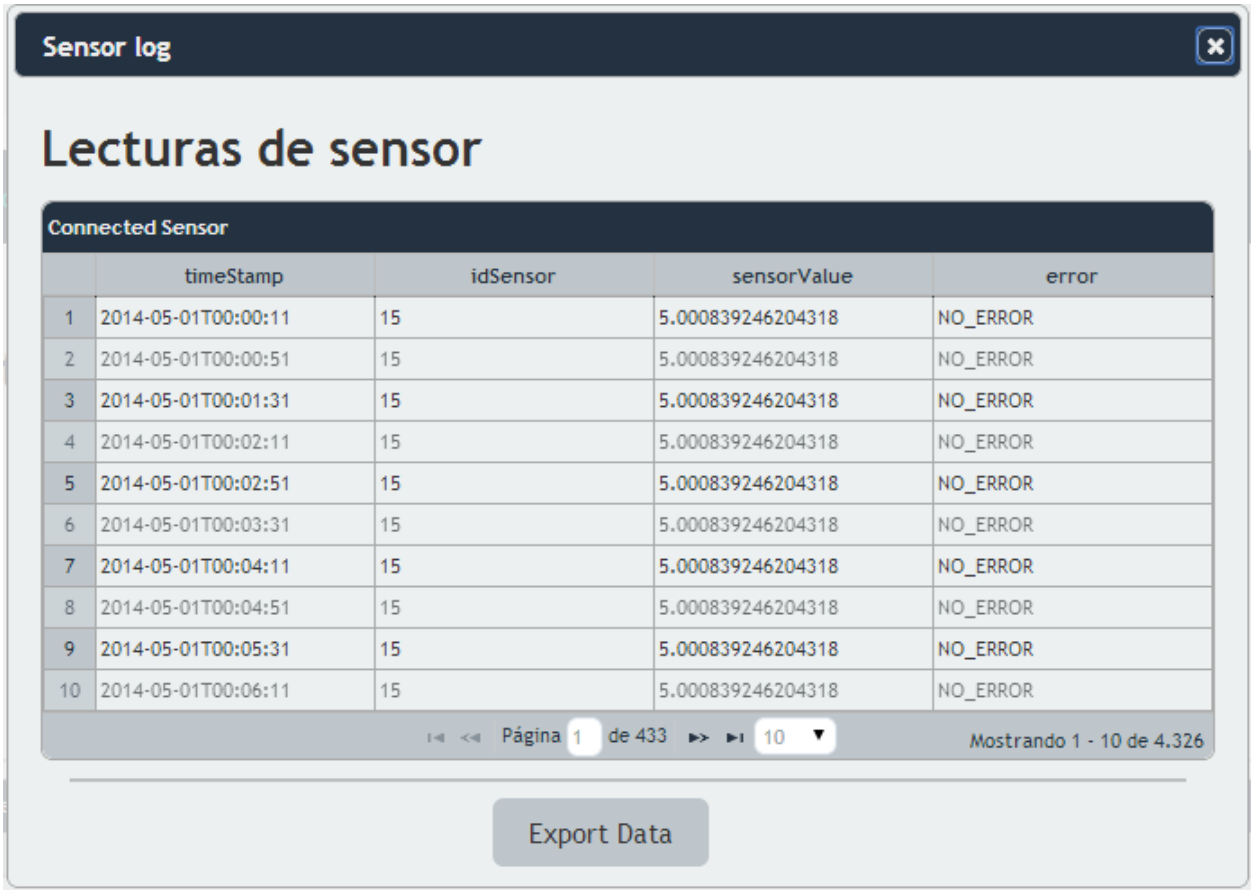

Ilustración 50: Vista del registro de históricos.

3.4. Iteración 4

3.4.1. Elección del estilo

El diseño de nuestra nueva interfaz pretende ser minimalista, donde predominan los espacios, y en donde los datos y las operaciones que podemos hacer con ellos sean los protagonistas. Se ha decidido por otra parte conservar el esquema de secciones anterior con la intención de no romper definitivamente con él, permitiendo que la adaptación al nuevo diseño sea más cómoda. Además, debido al tipo de aplicación, los datos que se manejan y las funciones que ésta posee, parece que la disposición de los elementos en la ventana del diseño anterior es el más adecuado.

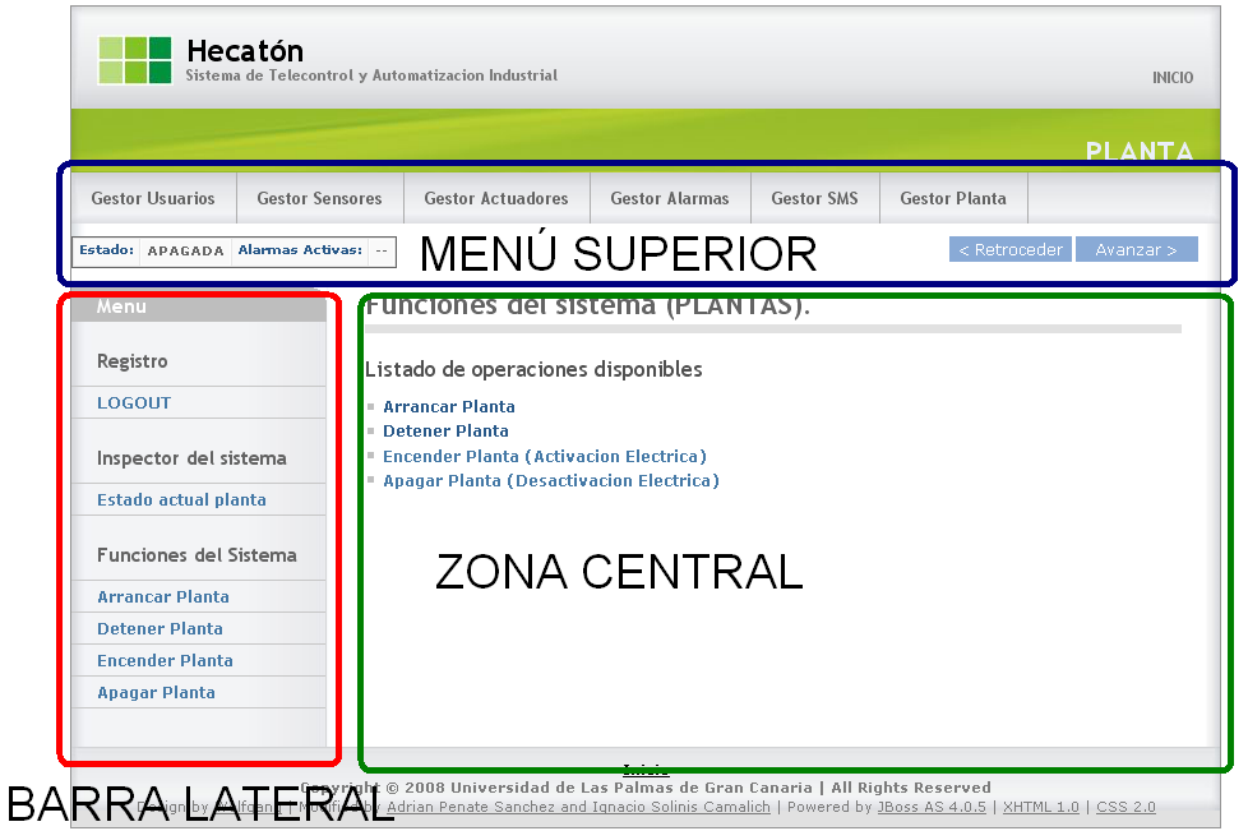

Ilustración 51: Esquema de vista de la versión anterior.

Ilustración 52: Nuevo diseño.

El nuevo diseño se construirá utilizando el *framework Bootstrap* en su versión 2.3.2. Se ha utilizado esta versión ya que existe un *plugin* para *Struts 2* que facilita la integración, pero este solamente da soporte para la ya mencionada versión de *Bootstrap*. En la versión actual, *Bootstrap 3*, se ofrece también código *Javascript* que implementa tablas y ventanas emergentes al igual que *JQuery UI*. Con lo cual sería interesante su adopción en un futuro. Por el momento y al estar *JQuery* y su *plugin* para *Struts 2* mucho más maduro, mantendremos las tablas, los enlaces y botones *AJAX*, las gráficas, y las ventanas emergentes en *JQuery*. Esto tiene una consecuencia directa y es que no todo el diseño será 100% "responsive design". Sin embargo como ya hemos mencionado, con una pequeña actualización, sería posible.

Para la elección de los colores, fuentes y demás elementos del sitio web se ha pensado en el uso de un tema oficial de *Bootstrap*. Concretamente el tema "Flaty" de *Bootwatch* (http://bootswatch.com/flatly/). Estos temas ofrecen una licencia de software libre "*MIT*" y son mantenidos por la comunidad. Cumplen con los estándares web, las definiciones de *Bootstrap*, son adaptables a diferentes dispositivos y modulares. Lo más importante es que, puede ser intercambiado fácilmente por otro diseño reemplazando su fichero *CSS* o modificando el mismo. Esto nos permite tener diseños adaptativos para la aplicación sin esfuerzo.

Ilustración 53: Diseño adaptado a dispositivos móviles

Para que esto último sea posible se deben introducir las mínimas modificaciones manuales en los ficheros *CSS*. Por desgracia *Bootstrap* y *JQuery* son incompatibles. Para solucionar este problema se ha creado un tema para *JQuery* adaptado con los elementos presentes en el tema "*Flaty*" de *Bootstrap*. De esta manera, los elementos *JQuery* como las tablas y los botones utilizan los mismos colores, las mismas fuentes y los mismos tamaños que el diseño en *Bootstrap*. Sin embargo para que todo funcione correctamente, se añaden algunos "*hacks*" en las clases *CSS* de *JQuery UI*. Para modificar el diseño de *JQuery* se deberán tener en cuenta las pequeñas modificaciones realizadas y que se encuentran todas localizadas en un mismo fichero *styles.css*.

3.4.2. Actualización de la librería *SmsLib*

Durante los proyectos anteriores se ha mantenido la versión 2.1.2 de esta librería cuando está próxima la liberación de la versión 4. La librería *SmsLib* permite enviar mensajes de texto a través de dispositivos *GSM* conectado a la red móvil. Encapsula y da soporte a una multitud de dispositivos de este tipo, ya sea con comunicación serie o *TCP/IP*.

Hasta ahora se ha venido utilizando el controlador para puertos seriales de java *JavaComm v2*. Si bien funciona, este controlador es solamente compatible con sistemas *Windows* de arquitectura *x86* (32 bits). De esta manera el sistema debe prescindir del módulo de envío y gestión *SMS* si se desea poner en funcionamiento la plataforma en otros sistemas.

En este proyecto se utilizará la versión estable 3.5.4. Entre las ventajas cabe destacar que con la nueva versión es posible utilizar el módem *GSM* en sistemas de 64 bits. Esto si requerirá cambiar el controlador para el dispositivo a *Rxtx*. *Rxtx* permite la comunicación con dispositivos serie utilizando *Java* para distintas plataformas. En *Java*, debido a que es un lenguaje independiente de la plataforma donde se ejecute, resulta complicado la estandarización de controladores para estos puertos en los diferentes sistemas. *Sun*, el desarrollador original de *Java* (recordemos que *Sun* y todos sus proyectos fueron comprados por *Oracle* en 2009), no puso demasiada atención en el desarrollo de soporte para comunicación serial. Se limitó a lanzar la *API JavaComm* en 2005, que como ya hemos dicho, solamente ofrecía soporte para sistemas *Windows* en 32 bits.

El uso de los controladores *JavaComm* o *Rxtx* es transparente para la nueva versión de la librería *SmsLib*. Esta intentará cargar los primeros y si no es posible los segundos. Por lo tanto no debemos cambiar nuestro código fuente si necesitamos ejecutar la aplicación en *Windows* o *Linux* por ejemplo. Lo que sí será necesario, es añadir los controladores a la instalación de *Java* en nuestro sistema.

La sintaxis de la nueva versión de *SmsLib* cambia ligeramente pero basta con unos pocos ajustes de nombres en los objetos para que funcione correctamente. Sin embargo,

durante este cambio se observó un funcionamiento anómalo en el código. Cuando el sistema se encuentra en monitorización, si se produce una alarma, o se desactiva, el sistema siempre intenta enviar un mensaje, aun cuando la alarma no tenga asociado un mensaje o número al que enviar. Sin embargo este no el peor error. Cada vez que se intenta enviar un mensaje, se crean todos los objetos necesarios para el envío (clase *singleton Service* de la librería *SmsLib*). Además se conecta y desconecta de la red para cada alarma. Si tenemos 5 alarmas activas, nos conectamos y desconectamos 5 veces. El proceso de conexión a la red *GSM* con el módem *WaveCom* del que se dispone requiere de más 1 minuto según la cobertura. Esto quiere decir que cada mensaje será enviado con retraso. Esto ocurre de la misma manera para las actuaciones y consultas sobre la planta por *SMS*, donde un "*timer*" se ejecuta cada cierto tiempo para comprobar si hay mensajes y responder a las órdenes si fuera necesario. Esto no es aceptable. Si la aplicación gestionará una instalación con muchos elementos donde ocurrieran alarmas cada cierto tiempo el sistema *Sms* colapsaría.

Para solucionar este problema se ha seguido las recomendaciones presentes en la documentación de la librería *SmsLib* [23] que indican no cerrar la conexión. Por lo tanto se ha movido el código de la conexión al proceso de arranque de la monitorización. De esta manera el envío de mensajes se realiza instantáneamente (siempre y cuando la red lo permita).

3.4.3. Corrección del problema del tamaño de ristras en alarmas y secuencias

Otro detalle importante es el uso de ristras de caracteres como el modelo de datos para las reglas de las alarmas y los pasos de los sensores. Resulta incomprensible que la definición de estas se realice con formularios, que después se convierten en ristras para ser almacenadas en la *BBDD*. Posteriormente se deben "*tokenizar*" (convertir en tokens analizando la ristra) y de nuevo convertir en objetos con campos separados para su manipulación en los manejadores de alarmas y secuencias en el núcleo de la aplicación. Este es un problema de diseño que debe abordarse aunque no es crítico. Por el contrario, sí es necesario cambiar el tipo de dato de mapeo de tales ristras de caracteres en la base de datos. Estas están definidas como ristras de hasta 256 caracteres. Si las reglas o pasos son extensos (y no necesariamente demasiado) estas no pueden ser añadidas.

Resulta sencillo corregir este error. Solamente es necesario añadir una anotación sobre los campos necesarios, esto es, en las reglas de las alarmas y los pasos de las secuencias. Esta anotación es ”@Lob”, que indica al motor de persistencia que este campo en la base de datos es una ristra larga. Afortunadamente el motor de persistencia *Hibernate* realizará el cambio en las tablas por nosotros siempre y cuando lo indiquemos en la configuración de persistencia *JPA*. Debemos añadir la propiedad “*hibernate.hbm2ddl.auto*” en el fichero *persistence.xml* como indicamos a continuación:

```
<persistence-unit name="SensorService">
	<properties>
		<property name="hibernate.default_schema" value="telecontrol" />
		<property name="hibernate.hbm2ddl.auto" value="update" />
		<property name="hibernate.ejb.cfgfile" value="hibernate.cfg.xml"/>
	<!-- hibernate.hbm2ddl.auto Automatically validates or exports schema DDL to
	the database when the SessionFactory is created. With create-drop, the
	database schema will be dropped when the SessionFactory is closed explicitly.
	e.g. validate | update | create | create-drop | none So the list of possible
	options are, validate: validate the schema, makes no changes to the database.
	update: update the schema. create: creates the schema, destroying previous
	data. -->
	</properties>
</persistence-unit>
```

4. Implementación

En esta sección se enumeran y describen algunos de los detalles que se han considerado como más relevantes en lo que se refiere a la implementación de la aplicación, así como las herramientas utilizadas en el desarrollo. Aunque parte de las herramientas han sido heredadas de los proyectos anteriores, se han introducido considerables mejoras. El fruto de esta dura fase del proyecto, es decir, el código fuente desarrollado, se puede encontrar en la carpeta "/telecontrol/workspace/telecontrol" del DVD adjunto a la misma.

4.1. Herramientas utilizadas

Nos encontramos ante una aplicación bastante compleja que incluye más de 40.000 líneas de código. Utilizamos diferentes *frameworks* y tecnologías que deben funcionar conjuntamente. Por esto es necesario el uso de las herramientas de desarrollo software adecuadas. De otra manera el mantenimiento de la aplicación sería inabordable.

4.1.1. *Eclipse*

Se trata de una plataforma de desarrollo de aplicaciones de código abierto basada en Java. Su principal ventaja es la utilización de *plugins* para añadir funcionalidad a este *IDE*. Durante este proyecto se ha utilizado la versión con nombre "*Juno*" con soporte para aplicaciones *Java EE*. Esta edición incluye toda una serie de *plugins* que facilitan el desarrollo, construcción y despliegue de aplicaciones *Java EE* en servidores de aplicaciones.

Además se han utilizado los siguientes *plugins* que considero imprescindibles para aumentar el control y la productividad en el desarrollo de aplicaciones sobre *JBoss*:

- ***Jboss Tools*** - Se trata de una colección de herramientas que facilita el desarrollo de aplicaciones basadas en este servidor de aplicaciones. Entre los *plugins* más destacados encontramos:
    - ***Hibernate tools***, que permite la interacción con la base de datos desde un proyecto *JPA*.
    - ***JBoss AS Tools***, que nos facilita el despliegue y testeo de aplicaciones en el servidor. Nos permite por ejemplo el despliegue automático de la aplicación después del empaquetado.
    - ***Maven Tools***, ofrece integración de eclipse con *Maven*, pudiendo realizar todas las tareas del ciclo de vida desde la interfaz gráfica.

- ***EGit*** - Este *plugin* permite utilizar la conocida herramienta libre de control de versiones *GIT* integrada en la interfaz gráfica de *Eclipse*.

- ***JRebel*** - Se trata de una utilidad que nos permite visualizar "en caliente" los cambios realizados en el código de nuestra aplicación directamente en la interfaz web. Cada vez que realizamos un cambio en el código debemos volver a empaquetar y desplegar la aplicación, por muy insignificante que este cambio sea, para ver el resultado del mismo. Esto hace que el desarrollo de aplicaciones *Java EE* sea en ocasiones lento y tedioso. *JRebel* actualiza automáticamente aquellas clases y ficheros que han sido modificados en la instancia del servidor de aplicaciones. Se trata del único software no libre que se ha utilizado en este proyecto. *JRebel* pertenece a la compañía estonia *ZeroTurnAround*. Aunque existe alguna alternativa libre como *DCEVM*, no ofrecen la funcionalidad y sencillez de *JRebel*. Se ha utilizado una licencia privada personal que permite el uso de *JRebel* sin ánimo de lucro. *JRebel* no será incluido con el DVD que adjunta el proyecto pero creo necesario recomendar su uso a futuros desarrolladores.

4.1.2. *MySQL Workbench*

Se trata de una herramienta visual de diseño de bases de datos que integra, desarrollo de software, administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos *MySQL*. Es el sucesor de *DBDesigner 4* de *fabFORCE.net*, y reemplaza el anterior conjunto de software, *MySQL GUI Tools Bundle*.

Es especialmente recomendable para la configuración del servidor de bases de datos, la importación y exportación de esquemas y datos, y para el diseño. Aunque con *JPA* no es necesario realmente el esquema de base de datos ya que las clases lo describen. Con *Workbench* podemos generarlo completamente desde las tablas de un esquema.

4.2. Descripción de la implementación

4.2.1. Dificultad en la comprensión del código fuente

El estado del código fuente preexistente era caótico. Ha resultado muy complicado comprender a qué parte de la aplicación corresponde cada una de los directorios que forman la aplicación.

Además se han encontrado multitud de carpetas, ficheros, clases java y páginas *JSP* que no cumplían ninguna función dentro de la aplicación. Parecían ser restos de pruebas, proyectos enteros de prueba dentro del código fuente. Además existían multitud de métodos dentro de las clases que no eran utilizados en ningún momento dentro de la aplicación. Este ha sido el principal hándicap que ha motivado el cambio de estrategia y objetivos del presente proyecto de fin de carrera.

En la medida de lo posible se ha continuado el esquema de desarrollo marcado en proyectos anteriores, conservando fielmente la separación entre módulos original.

4.2.2. Falta de continuidad entre ciclos de desarrollo

Las intenciones iniciales de la primera versión del proyecto, comparadas, con la segunda y la tercera son muy distintas. Existe funcionalidad desarrollada en la primera versión que, a día de hoy, no existe o no funciona. Sin embargo el código continúa estando presente en la aplicación a pesar de no hacer nada. Ha sido especialmente complicado reconocer la funcionalidad real de la aplicación que se ha debido, durante este proyecto, revisar y comprobar.

Con este último ciclo de desarrollo, si bien se rompe con mucha de las costumbres anteriores, se intenta desde un principio definir un sistema de desarrollo software moderno y fácil de mantener.

4.2.3. Lento desarrollo de *plugins* de *Struts 2*

Struts 2 permite incluir nuevas funcionalidades a través de *plugins*. Existen infinidad de *plugins* desarrollados por la comunidad y ofrecidos libremente. Sin embargo el desarrollo es lento y poco fiable. Durante la realización de este proyecto he debido actualizar constantemente ciertos *plugins* para obtener la funcionalidad deseada o corrección de errores ya conocidos.

Especialmente destacado es el caso de los *plugins* para *JQuery* y *Bootstrap*. Como se ha mencionado anteriormente en este documento, ha sido necesario utilizar la versión 2.3.2 de *Bootstrap* aun cuando existe desde hace tiempo la versión 3. Debido a esta razón el uso de *JQuery* ha sido imprescindible. Teniendo en cuenta que ambas librerías son incompatibles, se ha hecho un gran esfuerzo para hacerlas funcionar al mismo tiempo. El *plugin* de *JQuery* para *Struts 2* ha sido un quebradero de cabeza. Hasta la última versión 3.7.0, no se han corregido errores en ciertos aspectos como las tablas (*JGrid*) que eran imprescindible para el desarrollo de la interfaz.

4.2.4. Facilidad de actualización de la interfaz

Se ha prestado especial atención a la hora de desarrollar la nueva interfaz, conservando los elementos estándar que permitan un cambio completo de la misma. Aplicando distintos temas para *Bootstrap* y *JQuery*, podemos conseguir un cambio profundo en el diseño de la página web. También cabe destacar el mínimo uso de código *HTML*, el cual es prácticamente inexistente. Todas las vistas web se generan automáticamente haciendo uso de las etiquetas *JSP*. El uso de código *HTML* dentro de los archivos *.jsp* obliga a modificaciones manuales si se desea cambiar la interfaz.

4.2.3. Necesidades hardware para el desarrollo del proyecto

Para poder desarrollar este proyecto ha sido necesario el uso de sistemas *PC* bastantes actuales y potentes. El mantenimiento de miles de líneas de código corriendo sobre un *IDE Java*, con un servidor de aplicaciones que funciona también sobre java hace que sea imposible desarrollar en equipos antiguos. Aunque no lo parezca, ha sido un gran hándicap a la hora de hacer un desarrollo continuo y progresivo del proyecto. Sí Eclipse tarda minutos en abrir ficheros o el servidor de aplicaciones necesita 5 minutos para arrancar y desplegar la aplicación, no se presta la atención necesaria a la hora de escribir código. Esto es sin embargo un problema del que adolece toda aplicación *Java*, al funcionar en una máquina virtual.

5. Resultado y conclusiones

Lo primero que deseo destacar como cierre a este proyecto de fin de carrera es la oportunidad que él mismo me ha brindado a la hora de aprender nuevas formas y herramientas de desarrollo software. La curva de aprendizaje de la plataforma de desarrollo *Java EE* es realmente considerable. De tal manera que bastante tiempo del requerido para la realización de este proyecto ha sido el estudio y asimilación de esta tecnología. Aspecto clave para realmente entender qué es y que hace *Hecaton*.

También es importante mencionar el hecho de haber partido de un proyecto ya existente, después de tres ciclos de desarrollo independientes entre sí. Ha sido una tarea ardua entender no sólo el código, si no la motivación y las razones que han llevado a los alumnos anteriores a tomar ciertas decisiones. Muchos de estos detalles solo los he llegado a comprender después de mucho tiempo.

Hecaton es una herramienta de telecontrol y monitorización muy potente, basada en el uso de software libre. Lo más importante de todo es que funciona. Su núcleo de monitorización y control de alarmas es ingenioso y está bien diseñado. El nivel de funcionalidad y personalización que se ofrece es altísimo, permitiendo que podamos usar esta aplicación con éxito en infinidad de escenarios.

Sin embargo era imprescindible una actualización de la aplicación. El estado del código fuente era caótico y se ha debido actuar sobre este problema, dejando de lado algunos objetivos iniciales. Me resulta complicado entender que desde el proyecto inicial no se haya planteado en ningún momento la revisión de las herramientas de desarrollo o la reorganización del código fuente.

La tarea de refactorización llevada a cabo ha sido intensa. Para ilustrar este gran trabajo quisiera destacar que el código fuente de la versión anterior consistía en aproximadamente 90.000 líneas de código entre ficheros *Java* y *JSP*. A día de hoy los fuentes constan de no mucho más de 40.000 líneas manteniendo toda la funcionalidad. Cabe destacar

la gran reducción de código que ha permitido la introducción de Struts2 así como el rediseño de la interfaz. A día de hoy la interfaz realiza tarea más sencillas y similares en toda la aplicación con lo que la reutilización de código es considerable. La nueva organización de código, separando el proyecto en 3 subproyectos que pueden al mismo tiempo ser compilados y ejecutados, hace que el mantenimiento de la aplicación sea mucho más sencillo. También quisiera hacer énfasis en la introducción de identificadores numéricos para todas las entidades del sistema. El hecho de trabajar con claves no numéricas basadas en nombres o compuestas, es un detalle que ha sido ya destacado como error de diseño y problemático en la memoria del proyecto de Andrés del Pino Bolaños. También se menciona la cantidad de trabajo que supondría la revisión de este problema. Sin embargo ha sido llevada a cabo con éxito.

En este proyecto de fin de carrera se ha conseguido llevar la aplicación a estándares de desarrollo actuales. Se han introducido los principios del desarrollo ágil. La introducción de *Maven* en el proyecto es una piedra angular de este desarrollo. Desde la gestión de dependencias hasta la introducción del ciclo de desarrollo, se ha mejorado la forma en que se realizan cambios en el código fuente. Quiero destacar que antes de la realización de este proyecto no conocía la existencia de *Maven*. La lectura de documentación me ha permitido conocer esta herramienta. Su sencillez me ha permitido dominarla en relativamente poco tiempo y comprender sus enormes posibilidades.

Quisiera hacer un inciso en la necesidad de introducir los test en la aplicación. Durante el desarrollo de la misma se han realizado test de funcionamiento, sin llegar a escribir test unitarios. Esta tarea llevaría mucho tiempo, la implementación de test para código existente es una tarea ardua que requiere una comprensión profunda del mismo, cosa que no he llegado a alcanzar hasta el final de este proyecto. Sí es destacable, el hecho de que ahora exista una plataforma para la ejecución de test automática, como nos ofrece *Maven*. En mi opinión sería interesante ir más allá e introducir la metodología de desarrollo continuo. Esto reduciría el tiempo de desarrollo sustancialmente al no introducirse nuevos errores con cada cambio en el código.

Por otra parte se ha desarrollado una nueva interfaz moderna y agradable. Se ha rediseñado los casos de uso haciendo que la utilización de la herramienta sea más sencilla. Me parece digno de especial mención, la introducción de elementos dinámicos en la interfaz web. Además el uso de tecnologías como *AJAX* y *JSON* permitirá que los futuros desarrollos sean

mucho más simples. Por ejemplo, el desarrollo de una aplicación móvil, que no ha podido ser abordar en este proyecto, resultaría bastante sencillo ya que podemos reutilizar muchas de las acciones que realiza la interfaz web.

La revisión del funcionamiento del módem *GSM* es una tarea que surge realmente tarde en el desarrollo del proyecto al no ser imprescindible para que la aplicación funcione. La introducción del nuevo controlador *Rxtx* era imprescindible para poder utilizar el dispositivo con sistemas de 64 bits tanto en *Windows* como en *Linux*. De la misma manera era necesaria la actualización de la librería *SmsLib*. Afortunadamente esta actualización no ha supuesto un quebradero de cabeza en gran parte debido a la flexibilidad de la metodología de desarrollo utilizada.

Quiero destacar que en todo momento he tenido presente la idea de que este software, nacido en la universidad y que utiliza tecnologías libres, va a continuar siendo desarrollado por futuros alumnos. Espero y deseo que los cambios introducidos permitan que nuevos ciclos de desarrollo resulten más sencillos. Y que el estudio y comprensión de la aplicación, así como de su código fuente, resulte menos costoso.

6. Trabajos Futuros

Existen infinidad de tareas y nuevas ideas que presentan un gran atractivo para hacer de esta aplicación un producto deseable para potenciales clientes. Quiero recoger en los siguientes párrafos un resumen de aquellas tareas que considero más interesantes así como ofrecer las líneas de desarrollo que éstas deberían seguir.

En primer lugar creo necesario llevar a cabo una prueba real del software. Esto es algo que ya estaba contemplado en este proyecto y que en cierta medida se intentó realizar con el montaje de la aplicación en una instalación de desalación de agua para uso académico. Esta planta se encontraba en la facultad de ingeniería de esta universidad y estaba siendo usada en para el proyecto de doctorado de la alumna Noemí Melián Martel. Diferentes retrasos y problemas con el hardware de la planta han llevado a desestimar el uso de la aplicación en la misma. Sin embargo los test llevados a cabo y la retroalimentación obtenida de ellos, han llevado a corregir algunos fallos fundamentales en el software así como plantear nuevas funcionalidades. Por todo esto, considero como primera prioridad para futuros desarrollos, que se plantee una prueba real de la aplicación mediante la colaboración con alguna empresa.

Tan importante es esta tarea que cambiar o añadir nueva funcionalidad al proyecto cuando no está claro cómo o dónde puede funcionar, no lleva a ninguna parte. Aunque creo necesario, por ejemplo, revisar la interfaz web cuando el soporte para *Bootstrap 3* en *Struts 2* esté disponible. Esta tarea, así como otras similares, deben estar supeditadas al test en un entorno real de la aplicación.

Paralelo al desarrollo de nuevas funcionales, es muy conveniente introducir los test en la aplicación. La escritura de test unitario, que se ejecuten automáticamente cada vez que se realicen cambios en el software, hace que se elimine la posibilidad de aparición de errores. El tiempo de desarrollo aumentará considerablemente utilizando estas técnicas. Si bien la inversión de tiempo inicial en el desarrollo de test requiere, de un esfuerzo considerable. Más allá de la introducción de simples test unitarios, propondría la utilización de un entorno de integración continua que garantiza en todo momento que la aplicación está libre de errores.

Creo conveniente la actualización del módem *GSM* utilizado, aún y cuando este cambio requiere cierto esfuerzo. La utilización de la interfaz serie supone un quebradero de cabeza en *Java*. Además el hardware está anticuado. La conexión a la red *GSM* del módem requiere de más de un minuto. Es necesario plantear una revisión del hardware, explorando nuevas posibilidades. En caso que no fuera posible realizar este cambio sería interesante explorar soluciones software para acelerar las tareas del módem *GSM*.

Resulta especialmente interesante para esta aplicación el uso de dispositivos de lectura y actuación inalámbricos. También parte de los planteamientos iniciales de este proyecto, la falta de tiempo y la prioridad de otras tareas, llevaron a aparcar esta idea. Sin embargo se realizó un cambio importante y necesario a nivel de diseño para permitir el uso de nuevos dispositivos de adquisición de datos. Esto ha sido, la creación del módulo de dispositivos de adquisición de datos. Ahora sería una tarea sencilla definir nuevas interfaces que permitirán utilizar nuevo hardware sin que esto suponga tener que cambiar todo el código o funcionamiento de la aplicación.

A nivel de código fuente y diseño creo realmente necesario plantear una revisión del sistema de alertas y de secuencias. El uso de ristras para la definición de reglas y pasos no tiene sentido. Precisamente la capa web trabaja a nivel de objetos para la obtención de los valores de los formularios, luego se crean las ristras de caracteres para ser almacenada en las tablas. Más tarde las ristras se convierten de nuevo en objetos que son los utilizados por el núcleo de alertas. Como parece obvio, sobra una conversión. El uso de ristras para mostrar las reglas y paso no es desafortunado, pero si el hecho de guardar esta información como ristras en el base de datos. Simples métodos que nos conviertan los objetos a su versión ristra serían suficientes a la hora de mostrarlas al usuario. Todo esto hace que el código sea tremendamente complicado, cuando las tareas son simples. Es posible simplificar el código de manera que este pase a ser comprensible y su mantenimiento no requiera de mucho tiempo.

Por último quisiera proponer una revisión del módulo de usuarios y de cómo el sistema realiza el seguimiento de las acciones que estos realizan. A día de hoy los actores y roles del sistema han ido cambiando y no llega a ser comprensible, cuáles son realmente. Además es necesario un control sobre las acciones que se realizan en el sistema. Ocurre que se ha añadido cierta información de control dentro de los objetos para controlar quién y cuándo los ha

modificado. En primer lugar decir que no hay un criterio claro a la hora de definir esta información. Algunas clases incluyen un objeto que aglutina esta información, otras la introducen en los atributos de la clase y otras finalmente no la tienen. Conviene estudiar que es necesario y definir las relaciones entre objetos que nos permitan trazar y recuperar la historia de los objetos y quien realizó los cambios. Por ejemplo es incomprensible que el nombre de usuario identifique quien creó un sensor, y no su identificador.

Anexo A. Manual de uso de la aplicación

a.1. Usuarios

a.1.1. Acceder al sistema

En la zona central aparecerá un formulario para introducir el nombre de usuario y la contraseña, introduzca su usuario y su contraseña y pulse LOGIN. Para poder acceder al sistema debe estar dado de alta en el sistema y solo aparecerá la opción de *login* si no estamos ya autentificados en el sistema.

a.1.2. Salir del sistema

En la cabecera de la aplicación se encuentre en la parte derecha se encuentra indicado el usuario actual, su rol y debajo el botón LOGOUT. Pulsando sobre el mismo precederemos a salir del sistema. Seremos redirigidos a la ventana de *login*. Solo podemos salir del sistema si nos encontramos autentificados él.

a.1.3. Acceder al módulo de usuarios

En el menú superior nos situamos encima de ADMINISTRACIÓN y pulsamos. Se desplegará un submenú donde pulsaremos sobre USUARIOS. Nos aparecerá un una tabla donde veremos un resumen de los usuarios registrados en el sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar los usuarios.

a.1.4. Consultar usuarios

Dentro del módulo de usuarios, seleccionamos de la tabla la entrada del usuario que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la

tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del usuario.

a.1.5. Añadir un usuario

Dentro del módulo de usuario, debajo de la tabla de usuarios pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren los datos necesarios para dar de alta un usuario. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el usuario ha sido añadido. Cerrando esta ventana volveremos a la vista de la gestión de usuarios donde podremos encontrar al usuario añadido.

a.1.6. Eliminar un usuario

Dentro del módulo de usuarios, seleccionamos de la tabla la entrada del usuario que queremos eliminar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del usuario. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.1.7. Modificar usuario

Dentro del módulo de usuarios, seleccionamos de la tabla la entrada del usuario que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del usuario. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista de la gestión de usuarios.

a.2. Módulos

a.2.1. Acceder a la vista de módulos

En el menú superior nos situamos encima de MÓDULOS y pulsamos. Nos aparecerá un una tabla donde veremos un resumen de los módulos de adquisición de datos que han sido añadidos al sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar los mismos.

a.2.2. Añadir hardware de adquisición de datos

Dentro de la vista de módulos, debajo de la tabla donde se listan los módulos disponibles, pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren los datos necesarios para dar de alta un módulo de adquisición de datos en el sistema. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el módulo ha sido añadido. Cerrando esta ventana volveremos a la vista de módulos donde podremos encontrar el último añadido.

a.2.3. Consultar módulos de adquisición de datos

Dentro de la vista de módulos, seleccionamos de la tabla la entrada del módulo que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del módulo.

a.2.4. Eliminar módulo de adquisición de datos

Dentro de la vista de módulos, seleccionamos de la tabla la entrada del módulo que queremos eliminar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del módulo de adquisición de datos. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.2.5. Modificar hardware de adquisición de datos

Dentro de la vista de módulos, seleccionamos de la tabla la entrada del módulo que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del módulo de adquisición de datos seleccionado. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista de módulos.

a.3. Sensores

a.3.1. Catálogo

a.3.1.1. Acceder al catálogo de sensores

En el menú superior nos situamos encima de SENSORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre CATÁLOGO. Nos aparecerá un una tabla donde veremos un resumen de los sensores que han sido añadidos al catálogo. Desde esta ventana podremos añadir, modificar, eliminar y consultar sensores.

a.3.1.2.Insertar sensor en el catálogo de sensores

Dentro del catálogo de sensores, debajo de la tabla de sensores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren los datos necesarios para dar de alta un sensor en el catálogo. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el sensor ha sido añadido. Cerrando esta ventana volveremos a la vista del catálogo de sensores donde podremos encontrar el sensor añadido.

a.3.1.3. Eliminar sensor del catálogo

Dentro del catálogo de sensores, seleccionamos de la tabla la entrada del sensor que queremos eliminar del catálogo pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los

datos en detalle del sensor. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.3.1.4. Modificar sensores del catálogo

Dentro del catálogo de sensores, seleccionamos de la tabla la entrada del sensor que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del sensor. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista del catálogo de sensores.

a.3.1.5. Consultar sensor del catálogo

Dentro del catálogo de sensores, seleccionamos de la tabla la entrada del sensor que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del sensor.

a.3.2. Instalación

a.3.2.1. Acceder a sensores instalados

En el menú superior nos situamos encima de SENSORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre INSTALADOS. Nos aparecerá un una tabla donde veremos un resumen de los sensores del catálogo que han sido instalados. Desde esta ventana podremos añadir, modificar, eliminar y consultar estos sensores.

a.3.2.2. Instalación de un sensor

Dentro de la vista de sensores instalados, debajo de la tabla de sensores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos una tabla con el catálogo de sensores y un formulario donde se requieren los datos necesarios para instalar un sensor.

Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el sensor ha sido instalado. Cerrando esta ventana volveremos a la vista de sensores instalados donde podremos encontrar el sensor añadido.

a.3.2.3. Consultar sensor instalado

Dentro de la vista de sensores instalados, seleccionamos de la tabla la entrada del sensor que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del sensor instalado.

a.3.2.4. Eliminar sensor instalado

Dentro de la vista de sensores instalados, seleccionamos de la tabla la entrada del sensor que queremos desinstalar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del sensor. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.3.2.5. Modificar sensores instalados

Dentro de la vista de sensores instalados, seleccionamos de la tabla la entrada del sensor que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del sensor. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista de sensores instalados.

a.3.3. Conexión

a.3.3.1. Acceder a sensores conectados

En el menú superior nos situamos encima de SENSORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre CONEXIÓN. Nos aparecerá un una tabla donde veremos un resumen de los sensores instalados que han sido conectados a un módulo de adquisición de datos. Desde esta ventana podremos añadir, modificar, eliminar y consultar estos sensores.

a.3.3.2. Conectar un sensor a un módulo HAD

Dentro de la vista de sensores instalados, debajo de la tabla de sensores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos una tabla con los sensores instalados disponibles. Seleccionamos el sensor que deseamos conectar y pulsamos el botón ENVIAR. En la misma ventana aparecerá otra tabla con la lista de módulos y sus entradas disponibles. Seleccionamos la entrada del módulo donde queremos instalar el sensor y pulsamos el botón ENVIAR. En la siguiente ventana aparecerá un resumen del sensor seleccionado y del módulo así como la entrada seleccionada además de un formulario con el resto de campos a rellenar. Una vez añadidos el resto de los datos pulsamos el botón ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el sensor ha sido conectado. Cerrando esta ventana volveremos a la vista de sensores instalados donde podremos encontrar el sensor conectado.

a.3.3.3 Consultar sensor conectado

Dentro de la vista de sensores conectados, seleccionamos de la tabla la entrada del sensor que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del sensor conectado así como el módulo correspondiente.

a.3.3.4. Eliminar conexión de un sensor

Dentro de la vista de sensores conectados, seleccionamos de la tabla la entrada del sensor que queremos desconectar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle de la conexión del sensor. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.3.3.5. Modificar conexión sensor - módulo

Dentro de la vista de sensores instalados, seleccionamos de la tabla la entrada del sensor que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana donde encontramos una tabla con los sensores instalados disponibles donde el sensor conectado aparece seleccionado. Seleccionamos el nuevo sensor que deseamos conectar (o podemos mantener el mismo) y pulsamos el botón ENVIAR. En la misma ventana aparecerá otra tabla con la lista de módulos y sus entradas disponibles donde se encuentra seleccionada la entrada ocupada actualmente. Seleccionamos la entrada del módulo donde queremos instalar el sensor y pulsamos el botón ENVIAR. En la siguiente ventana aparecerá un resumen del sensor seleccionado y del módulo así como la entrada seleccionada además de un formulario con el resto de campos. Una vez modificados el resto de los datos pulsamos el botón ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que la conexión ha sido modificada. Cerrando esta ventana volveremos a la vista de sensores conectados donde podremos encontrar el sensor modificado.

a.4. Actuadores

a.4.1. Catálogo

a.4.1.1. Acceder al catálogo de actuadores

En el menú superior nos situamos encima de ACTUADORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre CATÁLOGO. Nos aparecerá un una tabla donde veremos un resumen de los actuadores que han sido añadidos al catálogo. Desde esta ventana podremos añadir, modificar, eliminar y consultar actuadores.

a.4.1.2. Insertar actuador en el catálogo de actuadores

Dentro del catálogo de actuadores, debajo de la tabla de actuadores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren

los datos necesarios para dar de alta un actuador en el catálogo. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el actuador ha sido añadido. Cerrando esta ventana volveremos a la vista del catálogo de actuadores donde podremos encontrar el actuador añadido.

a.4.1.3. Eliminar actuador del catálogo

Dentro del catálogo de actuadores, seleccionamos de la tabla la entrada del actuador que queremos eliminar del catálogo pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del actuador. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.4.1.4. Modificar actuador del catálogo

Dentro del catálogo de actuadores, seleccionamos de la tabla la entrada del actuador que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del actuador. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista del catálogo de actuadores.

a.4.1.5. Consultar actuador del catálogo

Dentro del catálogo de actuadores, seleccionamos de la tabla la entrada del actuador que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del actuador.

a.4.2. Instalación

a.4.2.1. Acceder a actuadores instalados

En el menú superior nos situamos encima de ACTUADORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre INSTALADOS. Nos aparecerá un una tabla donde veremos un resumen de los actuadores del catálogo que han sido instalados. Desde esta ventana podremos añadir, modificar, eliminar y consultar estos actuadores.

a.4.2.2. Instalación de un actuador

Dentro de la vista de actuadores instalados, debajo de la tabla de actuadores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos una tabla con el catálogo de actuadores y un formulario donde se requieren los datos necesarios para instalar un actuador. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el actuador ha sido instalado. Cerrando esta ventana volveremos a la vista de actuadores instalados donde podremos encontrar el actuador añadido.

a.4.2.3. Consultar actuador instalado

Dentro de la vista de actuadores instalados, seleccionamos de la tabla la entrada del actuador que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del actuador instalado.

a.4.2.4. Eliminar actuador instalado

Dentro de la vista de actuadores instalados, seleccionamos de la tabla la entrada del actuador que queremos desinstalar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del actuador. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.4.2.5. Modificar actuadores instalados

Dentro de la vista de actuadores instalados, seleccionamos de la tabla la entrada del actuador que queremos modificar pulsando sobre él de manera que quede marcado. Ahora,

debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del actuador. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista de actuadores instalados.

a.4.3. Conexión

a.4.3.1. Acceder a actuadores conectados

En el menú superior nos situamos encima de ACTUADORES y pulsamos. Se desplegará un submenú donde pulsaremos sobre CONEXIÓN. Nos aparecerá un una tabla donde veremos un resumen de los actuadores instalados que han sido conectados a un módulo de adquisición de datos. Desde esta ventana podremos añadir, modificar, eliminar y consultar estos actuadores.

a.4.3.2. Conectar un actuador a un módulo HAD

Dentro de la vista de actuadores instalados, debajo de la tabla de actuadores pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos una tabla con los actuadores instalados disponibles. Seleccionamos el actuador que deseamos conectar y pulsamos el botón ENVIAR. En la misma ventana aparecerá otra tabla con la lista de módulos y sus salidas disponibles dependiendo de la activación del actuador (analógica o digital). Seleccionamos la salida del módulo donde queremos instalar el actuador y pulsamos el botón ENVIAR. En la siguiente ventana aparecerá un resumen del actuador seleccionado y del módulo así como la salida seleccionada además de un formulario con el resto de campos a rellenar. Una vez añadidos el resto de los datos pulsamos el botón ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el actuador ha sido conectado. Cerrando esta ventana volveremos a la vista de actuadores instalados donde podremos encontrar el actuador conectado.

a.4.3.3. Consultar actuador conectado

Dentro de la vista de actuadores conectados, seleccionamos de la tabla la entrada del actuador que queremos consultar pulsando sobre él de manera que quede marcado. Ahora,

debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del actuador conectado así como el módulo correspondiente.

a.4.3.4. Eliminar conexión de un actuador

Dentro de la vista de actuadores conectados, seleccionamos de la tabla la entrada del actuador que queremos desconectar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle de la conexión del actuador. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.4.3.5. Modificar conexión actuador - módulo

Dentro de la vista de actuadores instalados, seleccionamos de la tabla la entrada del actuador que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana donde encontramos una tabla con los actuadores instalados disponibles donde el actuador conectado aparece seleccionado. Seleccionamos el nuevo actuador que deseamos conectar (o podemos mantener el mismo) y pulsamos el botón ENVIAR. En la misma ventana aparecerá otra tabla con la lista de módulos y sus salidas disponibles dependiendo de la activación del actuador (analógica o digital), donde se encuentra seleccionada la salida ocupada actualmente. Seleccionamos la salida del módulo donde queremos instalar el actuador y pulsamos el botón ENVIAR. En la siguiente ventana aparecerá un resumen del actuador seleccionado y del módulo así como la salida seleccionada además de un formulario con el resto de campos. Una vez modificados el resto de los datos pulsamos el botón ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que la conexión ha sido modificada. Cerrando esta ventana volveremos a la vista de actuadores conectados donde podremos encontrar el actuador modificado.

a.5. Alarmas

a.5.1. Acceder a la gestión de alarmas

En el menú superior nos situamos encima de ALARMAS y pulsamos. Nos aparecerá un una tabla donde veremos un resumen de las alarmas añadidas al sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar estas alarmas.

a.5.2. Insertar una alarma en el sistema

Dentro de la vista de alarmas del sistema, debajo de la tabla de alarmas pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la alarma, su nombre y descripción. Introducimos esta información y pulsamos el botón ENVIAR. En la siguiente ventana veremos los datos introducidos y tres secciones donde podemos añadir las reglas a la alarma:

- **Reglas de activación** - En esta sección podemos ver una tabla resumen con las reglas de activación añadidas a la alarma. Para añadir, modificar o eliminar estas reglas usaremos los botones que se encuentran debajo de la tabla.

    - AÑADIR - Pulsando sobre el botón aparecerá una nueva ventana emergente donde encontraremos una tabla de sensores conectados así como las opciones para definir la regla de activación. Introducimos los datos y pulsamos ENVIAR. Si los datos son correctos se cerrará la ventana y veremos nuestra regla en la lista de reglas de activación de la alarma.

    - MODIFICAR - Si seleccionamos una regla de activación y pulsamos este botón aparecerá una nueva ventana igual a añadir donde los datos de la regla se encuentra ya escritos. Realizamos los cambios que consideremos oportunos y pulsamos el botón ENVIAR. La ventana se cierra y podremos ver los cambios en la lista.

    - ELIMINAR - Seleccionando una regla de activación y pulsando el botón se desplegará una ventana emergente con los detalles de la regla. Para confirmar pulsando en el botón ELIMINAR. Se cierra la ventana y la regla queda eliminada.

- **Reglas de alarma** - En esta sección podemos ver una tabla resumen con las reglas de alarma añadidas a la alarma. Para añadir, modificar o eliminar estas reglas de alarma usaremos los botones que se encuentran debajo de la tabla.

    - AÑADIR - Pulsando sobre el botón aparecerá una nueva ventana emergente donde encontraremos una tabla de alarmas disponibles en el sistema así como las opciones para definir la regla de alarma. Introducimos los datos y pulsamos ENVIAR. Si los datos son correctos se cerrará la ventana y veremos nuestra regla en la lista de reglas de alarma de la alarma en cuestión.

    - MODIFICAR - Si seleccionamos una regla de alarma y pulsamos este botón aparecerá una nueva ventana igual a añadir donde los datos de la regla se encuentra ya escritos. Realizamos los cambios que consideremos oportunos y pulsamos el botón ENVIAR. La ventana se cierra y podremos ver los cambios en la lista.
    - ELIMINAR - Seleccionando una regla de alarma y pulsando el botón se desplegará una ventana emergente con los detalles de la regla. Para confirmar pulsando en el botón ELIMINAR. Se cierra la ventana y la regla queda eliminada.

- **Reglas de comparación**- En esta sección podemos ver una tabla resumen con las reglas de comparación de sensores añadidas a la alarma. Para añadir, modificar o eliminar estas reglas usaremos los botones que se encuentran debajo de la tabla.

    - AÑADIR - Pulsando sobre el botón aparecerá una nueva ventana emergente donde encontraremos dos tablas de sensores conectados así como las opciones para definir la regla de activación. Seleccionamos el sensor de la primera tabla que deseamos comparar con un sensor de la segunda tabla. Introducimos el resto de los datos y pulsamos ENVIAR. Si los datos son correctos se cerrará la ventana y veremos nuestra regla en la lista de reglas de activación de la comparación.
    - MODIFICAR - Si seleccionamos una regla de comparación y pulsamos este botón aparecerá una nueva ventana igual a añadir donde los datos de la regla se encuentra ya escritos. Realizamos los cambios que consideremos oportunos y

pulsamos el botón ENVIAR. La ventana se cierra y podremos ver los cambios en la lista.

- ELIMINAR - Seleccionando una regla de comparación y pulsando el botón se desplegará una ventana emergente con los detalles de la regla. Para confirmar pulsando en el botón ELIMINAR. Se cierra la ventana y la regla queda eliminada.

Cuando estemos conformes con las reglas añadidas pulsaremos en el botón ENVIAR en la parte baja de la ventana. Se el siguiente paso donde se añaden las acciones a realizar cuando la alarma es activada:

- **Reglas de actuación**- En esta sección podemos ver una tabla resumen con las reglas de actuación añadidas a la alarma. Para añadir, modificar o eliminar estas reglas usaremos los botones que se encuentran debajo de la tabla.

  - AÑADIR - Pulsando sobre el botón aparecerá una nueva ventana emergente donde encontraremos una tabla de actuadores conectados así como un campo para indicar el valor de salida del actuador. Introducimos los datos y pulsamos ENVIAR. Si los datos son correctos se cerrará la ventana y veremos nuestra regla en la lista de reglas de actuación de la alarma.

  - MODIFICAR - Si seleccionamos una regla de activación y pulsamos este botón aparecerá una nueva ventana igual a añadir donde los datos de la regla se encuentra ya escritos. Realizamos los cambios que consideremos oportunos y pulsamos el botón ENVIAR. La ventana se cierra y podremos ver los cambios en la lista.
  - ELIMINAR - Seleccionando una regla de activación y pulsando el botón se desplegará una ventana emergente con los detalles de la regla. Para confirmar pulsando en el botón ELIMINAR. Se cierra la ventana y la regla queda eliminada.

- **Reglas de secuencias** - En esta sección podemos ver una tabla resumen con las reglas de secuencias añadidas a la alarma. Para añadir, modificar o eliminar estas reglas de alarma usaremos los botones que se encuentran debajo de la tabla.

- AÑADIR - Pulsando sobre el botón aparecerá una nueva ventana emergente donde encontraremos una tabla de secuencias disponibles en el sistema así como las opciones para definir la regla de secuencias. Introducimos los datos y pulsamos ENVIAR. Si los datos son correctos se cerrará la ventana y veremos nuestra regla en la lista de reglas de alarma de la alarma en cuestión.

- MODIFICAR - Si seleccionamos una regla de secuencia y pulsamos este botón aparecerá una nueva ventana igual a añadir donde los datos de la regla se encuentra ya escritos. Realizamos los cambios que consideremos oportunos y pulsamos el botón ENVIAR. La ventana se cierra y podremos ver los cambios en la lista.

- ELIMINAR - Seleccionando una regla de secuencia y pulsando el botón se desplegará una ventana emergente con los detalles de la regla. Para confirmar pulsando en el botón ELIMINAR. Se cierra la ventana y la regla queda eliminada.

Una vez añadidas las acciones pulsamos sobre el botón ENVIAR. Nos llevara a la última ventana donde veremos un resumen de la alarma y las reglas añadidas. Debajo veremos los últimos campos de opciones. Después de comprobar y añadir los datos, pulsamos el botón ENVIAR. Nos aparecerá un mensaje confirmando que la alarma ha sido añadida.

a.5.3. Consultar alarma

Dentro de la vista de alarmas, seleccionamos de la tabla la entrada de la alarma que queremos consultar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle de la alarma correspondiente.

a.5.4. Eliminar alarma

Dentro de la vista de alarmas, seleccionamos de la tabla la entrada de la alarma que queremos eliminar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la

tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle de la alarma y sus reglas. Para eliminarla debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.5.5. Modificar Alarma

Dentro de la vista de alarmas, seleccionamos de la tabla la entrada de la alarma que queremos modificar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la alarma, su nombre y descripción. En este caso estos datos están ya presentes. A partir de aquí el procedimiento es igual a añadir alarma. Siga las instrucciones de la sección a.5.2.

a.6. secciones

a.6.1. Acceder a la gestión de secciones

En el menú superior nos situamos encima de ADMINISTRACIÓN y pulsamos. Se desplegará un submenú donde pulsaremos sobre SECCIONES. Nos aparecerá un una tabla donde veremos un resumen de las secciones añadidas al sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar estas secciones.

a.6.2. Insertar una sección en el sistema

Dentro de la vista de secciones del sistema, debajo de la tabla de secciones pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la sección. Introducimos esta información y pulsamos el botón ENVIAR. En la siguiente ventana veremos los datos introducidos y diferentes apartados donde podremos añadir elementos a la sección:

- **Añadir/quitar sensores** - En este espacio podemos añadir los sensores presentes en el sistema a la sección. Pulsamos en el botón AÑADIR/QUITAR y se abrirá una nueva ventana emergente. En ella veremos una nueva tabla donde se muestran los sensores. Podemos elegir uno o varios pulsando sobre las entradas de la lista. Para quitar simplemente debemos pulsar sobre un sensor ya seleccionado y este quedará no seleccionado. Cuando hayamos terminado pulsamos en el botón ENVIAR. Se cerrará la venta y podremos ver los sensores seleccionados en la lista.
- **Añadir/quitar actuadores** - En este espacio podemos añadir los actuadores presentes en el sistema a la sección. Pulsamos en el botón AÑADIR/QUITAR y se abrirá una nueva ventana emergente. En ella veremos una nueva tabla donde se muestran los actuadores. Podemos elegir uno o varios pulsando sobre las entradas de la lista. Para quitar simplemente debemos pulsar sobre un sensor ya seleccionado y este quedará no seleccionado. Cuando hayamos terminado pulsamos en el botón ENVIAR. Se cerrará la venta y podremos ver los actuadores seleccionados en la lista.

- **Añadir/quitar alarmas** - En este espacio podemos añadir las alarmas presentes en el sistema a la sección. Pulsamos en el botón AÑADIR/QUITAR y se abrirá una nueva ventana emergente. En ella veremos una nueva tabla donde se muestran las alarmas. Podemos elegir una o varias pulsando sobre las entradas de la lista. Para quitar simplemente debemos pulsar sobre una alarma ya seleccionada y esta quedará no seleccionada. Cuando hayamos terminado pulsamos en el botón ENVIAR. Se cerrará la venta y podremos ver las alarmas seleccionadas en la lista.

- **Añadir/quitar secuencias** - En este espacio podemos añadir las secuencias presentes en el sistema a la sección. Pulsamos en el botón AÑADIR/QUITAR y se abrirá una nueva ventana emergente. En ella veremos una nueva tabla donde se muestran las secuencias. Podemos elegir una o varias pulsando sobre las entradas de la lista. Para quitar simplemente debemos pulsar sobre una secuencia ya seleccionada y esta quedará no seleccionada. Cuando hayamos terminado pulsamos en el botón ENVIAR. Se cerrará la venta y podremos ver las secuencias seleccionadas en la lista.

Una vez estemos conformes con los elementos añadidos a la sección pulsamos sobre el botón ENVIAR. Nos aparecerá un mensaje confirmando que la sección ha sido añadida.

a.6.3. Consultar sección

Dentro de la vista de secciones, seleccionamos de la tabla la entrada de la sección que queremos consultar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle de la sección correspondiente y sus elementos.

a.6.4. Eliminar sección

Dentro de la vista de secciones, seleccionamos de la tabla la entrada de la sección que queremos eliminar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle de la sección. Para eliminarla debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.6.5. Modificar sección

Dentro de la vista de secciones, seleccionamos de la tabla la entrada de la sección que queremos modificar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la sección. En este caso estos datos están ya presentes. A partir de aquí el procedimiento es igual a añadir una sección. Siga las instrucciones de la sección a.6.2.

a.7. Secuencias de acciones

a.7.1. Acceder a la gestión de secuencias

En el menú superior nos situamos encima de SECUENCIAS y pulsamos. Nos aparecerá un una tabla donde veremos un resumen de las secuencias añadidas al sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar estas secuencias.

a.7.2. Insertar una secuencia en el sistema

Dentro de la vista de secuencias del sistema, debajo de la tabla de alarmas pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la secuencia. Introducimos esta información y pulsamos el botón ENVIAR. En la siguiente ventana veremos los datos introducidos seguidos de múltiples botones que permiten añadir pasos. Debajo aparece una tabla que mostrará los pasos añadidos a la secuencia. Desde ella podremos modificar o eliminar pasos. A continuación se describe qué pasos podemos añadir a la secuencia::

- **Añadir paso de espera temporal** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos definir a un tiempo de espera dependiendo de que se cumplan o no una serie de reglas. Estas reglas son similares a las usadas para las alarmas. Remitimos a la sección XXX para obtener información sobre las mismas. Seleccionamos el orden que ocupará este paso en la secuencia, su etiqueta si es necesario y el tiempo que debe pasar hasta que comience la acción del paso. Añadimos el resto de opciones y pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

- **Añadir paso de espera condicional** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos añadir un tiempo de espera que dependa del valor de un sensor que seleccionaremos de una lista. Seleccionamos el orden que ocupará este paso en la secuencia, su etiqueta si es necesario y el tiempo que debe pasar hasta que comience la acción del paso. Añadimos el resto de opciones y pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

- **Añadir paso de actuación** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos indicar un valor a un actuador conectado al sistema que seleccionaremos de una tabla. Seleccionamos el orden que ocupará este paso en la secuencia, su etiqueta si es

necesario y el tiempo que debe pasar hasta que comience la acción del paso. Añadimos el resto de opciones y pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

- **Añadir paso de salto** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos definir a qué otro paso saltará la secuencia de acciones. Podremos seleccionar el salto de los disponibles en una tabla. Seleccionamos el orden que ocupará este paso en la secuencia y añadimos el resto de opciones. Pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

- **Añadir paso de salto condicional** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos definir a qué otro paso saltará la secuencia de acciones dependiendo de que se cumplan o no una serie de reglas. Estas reglas son similares a las usadas para las alarmas. Remitimos a la sección XXX para obtener información sobre las mismas. Las dos últimas tablas mostrarán los pasos disponibles en la secuencia. En la primera tabla podemos seleccionar a que paso saltara la secuencia si se cumplen todas las reglas definidas. En el segundo seleccionamos a que paso ir si alguna de las reglas no se cumple. Seleccionamos el orden que ocupará este paso en la secuencia y añadimos el resto de opciones. Pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

- **Añadir paso de arranque/detención de secuencia** - Pulsando sobre el correspondiente botón se abrirá una nueva ventana emergente. En esta se mostrará un formulario donde podemos definir seleccionar una secuencia para arrancar o detener. Debemos tener en cuenta que si seleccionamos arrancar secuencias y esta se encuentra activa en el momento que la secuencia llega a este paso no se realizará ninguna acción. Lo mismo ocurrirá con si seleccionamos detener y la secuencia se encuentra detenida en el momento que se ejecuta este paso. Seleccionamos el orden que ocupará este paso en la secuencia, su etiqueta si es necesario y el tiempo que debe pasar hasta que comience la acción del paso. Añadimos el resto de opciones y pulsamos AÑADIR. Veremos el paso reflejado en la tabla.

Una vez estemos conformes con los pasos añadidos a la secuencia pulsamos sobre el botón ENVIAR. Nos aparecerá un mensaje confirmando que la sección ha sido añadida.

a.7.3. Consultar secuencia

Dentro de la vista de secuencias, seleccionamos de la tabla la entrada de la secuencia que queremos consultar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle de la secuencia correspondiente.

a.7.4. Eliminar secuencia

Dentro de la vista de secuencias, seleccionamos de la tabla la entrada de la secuencia que queremos eliminar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle de la secuencia y sus pasos. Para eliminarla debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.7.5. Modificar secuencia

Dentro de la vista de secuencias, seleccionamos de la tabla la entrada de la secuencia que queremos modificar pulsando sobre ella de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana donde encontramos un formulario con los datos principales de la secuencia. En este caso estos datos están ya presentes. A partir de aquí el procedimiento es igual a añadir una secuencia. Siga las instrucciones de la sección a.7.2.

a.7.6. Arrancar/detener secuencia de acciones

Pulsando sobre el último enlace en el menú lateral accederemos a la vista para añadir o detener secuencias de acciones. Se nos mostrarán dos tablas:

- En la primera veremos las secuencias disponibles en el sistema y se encuentran detenidas. Si seleccionamos una de ellas y pulsamos sobre el botón

ARRANCAR SECUENCIA que aparece debajo, la secuencia comenzará su ejecución. Esta secuencia será removida de tabla de secuencias detenidas y pasará a la segunda tabla.

- En la segunda veremos las secuencias que se están actualmente en ejecución. Si seleccionamos una de ellas y pulsamos sobre el botón DETENER SECUENCIA que aparece debajo, la secuencia será detenida por el sistema y pasará a la primera tabla.

a.8. Sms

a.8.1. Teléfonos

a.8.1.1. Acceder al catálogo de teléfonos

En el menú superior nos situamos encima de SMS y pulsamos. Se desplegará un submenú donde pulsaremos sobre TELÉFONOS. Nos aparecerá un una tabla donde veremos un resumen de los sensores que han sido añadidos al catálogo. Desde esta ventana podremos añadir, modificar, eliminar y consultar los teléfonos.

a.8.1.2. Insertar un teléfono en el catálogo

Dentro del catálogo de teléfonos, debajo de la tabla de teléfonos pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren los datos necesarios para dar de alta un teléfono en el catálogo. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el teléfono ha sido añadido. Cerrando esta ventana volveremos a la vista del catálogo de teléfonos donde podremos encontrar el teléfono añadido.

a.8.1.3. Eliminar teléfono del catálogo

Dentro del catálogo de teléfonos, seleccionamos de la tabla la entrada del teléfono que queremos eliminar del catálogo pulsando sobre él de manera que quede marcado. Ahora,

debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del teléfono. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.8.1.4. Modificar un teléfono del catálogo

Dentro del catálogo de teléfonos, seleccionamos de la tabla la entrada del teléfono que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del teléfono. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista del catálogo de teléfonos.

a.8.1.5. Consultar un teléfono del catálogo

Dentro del catálogo de teléfonos, seleccionamos de la tabla la entrada del teléfono que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del teléfono.

a.8.2. Mensajes

a.8.2.1. Acceder al catálogo de mensajes

En el menú superior nos situamos encima de SMS y pulsamos. Se desplegará un submenú donde pulsaremos sobre MENSAJES. Nos aparecerá un una tabla donde veremos un resumen de los mensajes que han sido añadidos al catálogo. Desde esta ventana podremos añadir, modificar, eliminar y consultar los mensajes.

a.8.2.2. Insertar un mensaje en el catálogo

Dentro del catálogo de mensajes, debajo de la tabla de mensajes pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos un formulario donde se requieren los datos necesarios para dar de alta un mensaje en el catálogo. En primer veremos una tabla donde podemos elegir los teléfonos a los que este mensaje será enviado. Introducimos los valores deseados y pulsamos en ENVIAR. Veremos un mensaje en la misma ventana donde se nos confirma que el mensaje ha sido añadido. Cerrando esta ventana volveremos a la vista del catálogo de mensajes donde podremos encontrar el mensaje añadido.

a.8.2.3. Eliminar mensaje del catálogo

Dentro del catálogo de mensajes, seleccionamos de la tabla la entrada del mensaje que queremos eliminar del catálogo pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle del mensaje. Para eliminarlo debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.8.2.4. Modificar un mensaje del catálogo

Dentro del catálogo de mensajes, seleccionamos de la tabla la entrada del mensaje que queremos modificar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el botón MODIFICAR. Aparecerá una ventana que muestra un formulario con los datos del mensaje y los teléfonos asociados al mismo. Realice las modificaciones necesarias y pulse el botón ENVIAR. Aparecerá un mensaje de confirmación. Cerrando la ventana de confirmación volverá a la vista del catálogo de mensajes.

a.8.2.5. Consultar un mensaje del catálogo

Dentro del catálogo de mensajes, seleccionamos de la tabla la entrada del mensaje que queremos consultar pulsando sobre él de manera que quede marcado. Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle del mensaje.

a.8.3. Asociaciones

a.8.3.1. Acceder a la vista de asociaciones sms-alarma

En el menú superior nos situamos encima de SMS y pulsamos. Se desplegará un submenú donde pulsaremos sobre ASOCIACIONES. Nos aparecerá un una tabla donde veremos un resumen de las asociaciones entre mensajes y alarmas del sistema. Desde esta ventana podremos añadir, modificar, eliminar y consultar qué mensajes se enviarán cuando ocurra una alarma.

a.8.3.2. Insertar una asociación

Dentro de la vista de asociaciones sms-alarma, debajo de la tabla de asociaciones pulsamos sobre el botón AÑADIR. Aparecerá una ventana donde encontramos dos tablas. En la primera se muestran las alarmas del sistema que no tienen ya un mensaje asociado. En la segunda la lista de mensajes disponibles. Debemos seleccionar un elemento de cada tabla para realizar la asociación sms-alarma. Una vez hecho esto pulsamos el botón AÑADIR. Veremos un mensaje en la misma ventana donde se nos confirma que el mensaje y la alarma han sido correctamente asociados. Cerrando esta ventana volveremos a la vista de asociaciones donde podremos encontrar la relación añadida.

a.8.3.3. Eliminar mensaje del catálogo

Dentro de la vista de asociaciones sms-alarma, seleccionamos de la tabla la entrada de la asociación que queremos eliminar del catálogo pulsando sobre esta de manera que quede marcada. Ahora, debajo de la tabla pulsamos en el botón ELIMINAR. Aparecerá una ventana que muestra los datos en detalle tanto de la alarma como del mensaje. Para eliminar la asociación debemos confirmar pulsando sobre el botón ELIMINAR que encontraremos al final de la ventana. Una vez pulsado se nos mostrará un mensaje de confirmación.

a.8.3.4. Consultar una asociación

Dentro de la vista de asociaciones sms-alarma, seleccionamos de la tabla la entrada de la asociación que queremos consultar pulsando sobre ella de manera que quede marcada.

Ahora, debajo de la tabla pulsamos en el primer botón VER. Aparecerá una ventana que muestra los datos en detalle de la asociación.

a.9. Gestión vía sms

a.9.1. Consultar listado sensores vía sms

Para recibir información vía SMS sobre el estado de los sensores dados de alta en el sistema, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna o Standard, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Listar sensores.

2. La seguridad en el envío de SMS esté fijada a Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña listar sensores.

   La información recibida será, Identificador Descripción Valor.

a.9.2. Consultar listado actuadores vía sms

Para recibir información vía SMS sobre el estado de los actuadores dados de alta en el sistema, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna o Standard, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Listar actuadores.

2. La seguridad en el envío de SMS esté fijada a Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña listar actuadores.

   La información recibida será, Identificador Descripción Valor Rango_inicial Rango_final.

a.9.3. Consultar listado alarmas vía sms

Para recibir información vía SMS sobre el estado de las alarmas dadas de alta en el sistema, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna o Standard, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Listar alarmas.

2. La seguridad en el envío de SMS esté fijada a Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña listar alarmas.

La información recibida será un listado de los nombres de las alarmas que estén activas en estos momentos.

a.9.4. Consultar listado secuencias de acciones vía sms

Para recibir información vía SMS sobre el estado de las secuencias de acciones dadas de alta en el sistema, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna o Standard, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Listar secuencias.

2. La seguridad en el envío de SMS esté fijada a Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña listar secuencias.

La información recibida será un listado de los nombres de las secuencias que estén activas e inactivas en estos momentos.

a.9.5. Modificar valor actuador vía sms:

Para modificar el valor de un actuador vía SMS, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Modificar actuador identificador valor.

2. La seguridad en el envío de SMS esté fijada a Standard o Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña modificar actuador identificador valor.

Una vez procesado el mensaje por el sistema, se recibirá una confirmación de la actuación.

a.9.6. Arrancar secuencia de acciones vía sms

Para arrancar una secuencia de acciones vía SMS, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Arrancar secuencia nombre_secuencia.

2. La seguridad en el envío de SMS esté fijada a Standard o Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña arrancar secuencia nombre_secuencia.

Una vez procesado el mensaje por el sistema, se recibirá una confirmación de la puesta en marcha de la secuencia.

a.9.7. Detener secuencia de acciones vía sms

Para detener una secuencia de acciones vía SMS, deberemos distinguir entre dos situaciones:

1. La seguridad en el envío de SMS está fijada a Ninguna, en cuyo caso simplemente tendremos que enviar un SMS cuyo contenido sea, Detener secuencia nombre_secuencia.

2. La seguridad en el envío de SMS esté fijada a Standard o Alta, en cuyo caso, tendremos que enviar un SMS cuyo contenido sea, Usuario Contraseña detener secuencia nombre_secuencia.

Una vez procesado el mensaje por el sistema, se recibirá una confirmación de la puesta en marcha de la secuencia.

a.10. Planta

a.10.1. Arrancar monitorización

En la barra lateral pulsaremos ARRANCAR MONITORIZACIÓN. Una vez arrancados todos los sistemas de monitorización se muestra la vista de monitorización de la instalación. En caso de que la monitorización ya estuviera activa, se mostrará la vista directamente sin realizar ningún otro cambio.

a.10.2. Detener monitorización

En la barra lateral pulsaremos DETENER MONITORIZACIÓN. Una vez detenidos todos los sistemas de monitorización se muestra la vista por defecto que indica que sistema está detenido. En caso de que la monitorización ya estuviera inactiva, se mostrará la vista por defecto directamente sin realizar ningún otro cambio

a.10.3. Visualizar estado de la planta

En menú superior pulsaremos sobre PÁGINA ESTADO. La ventana se nuestro navegador nos mostrará la vista de monitorización si la monitorización de la planta está activa. En caso de que esté parada, se muestra la vista por defecto donde se indica que la monitorización esta desactivada.

a.10.4. Consulta datos de la planta:

En el menú superior nos situamos encima de ADMINISTRACIÓN y pulsamos. Aparecerá un menú desplegable. Pulsamos sobre PLANTA y nos aparecerán los datos.

a.10.5. Modificación datos de la planta

Dentro de la consulta de datos de la planta tenemos un botón llamado MODIFICAR. Pulsando sobre él y se nos abrirá una ventana emergente con un formulario donde podemos modificar los valores de configuración de la planta. Una vez realizados los cambios solo tenemos que pulsar en el botón ENVIAR. Se nos mostrará un mensaje confirmándonos la correcta modificación de los datos.

a.11. Históricos

a.11. Visualizar históricos

En el menú superior nos situamos encima de ADMINISTRACIÓN y pulsamos. Aparecerá un menú desplegable. Pulsamos sobre HISTÓRICOS y nos aparecerá la ventana de consulta de registros. En ella debemos seleccionar una fecha de inicio y otra de fin para la consulta en el sistema de bases de datos. Podremos elegir entre mostrar los datos en modo de tabla o mediante una gráfica. Seleccionamos uno de los elementos de las tablas y finalmente pulsamos el botón CONSULTAR. Se abrirá una ventana donde se mostrarán los datos en el formato deseado.

a.12. Visualización de la planta

a.12.1. Iniciar aplicación de visualización de la planta

En el menú superior nos situamos encima de ADMINISTRACIÓN y pulsamos. Se abre un menú desplegable donde debemos pulsar sobre VISUALIZACIÓN. El navegador nos llevará al *applet Java* de visualización de la planta. Por favor comprueba todas las excepciones de seguridad que puedan aparecer en tu navegador y que no permitan mostrar el *applet*.

Las siguientes funcionalidades se llevan a cabo desde la aplicación de visualización de la imagen de la planta.

a.12.2. Adjuntar plano

En la pestaña Editor pulsamos Adjuntar Plano. Nos saldrá un menú para seleccionar la imagen, seleccionamos la imagen en nuestro sistema y pulsamos Abrir.

a.12.3. Añadir visualización de sensor

En la pestaña Editor seleccionamos el identificado de la lista a la derecha del botón sensor. Pulsamos el botón Sensor y picamos en el lugar de la zona de edición donde queremos que aparezca el cajetín de visualización.

a.12.4. Añadir visualización de actuador

En la pestaña Editor seleccionamos el identificado de la lista a la derecha del botón sensor. Pulsamos el botón Actuador y picamos en el lugar de la zona de edición donde queremos que aparezca el cajetín de visualización.

a.12.5. Mover cajetines

En la pestaña Editor pulsamos el botón Mover, a continuación mantenemos pulsado sobre el cajetín que queremos desplazar, cuando este en el lugar deseado dejamos de pulsar el botón del ratón.

a.12.6. Borrar cajetines

En la pestaña Editor pulsamos el botón Borrar, a continuación pulsamos sobre el cajetín que queremos borrar.

a.12.7. Limpiar imagen

En la pestaña Editor si deseamos borrar todos los cajetines pulsaremos el botón Limpiar.

a.12.8. Guardar diseño de planta

En la pestaña Editor para enviar los cambios en la imagen al servidor pulsamos el botón Guardar. Solo los cambios que han sido enviados al servidor podrán ser visibles para otros usuarios.

a.12.9. Cargar imagen de la planta

Cuando iniciamos la aplicación de visualización de la planta, esta auto-detecta si en el servidor ya existe una versión de la imagen y carga la versión actual. Una vez iniciada la aplicación si deseas refrescar la imagen con el servidor, haremos lo siguiente, en la pestaña Visualizador pulsaremos el botón Cargar.

a.12.10. Cambiar sección editor/visualizador de la planta

Tanto en la pestaña de Editor como en la de Visualizador, podremos encontrar a la derecha del botón Sección una lista con las secciones dadas de alta actualmente en el sistema.

Para cambiar entre una y otra, simplemente tendremos que seleccionar una sección del listado y pulsar sobre el botón Sección.

a.12.11. Arrancar Secuencia de acciones

En la pestaña Visualizador, podremos encontrar a la derecha del botón Arrancar un listado de las secuencias de acciones que en estos momentos se encuentran paradas. Para poner en marcha una de estas secuencias, sólo es necesario seleccionar alguna de ellas en el listado y pulsar sobre el botón Arrancar.

a.12.12. Detener secuencia de acciones

En la pestaña Visualizador, podremos encontrar a la derecha del botón Detener un listado de las secuencias de acciones que en estos momentos se encuentran en marcha. Para detener una de estas secuencias, sólo es necesario seleccionar alguna de ellas en el listado y pulsar sobre el botón Detener.

a.12.13. Modificar valor actuador

En la pestaña Visualizador, podremos ver la imagen que se ha cargado de la planta junto con los sensores y actuadores que se han colocado sobre la misma. Para modificar un actuador, basta con pulsar sobre su cajetín, al hacer esto, se abrirá una nueva ventana donde podremos ver una mayor información del actuador y asignarle un nuevo valor.

a.12. Ejemplo de creación de una planta completa

A continuación explicaremos la metodología a seguir para crear un sistema de control básico partiendo de cero. Nos apoyaremos en las descripciones de uso de la sección anterior. Enumeraremos paso a paso qué debemos hacer referenciando a la sección anterior para entrar en mayor detalle sobre cómo hacer cada tarea.

Paso 1. Introducir datos de la planta

Para este paso usaremos la funcionalidad descrita en la sección **Modificación datos de la planta.**

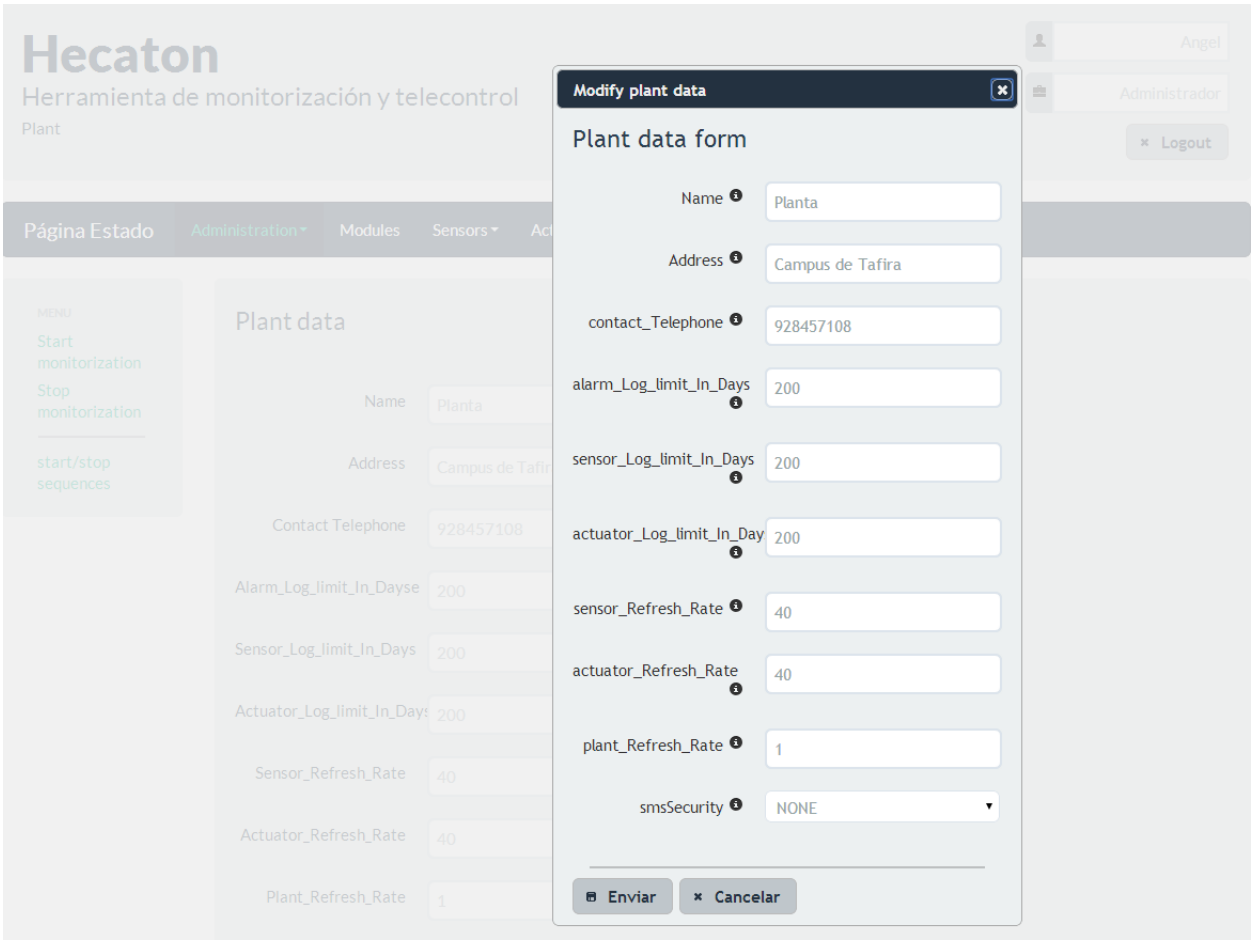

Ilustración 54: Inserción datos de la planta.

Paso 2. Introducir sensor en el catálogo

Para este paso usaremos la funcionalidad descrita en la sección Insertar **Sensor en el catálogo de sensores**.

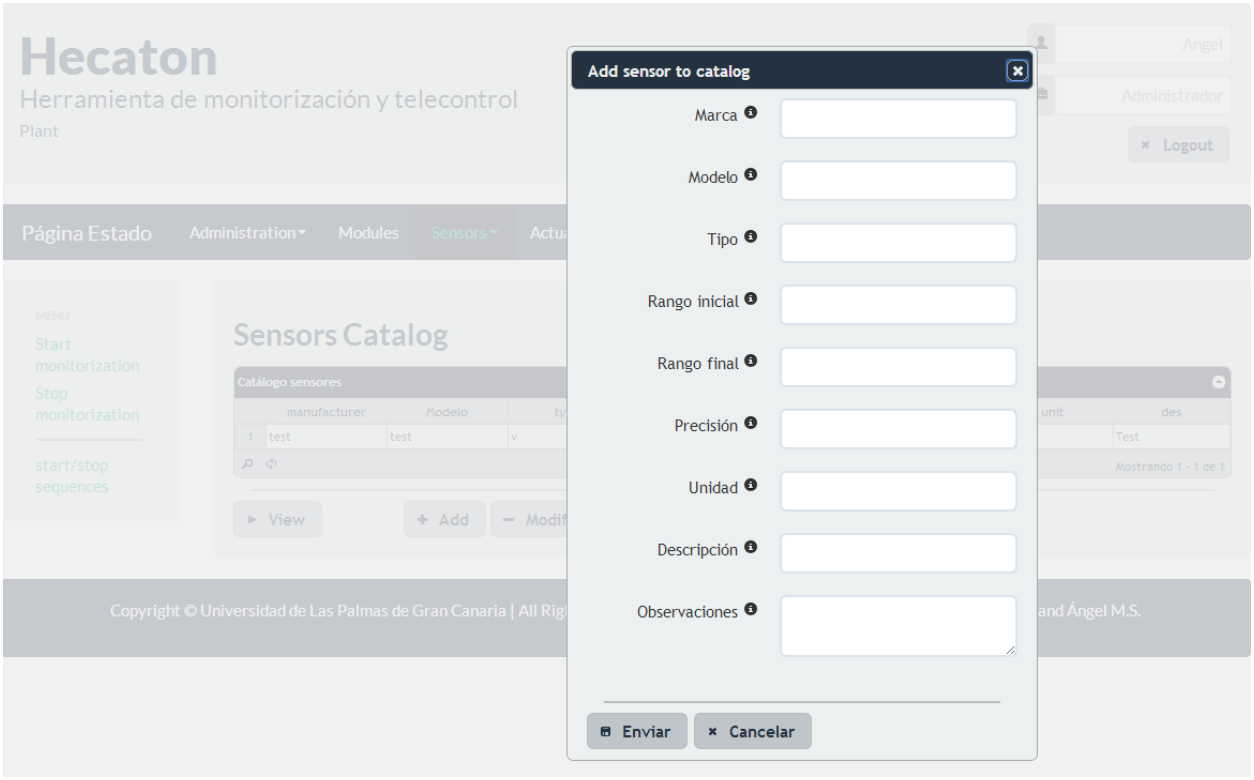

Ilustración 55: Inserción de un sensor en el catálogo.

Paso 3. Instalar un sensor

Para este paso usaremos la funcionalidad descrita en la sección **Instalación de un sensor**.

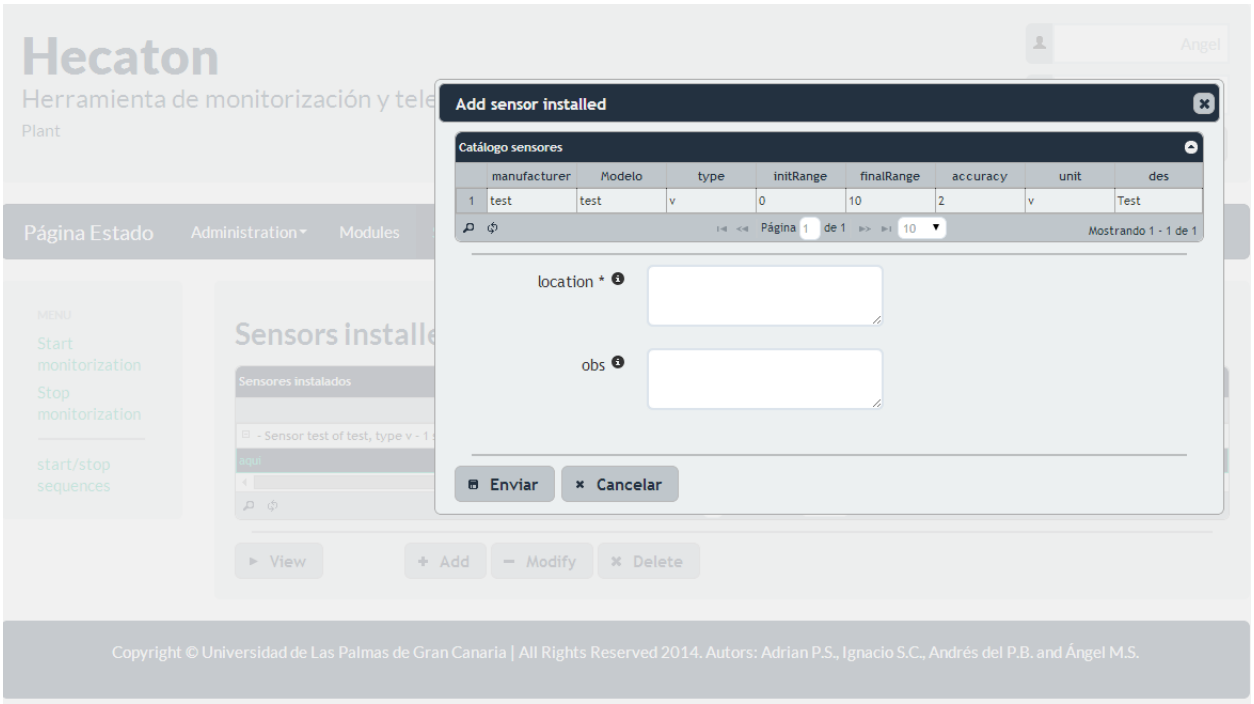

Ilustración 56: Instalación de un sensor.

Paso 4. Insertar módulo de adquisición de datos

Para este paso usaremos la funcionalidad descrita en la sección **Instalar hardware de adquisición de datos**.

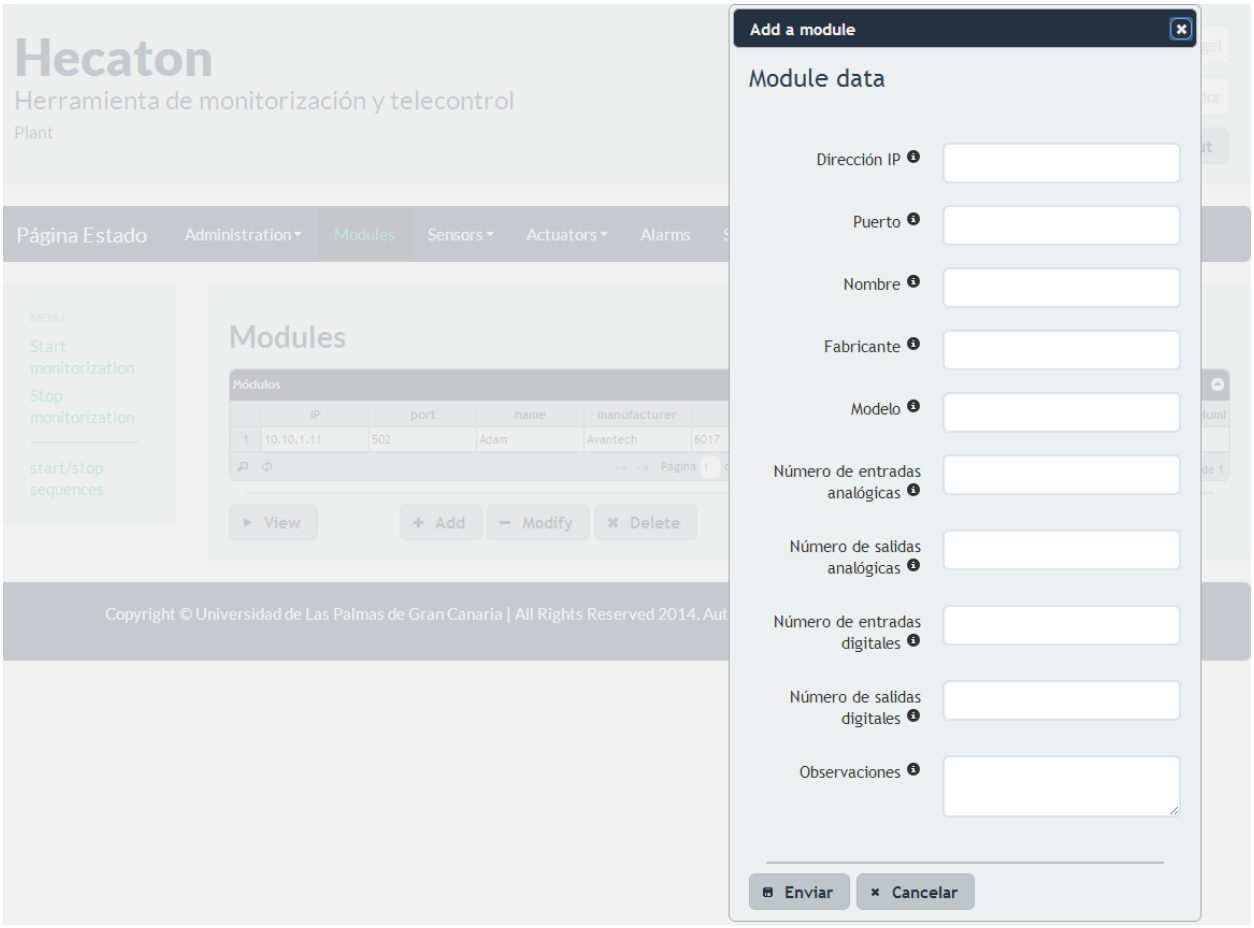

Ilustración 57: Instalación de un modo *HAD.*

Paso 5. Conectar sensor a módulo

Para este paso usaremos la funcionalidad descrita en la sección **Conectar un sensor a un módulo *HAD***.

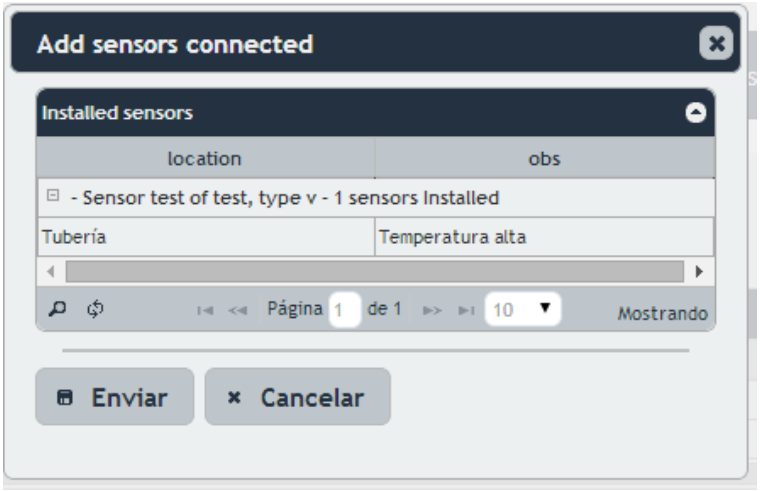

Ilustración 53: Conexión de un sensor a un módulo *HAD* 1.

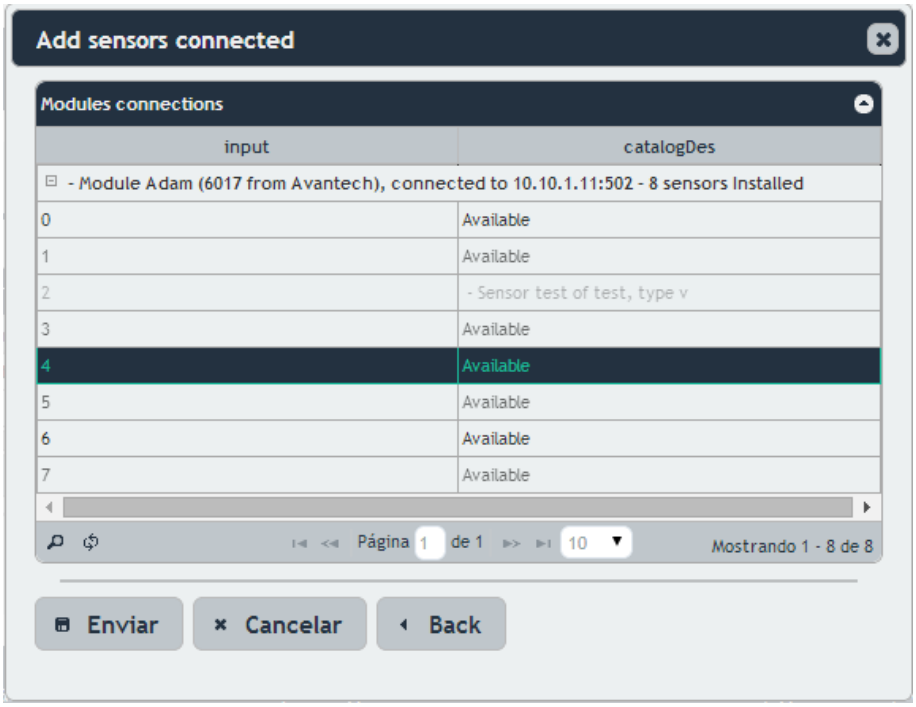

Ilustración 58: Conexión de un sensor a módulo *HAD* 2.

Paso 6. Insertar actuador en el catálogo

Para este paso usaremos la funcionalidad descrita en la sección **Insertar Actuador en el catálogo de actuadores**.

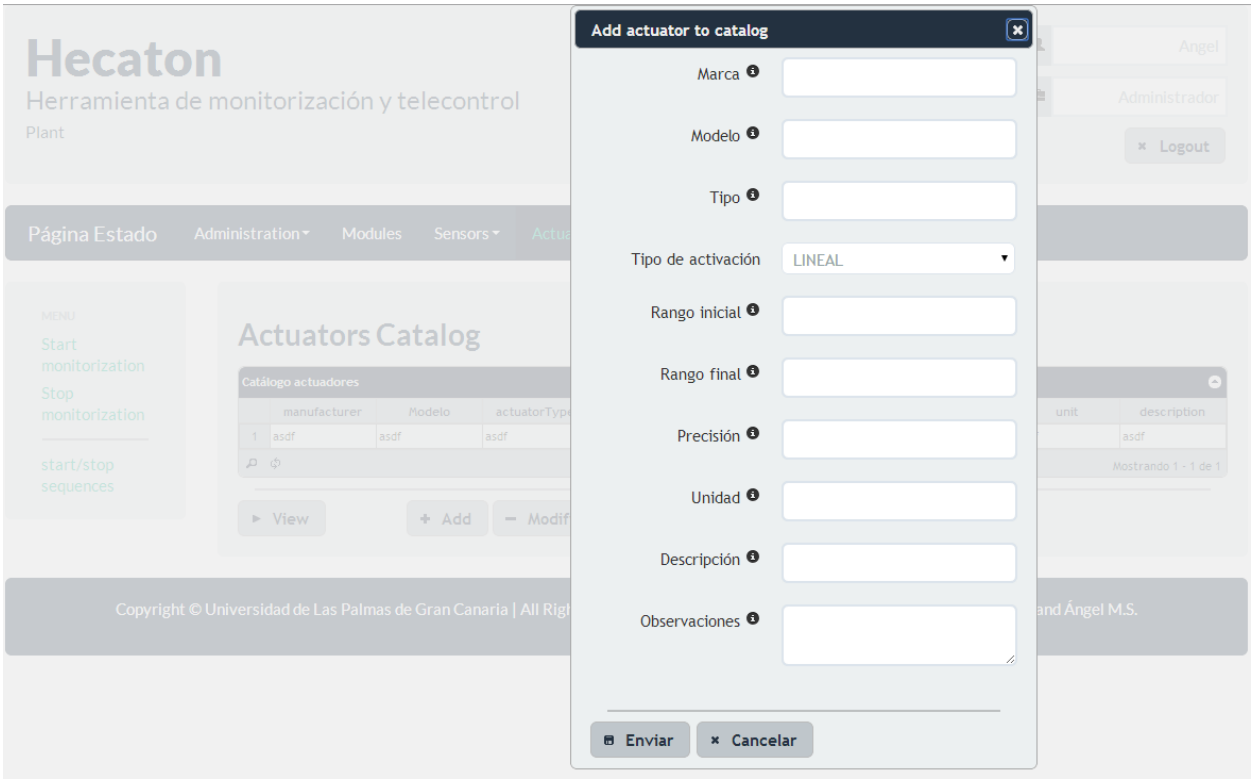

Ilustración 59: Inserción de un actuador en el catálogo.

Paso 7. Instalar actuador

Para este paso usaremos la funcionalidad descrita en la sección **Instalación de un actuador**.

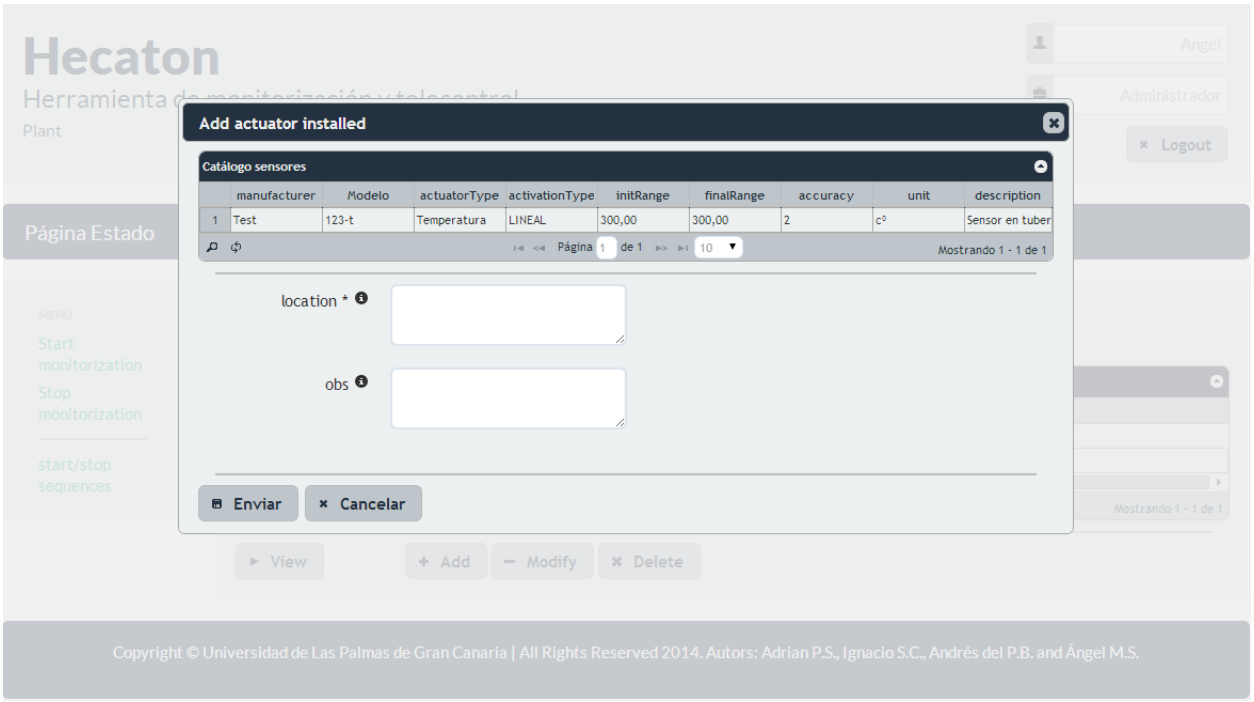

Ilustración 60: Instalación de un actuador.

Paso 8. Conectar actuador al *HAD*

Para este paso usaremos la funcionalidad descrita en la sección **Conectar un actuador a un módulo *HAD***.

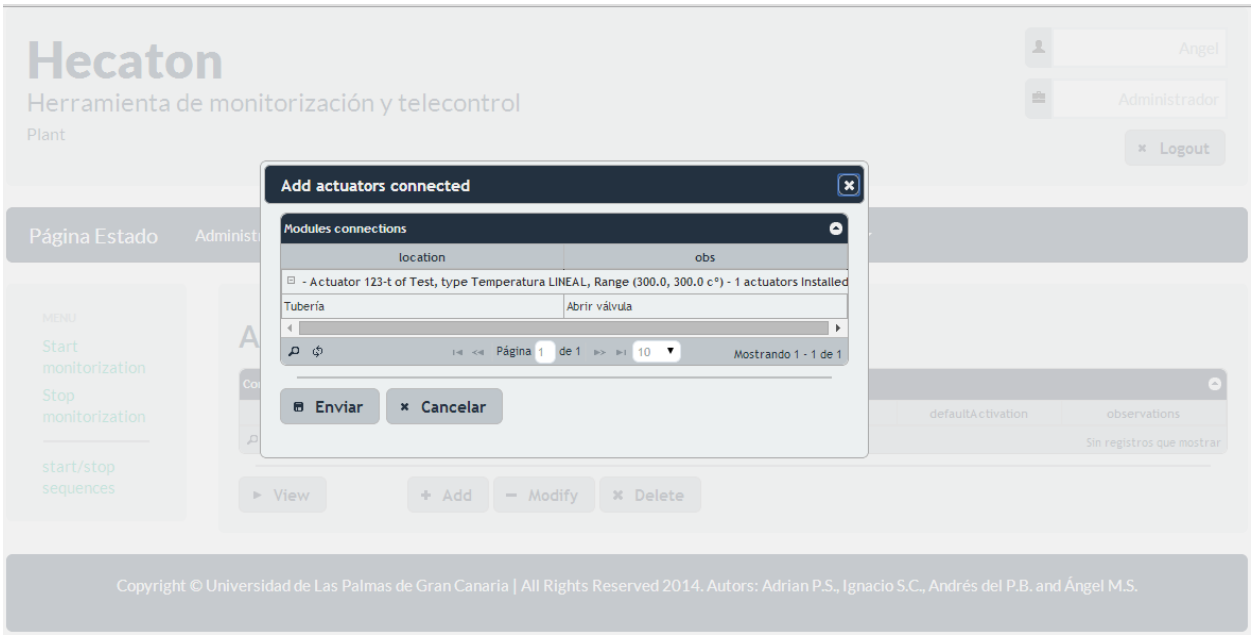

Ilustración 61: Conexión de un actuador a módulo *HAD.*

Paso 9. Insertar alarma

Para este paso usaremos la funcionalidad descrita en la sección **Insertar alarma en el sistema**.

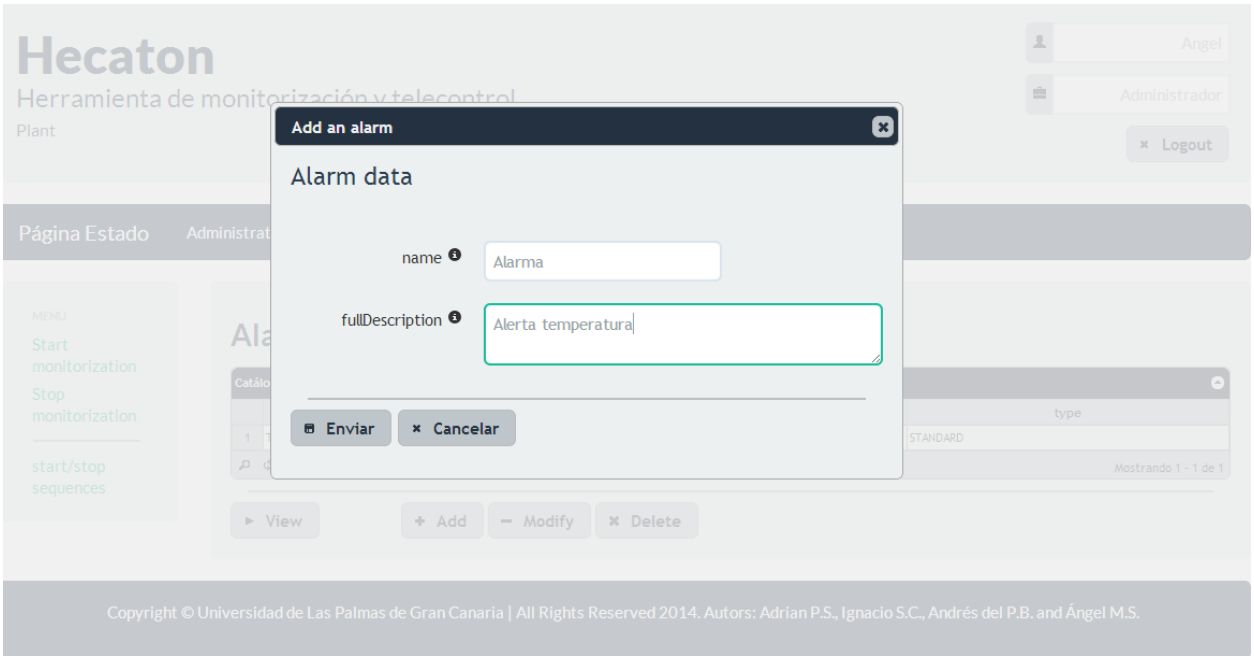

Ilustración 62: Inserción de alarma.

Paso 10. Insertar una secuencia

Para esto usaremos la funcionalidad descrita en la sección **Insertar secuencia de acciones**.

En primer lugar rellenamos los campos de información general.

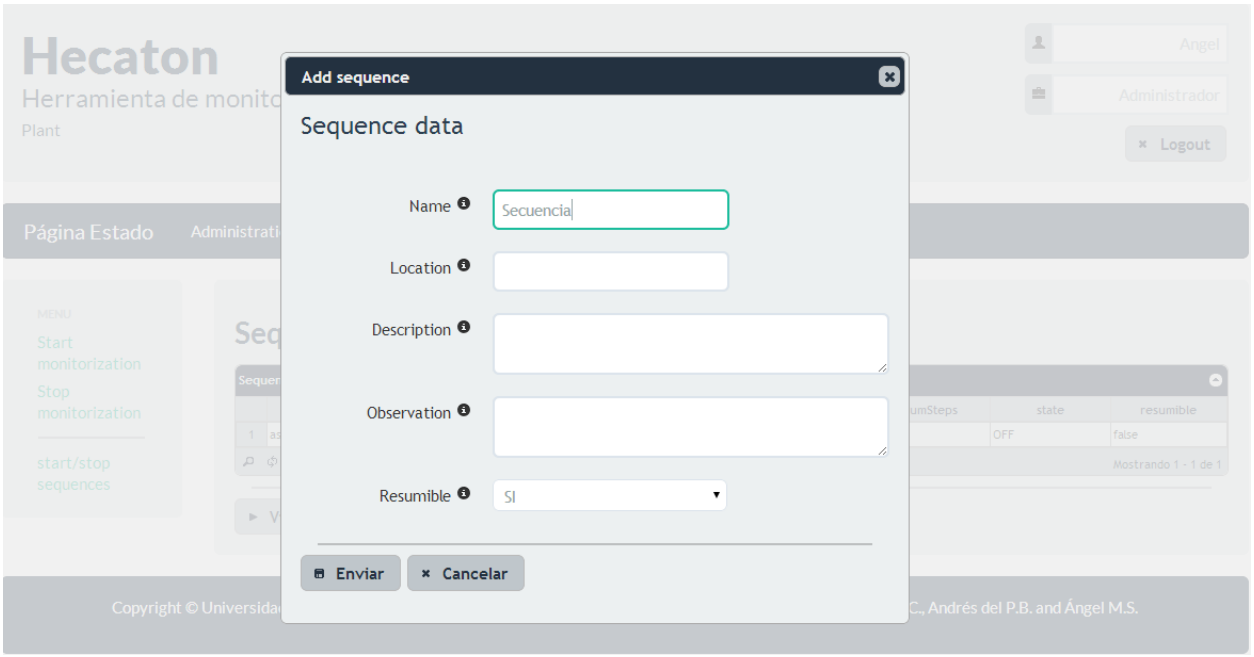

Ilustración 63: Información general de una secuencia.

A continuación llegamos a la pantalla de creación de los pasos a seguir por la secuencia.

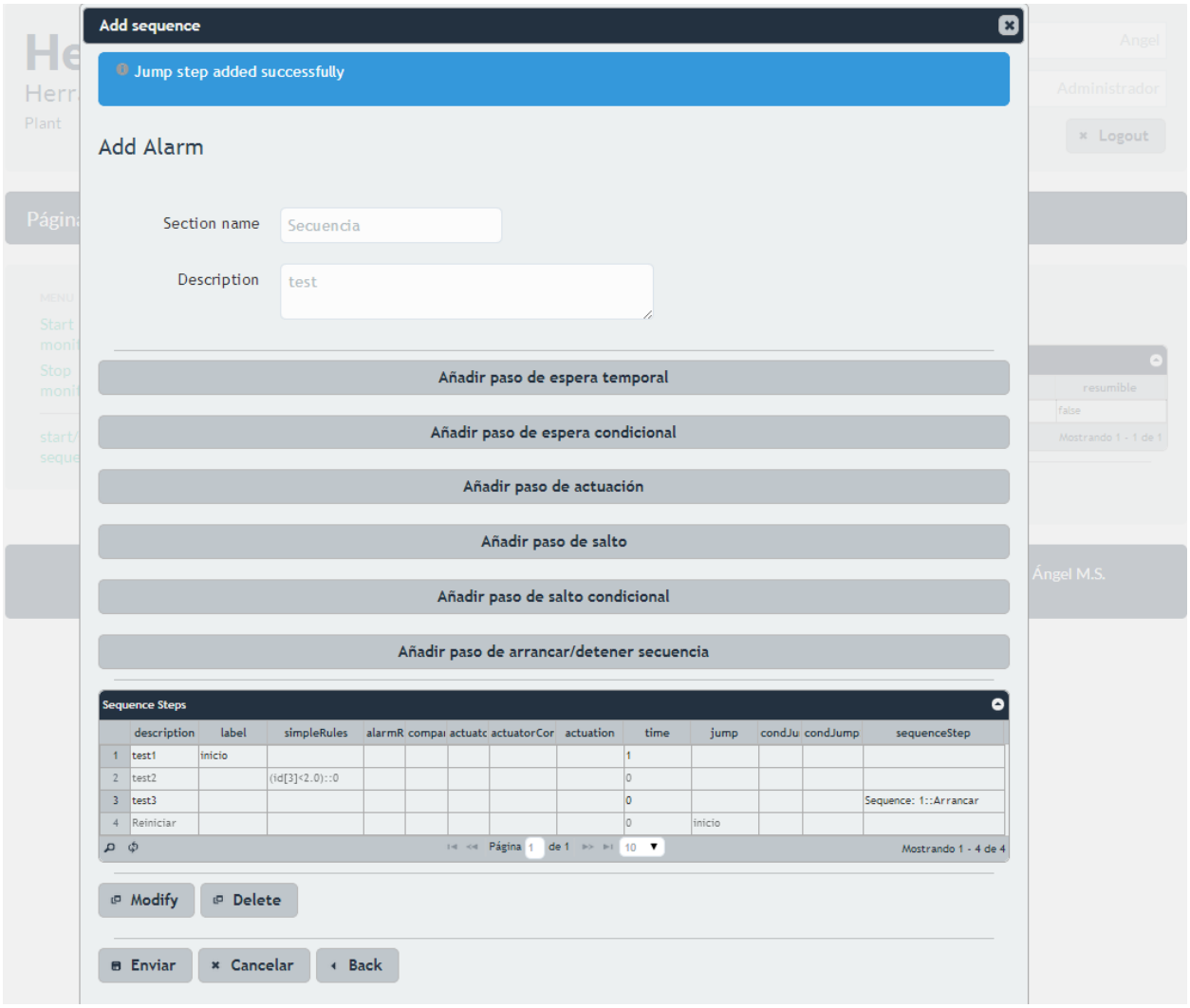

Ilustración 64: Inserción de pasos en una secuencia.

Paso 10.1. Insertar paso de espera temporal

Añadimos un paso de espera temporal tal y como ya se explicó en la sección **Insertar secuencia de acciones**.

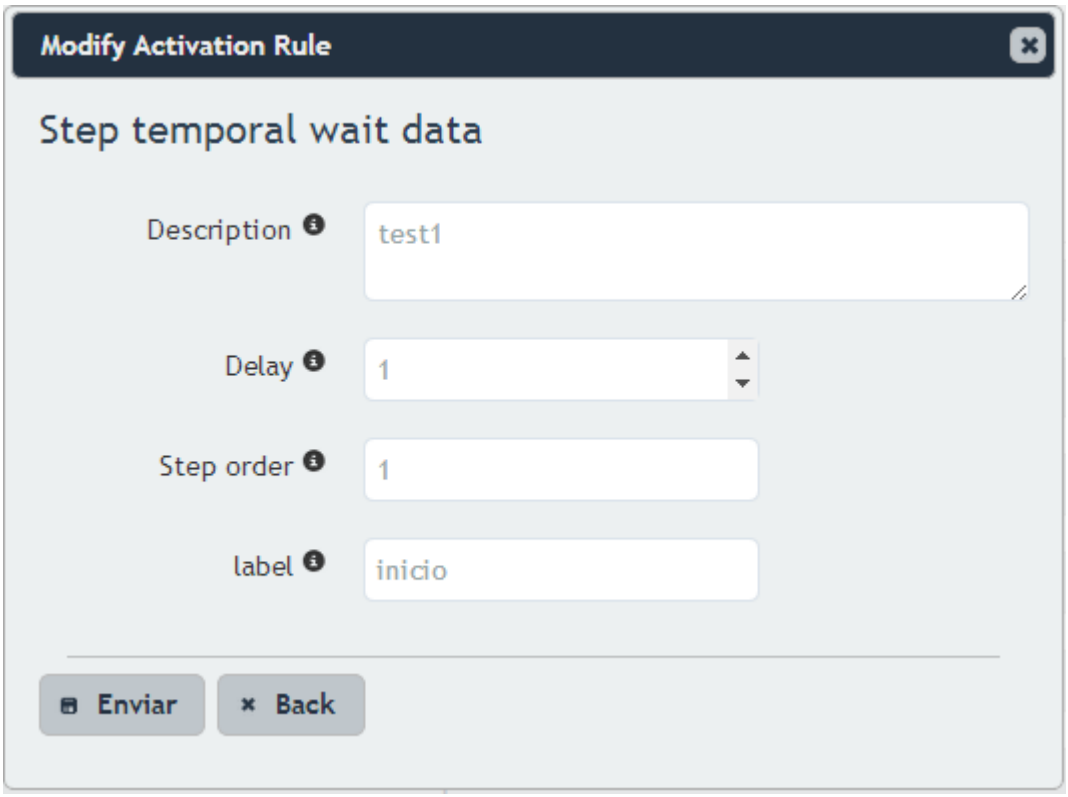

Ilustración 65: Inserción paso de espera temporal.

Paso 10.2. Insertar paso de actuación

Añadimos un paso de actuación tal y como ya se explicó en la sección **Insertar secuencia de acciones**.

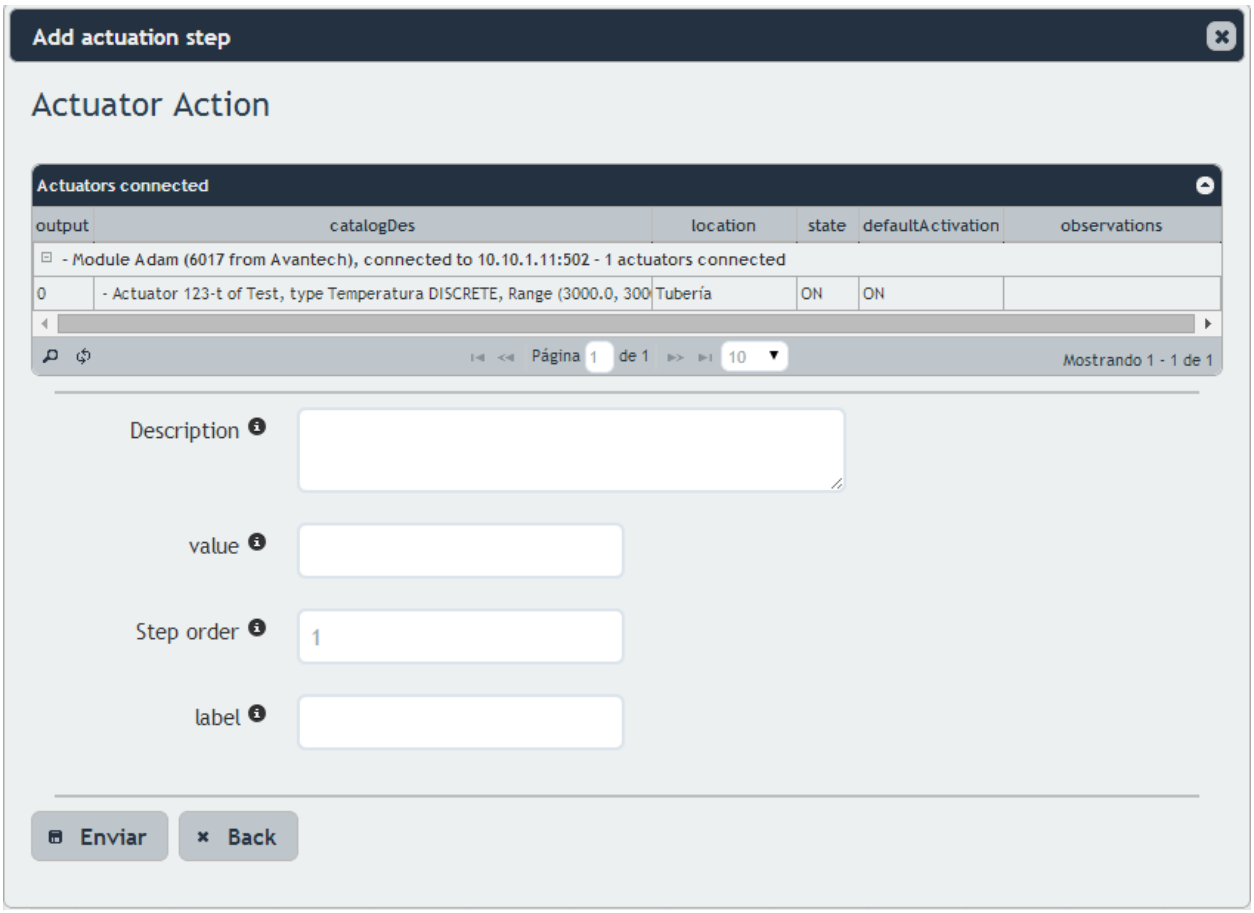

Ilustración 66: Inserción paso de actuación.

Paso 10.3. Insertar paso de espera condicional

Añadimos un paso de espera condicional tal y como ya se explicó en la sección **Insertar secuencia de acciones**.

Ilustración 67: Inserción paso de espera condicional 1.

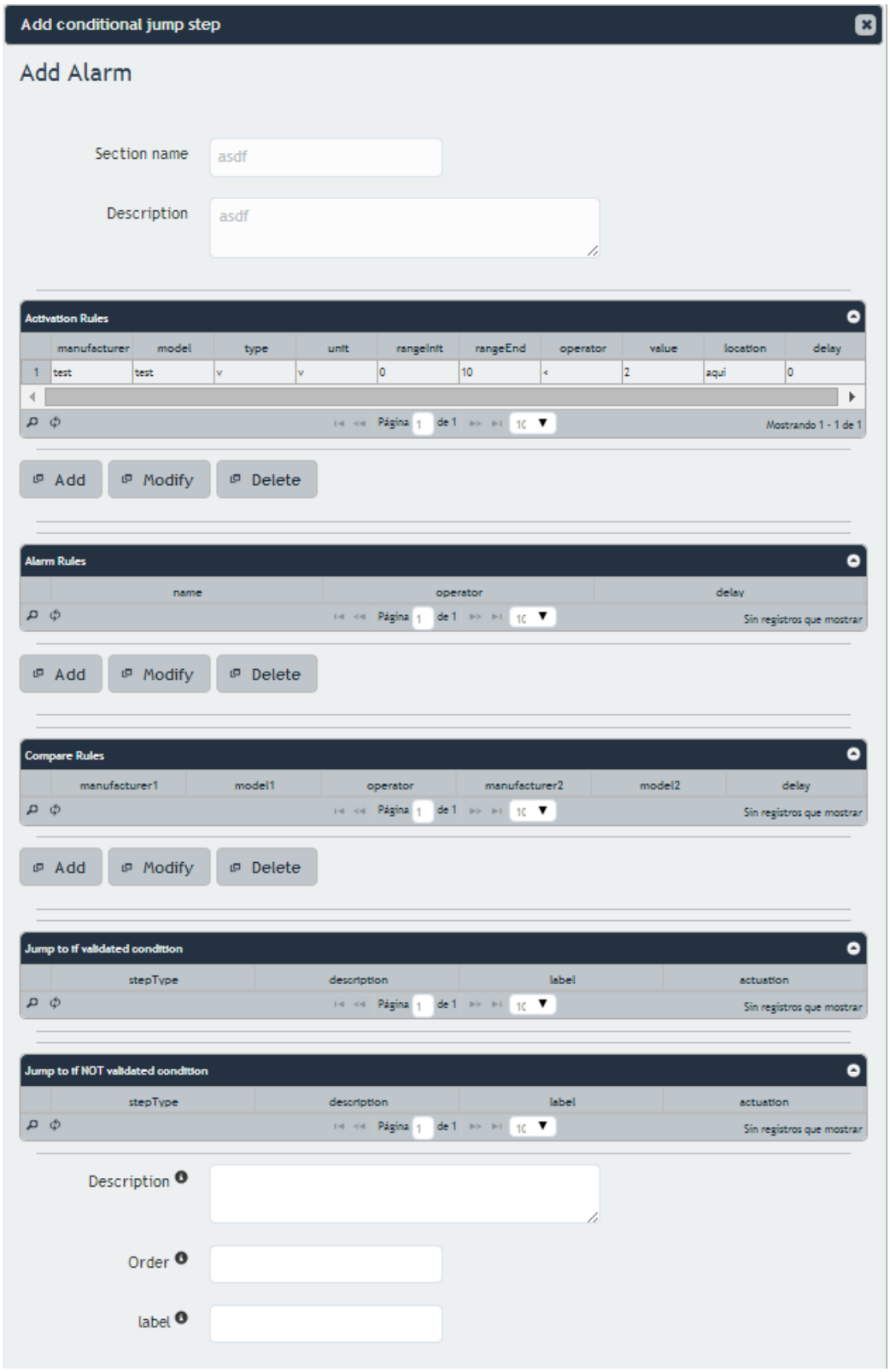

Ilustración 68: Inserción paso de espera condicional 2.

Paso 10.4. Insertar paso de salto condicional

Añadimos un paso de salto condicional tal y como ya se explicó en la sección **Insertar secuencia de acciones**.

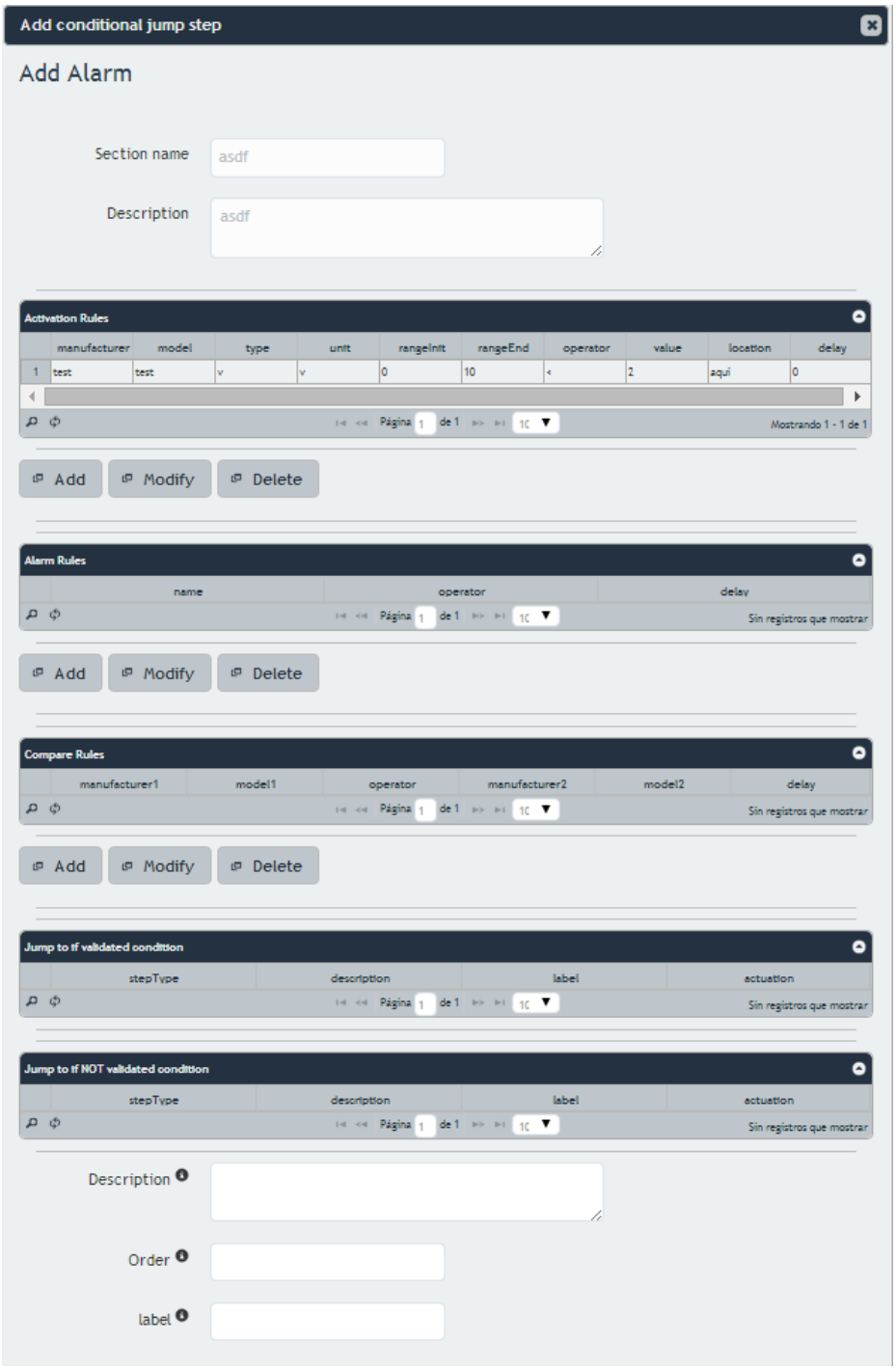

Ilustración 69: Inserción paso de salto condicional 1.

Paso 10.5. Insertar paso de arranque/paro de secuencia

Añadimos un paso de arranque/paro de secuencia tal y como se explicó en la sección **Insertar secuencia de acciones**.

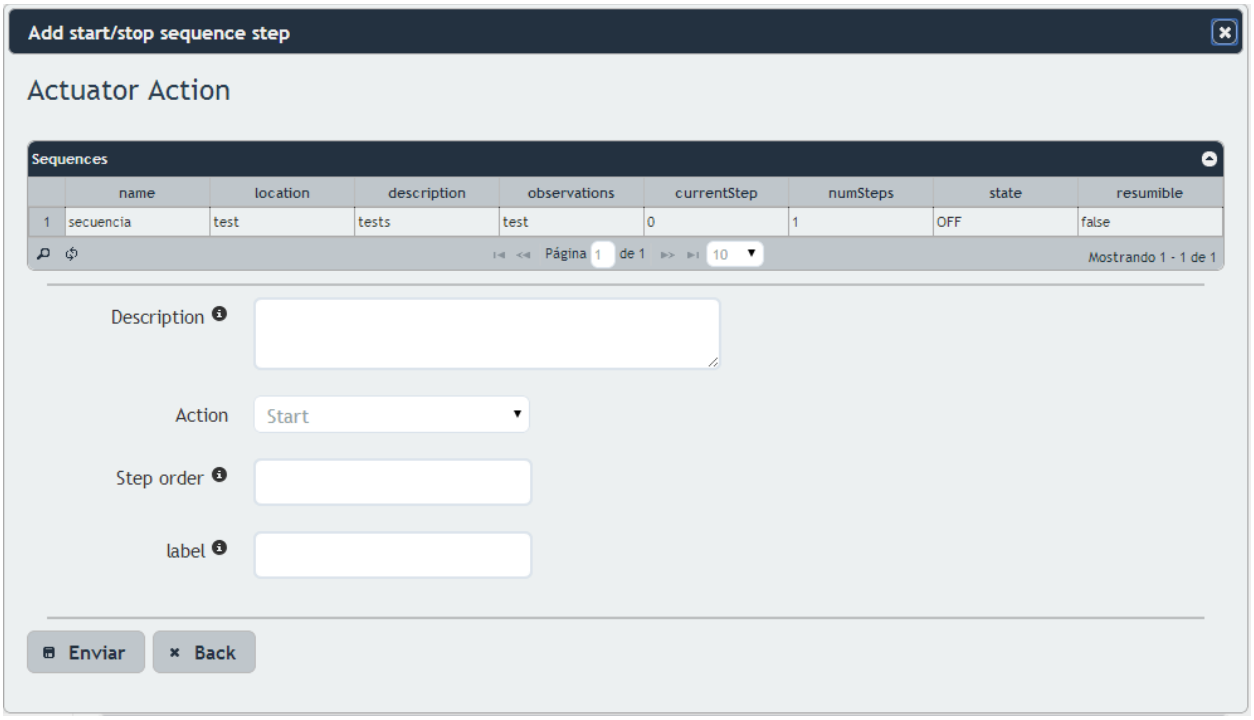

Ilustración 70: inserción paso de arrancar/detener secuencia.

Paso 10.6. Insertar paso de salto

Añadimos un paso de salto tal y como se explicó en la sección **Insertar secuencia de acciones**.

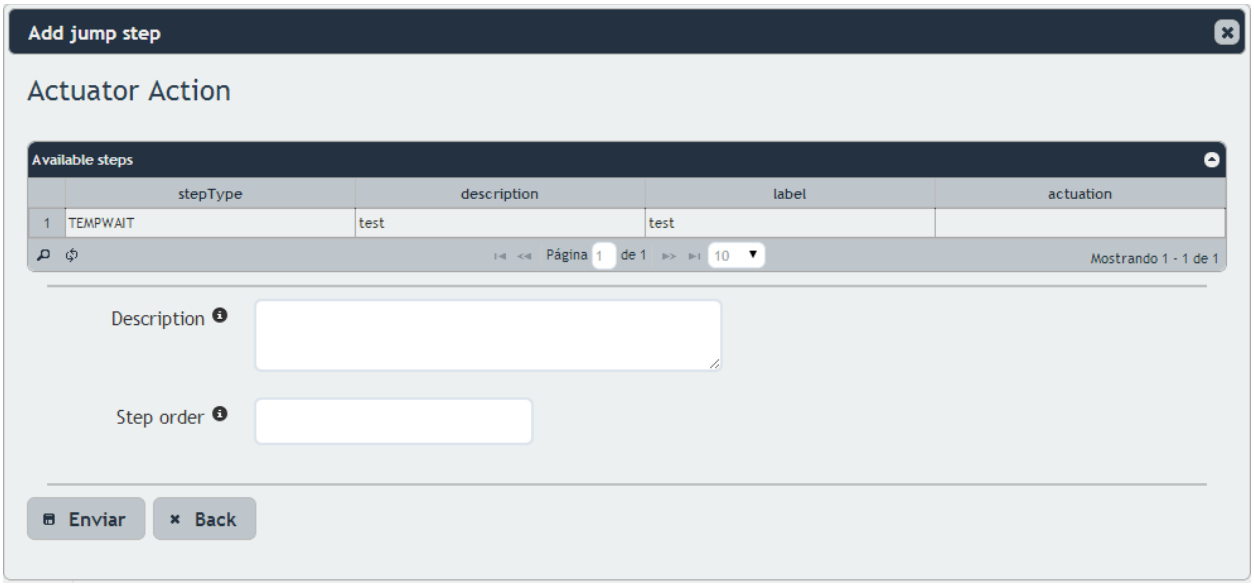

Ilustración 71: Inserción paso de salto.

Paso 11. Crear sección

Para este paso usaremos la funcionalidad descrita en **Crear Sección**.

Introducimos la información general de la sección.

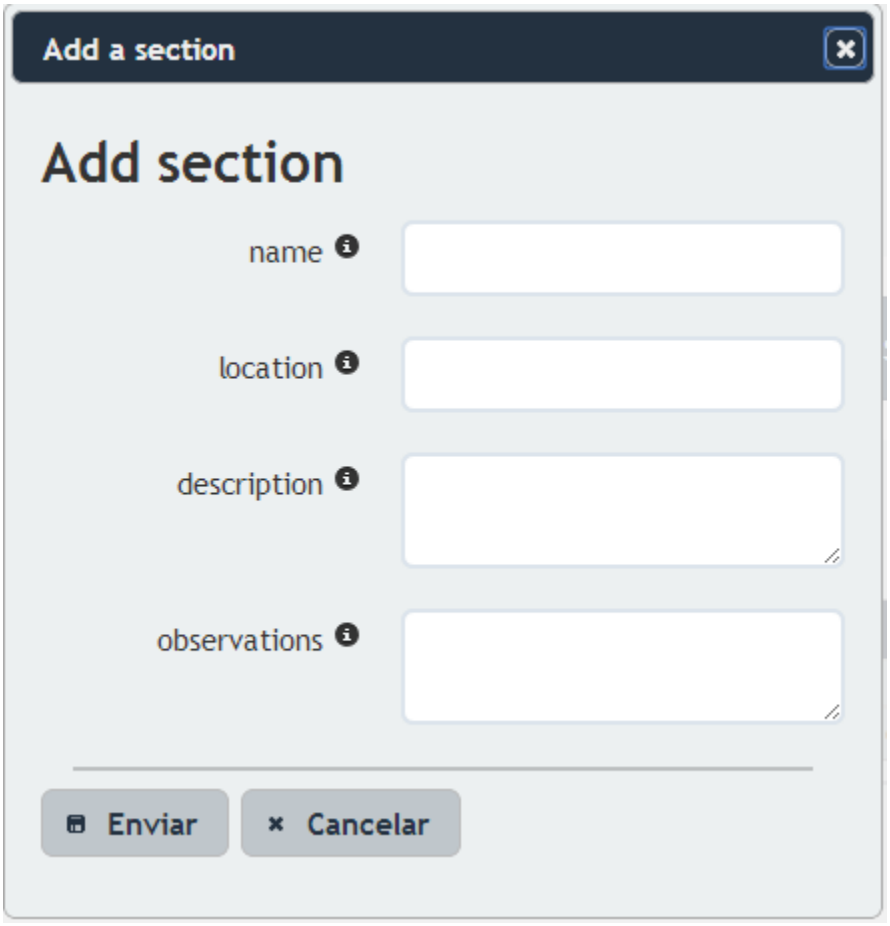

Ilustración 72: Creación de sección - datos iniciales.

A continuación pasamos a la pantalla de inserción de elementos en la sección, donde podremos ver los que ya hemos ido añadiendo a la misma.

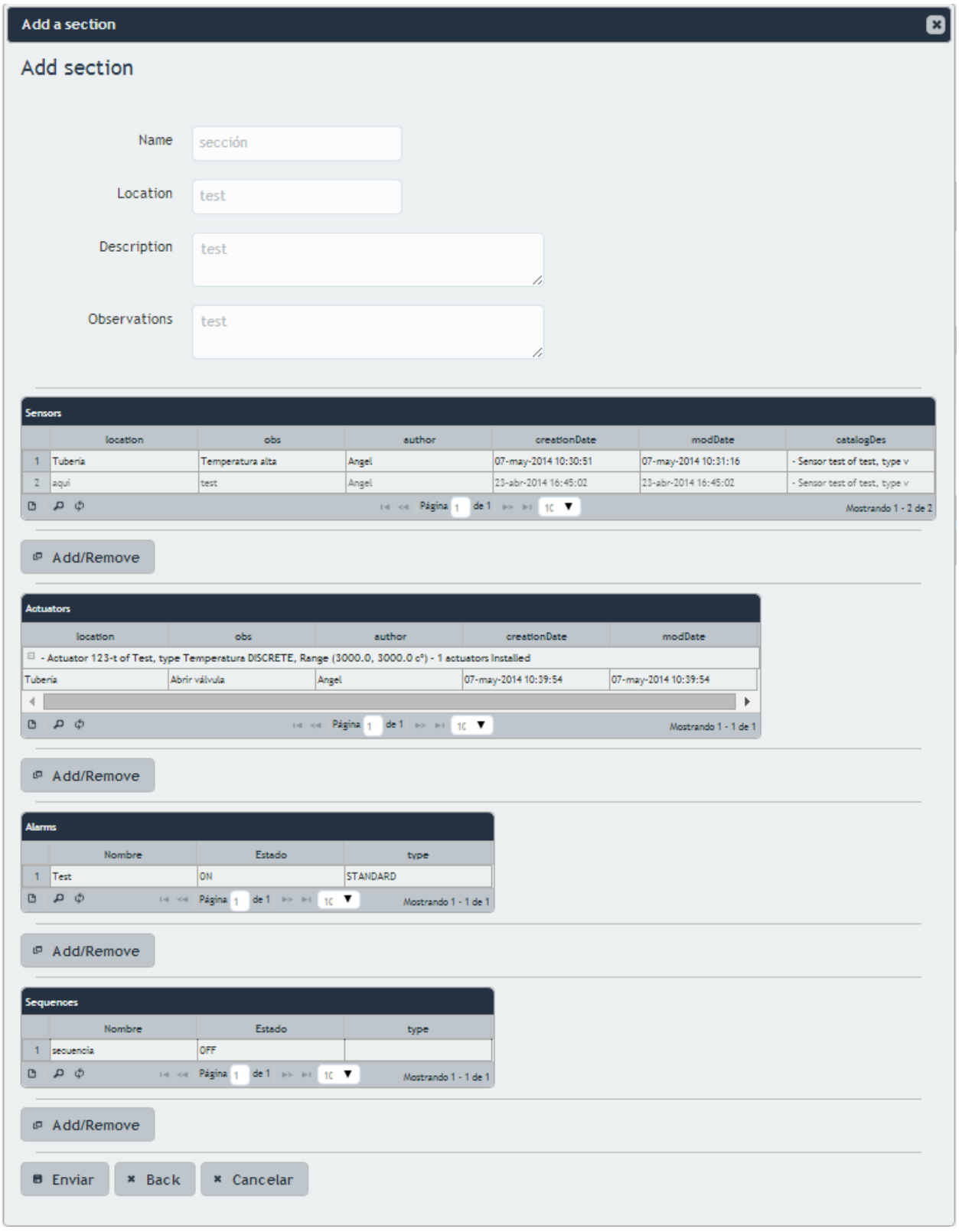

Ilustración 73: Creación de sección - Especificación de componentes.

Paso 12. Arrancar motor de alarmas

Para este paso usaremos la funcionalidad descrita en la sección **Arrancar motor de alarmas**.

Ilustración 74: Arrancar motor de alarmas.

Paso 13. Visualizar estado de la planta

Para este paso usaremos la funcionalidad descrita en la sección **Visualizar estado de la planta**.

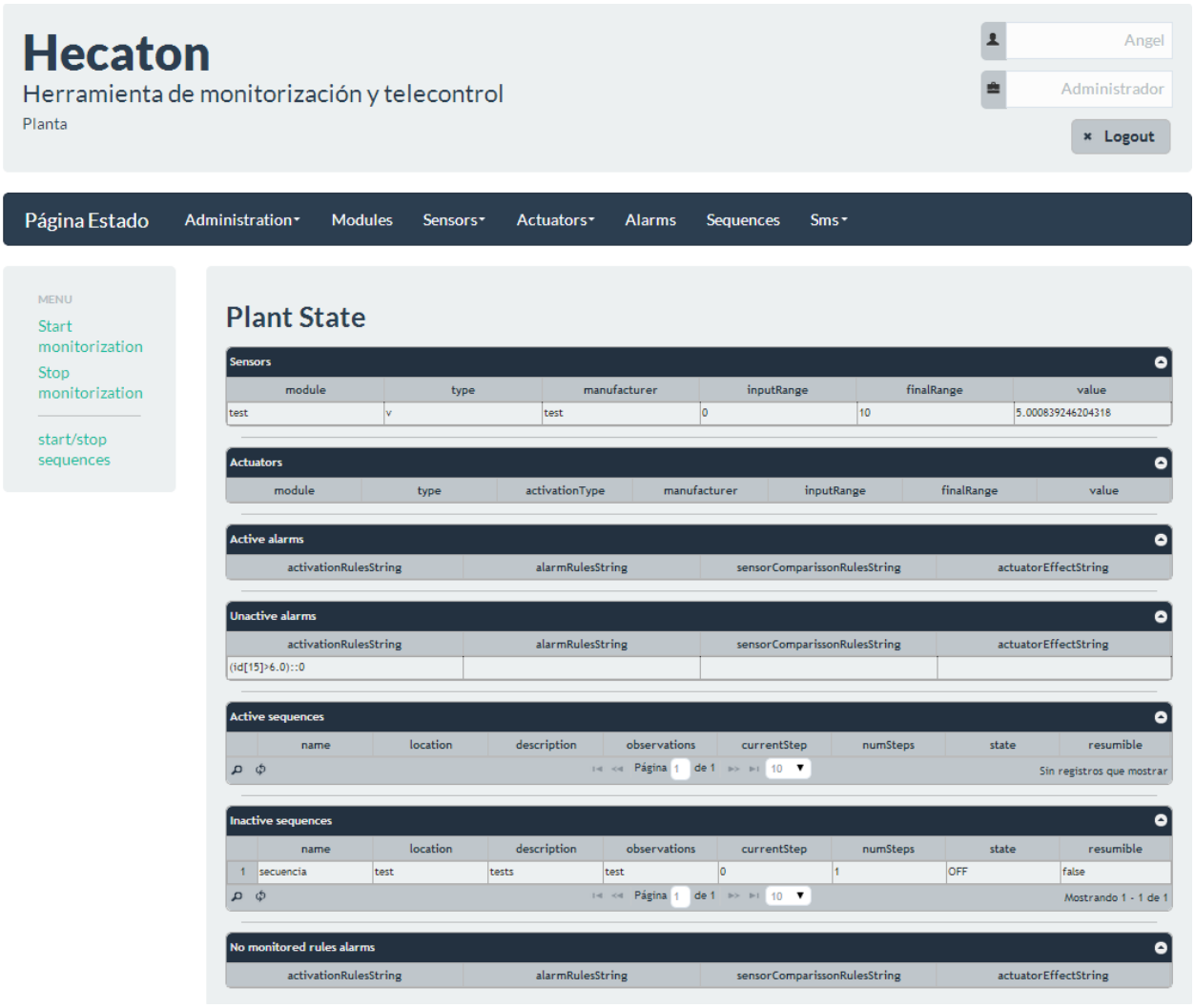

Ilustración 75: Estado actual de la planta 1.

En esta vista del estado actual de la planta podemos ver el listado de sensores, actuadores, alarmas activas, las secuencias activas e inactivas, también las alarmas inactivas y las alarmas que no están siendo monitorizadas.

.

Paso 14. Adjuntamos diagrama a imagen de la planta

Para este paso usaremos la funcionalidad descrita en la sección **Adjuntar plano**. Podremos adjuntar un plano distinto por sección.

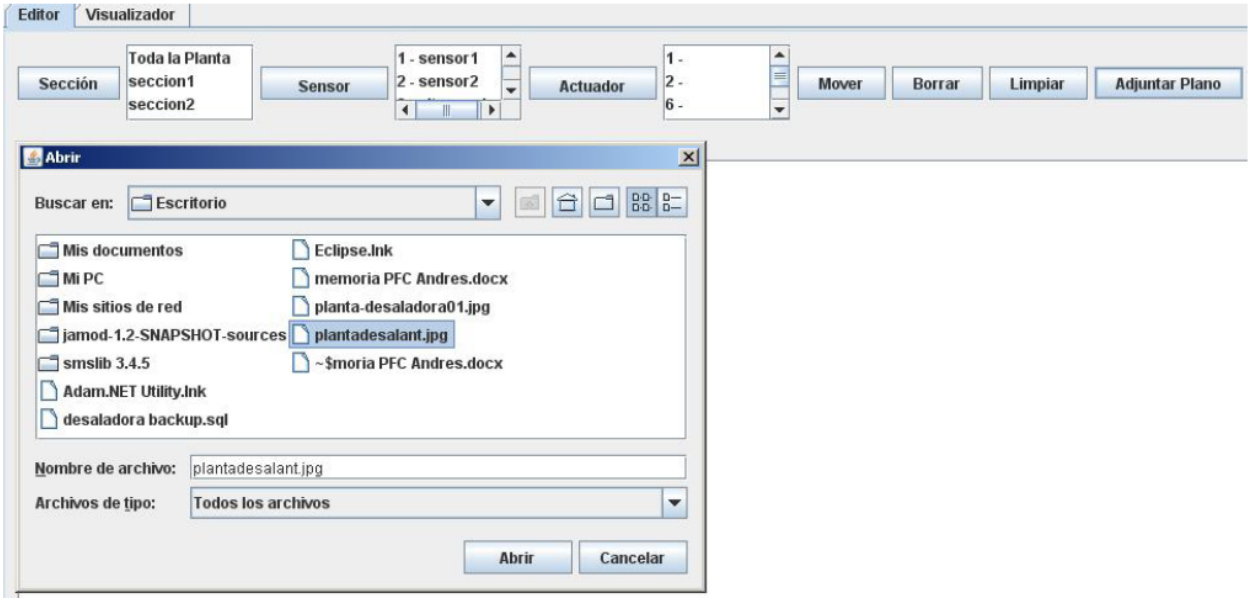

Ilustración 76: Adjuntar diagrama a *applet*.

Paso 15. Añadir sensor

Para este paso usaremos la funcionalidad descrita en la sección **Añadir visualización de sensor**.

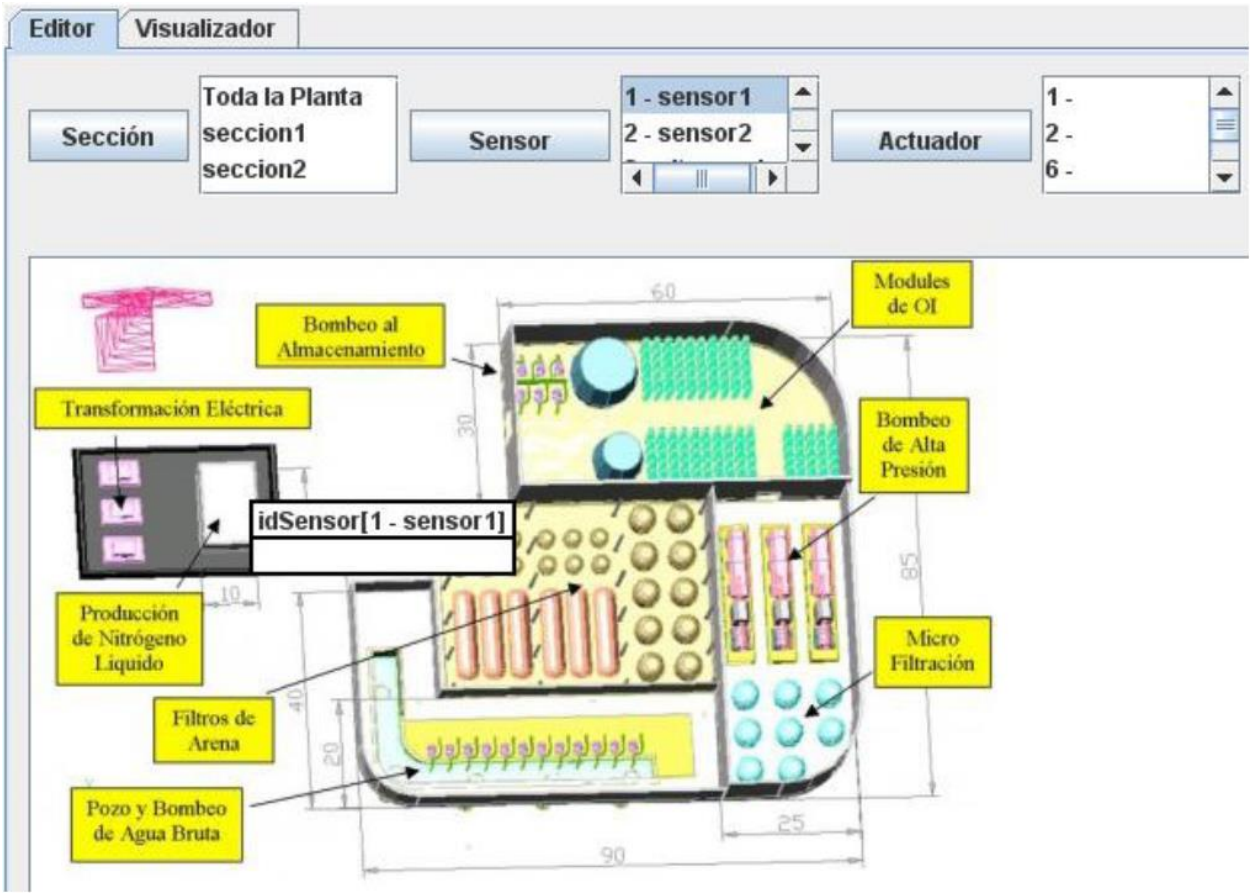

Ilustración 77: Añadir sensor *applet*.

Paso 16. Añadir actuador

Para este paso usaremos la funcionalidad descrita en la sección **Añadir visualización de actuador**.

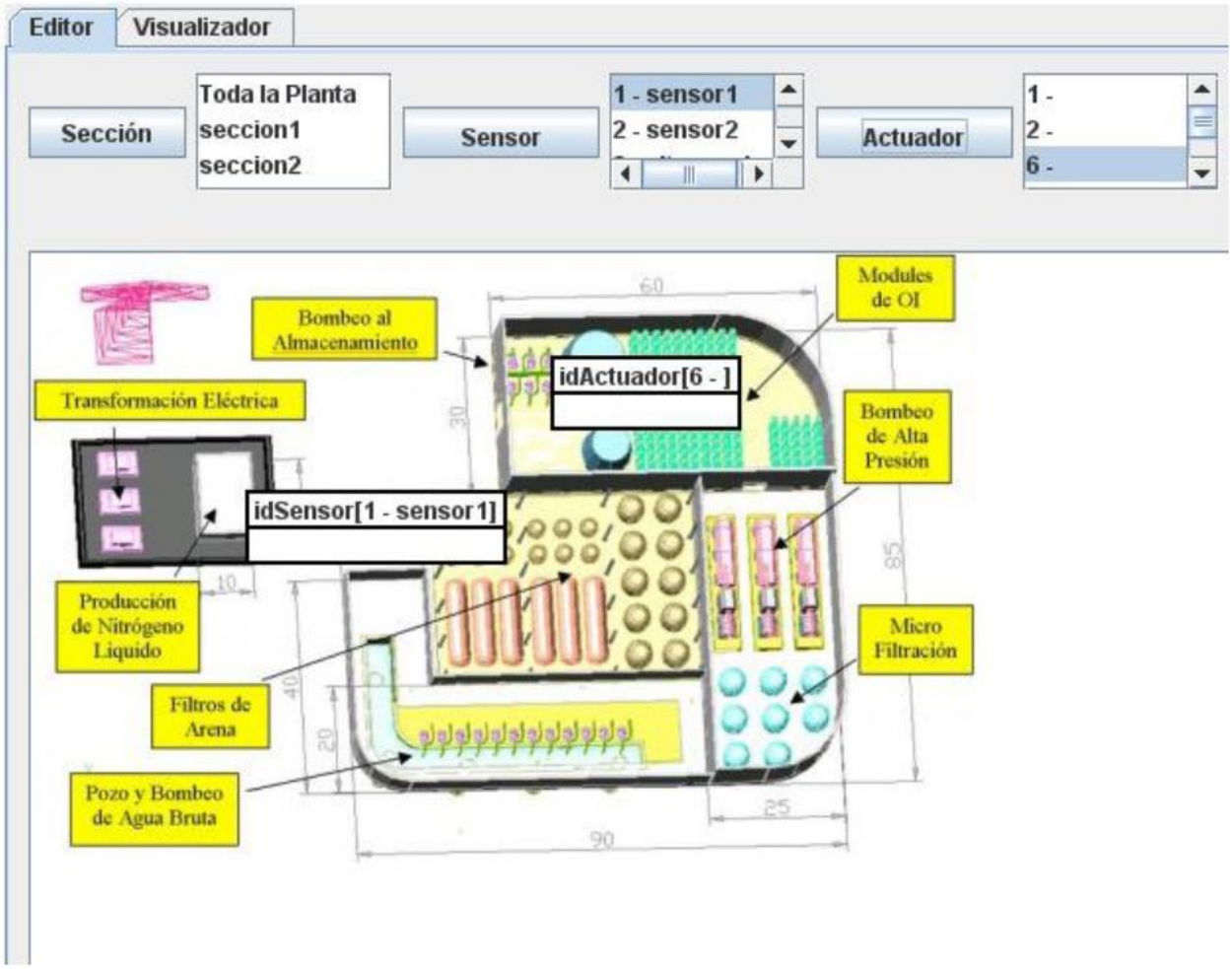

Ilustración 78: Añadir actuador *applet*.

Paso 17. Guardar cambios en la imagen

Para este paso usaremos la funcionalidad descrita en la sección **Guardar diseño de planta**.

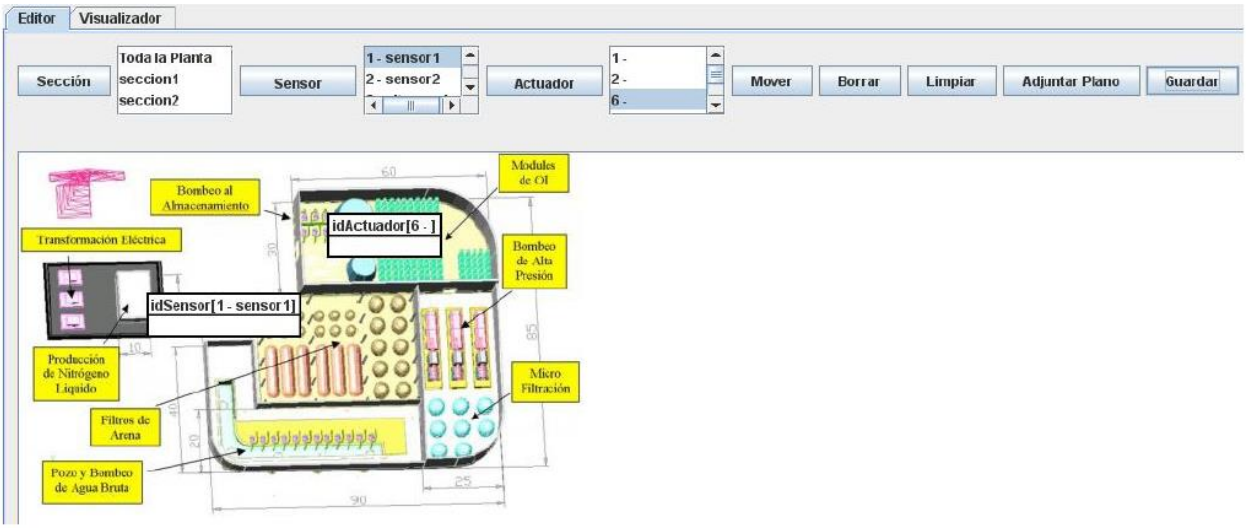

Ilustración 79: Guardar cambios *applet*.

Paso 18. Cambiar de sección

Para este paso usaremos la funcionalidad descrita en la sección C**ambiar sección editor/visualizador de la planta**.

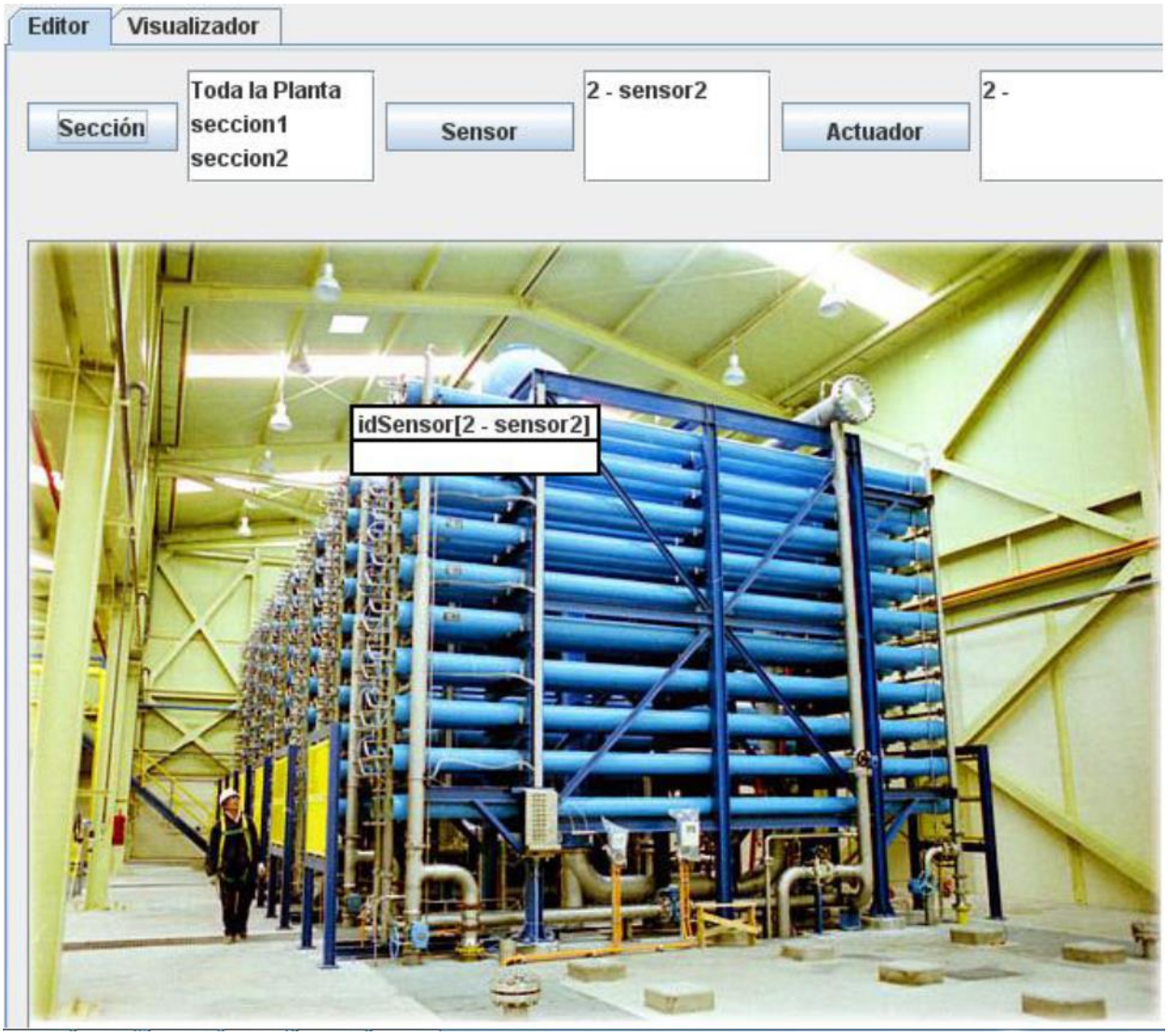

Ilustración 80: Cambiar sección *applet*.

Paso 19. Cargar imagen desde el servidor

Para este paso usaremos la funcionalidad descrita en la sección **Cargar imagen de la planta**.

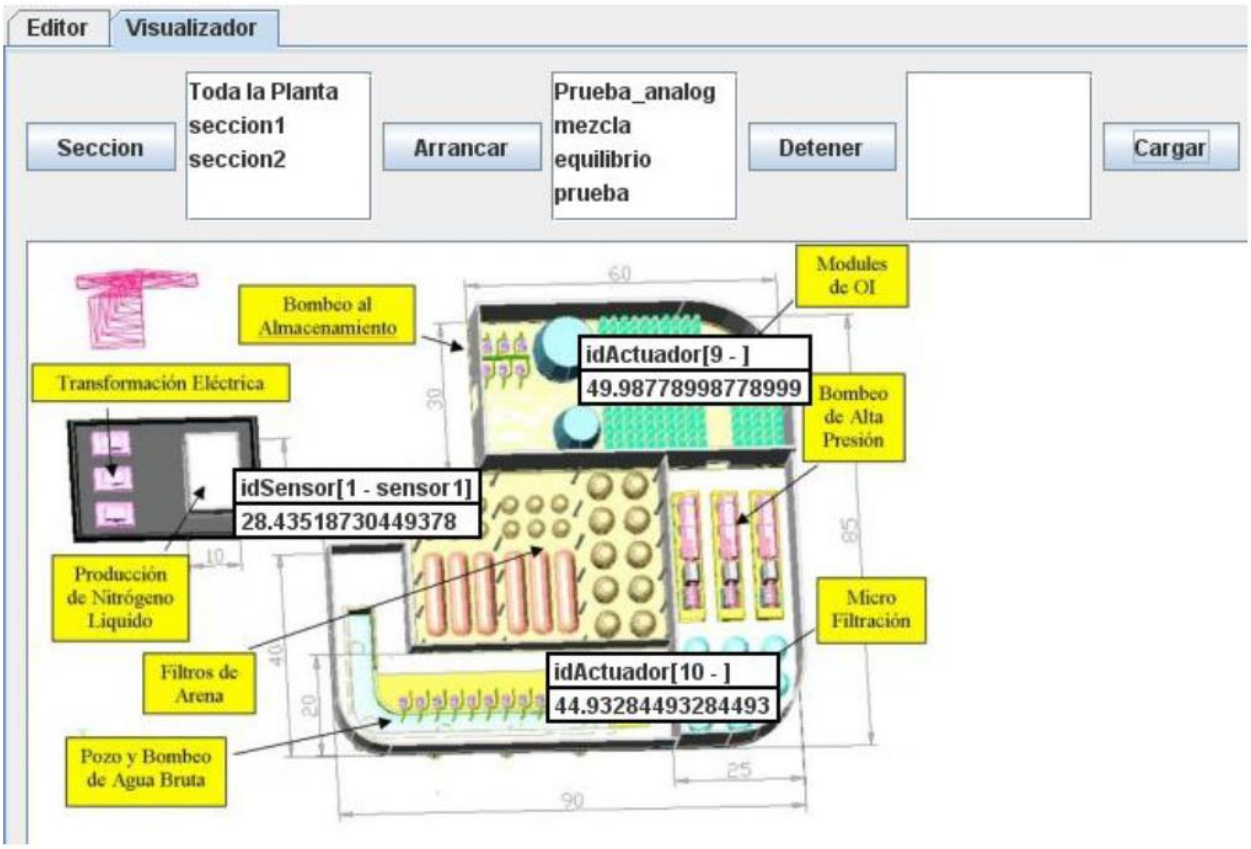

Ilustración 81: Cargar imagen servidor *applet*.

Paso 20. Arrancar secuencia de acciones

Para este paso usaremos la funcionalidad descrita en la sección **Arrancar secuencia de acciones**.

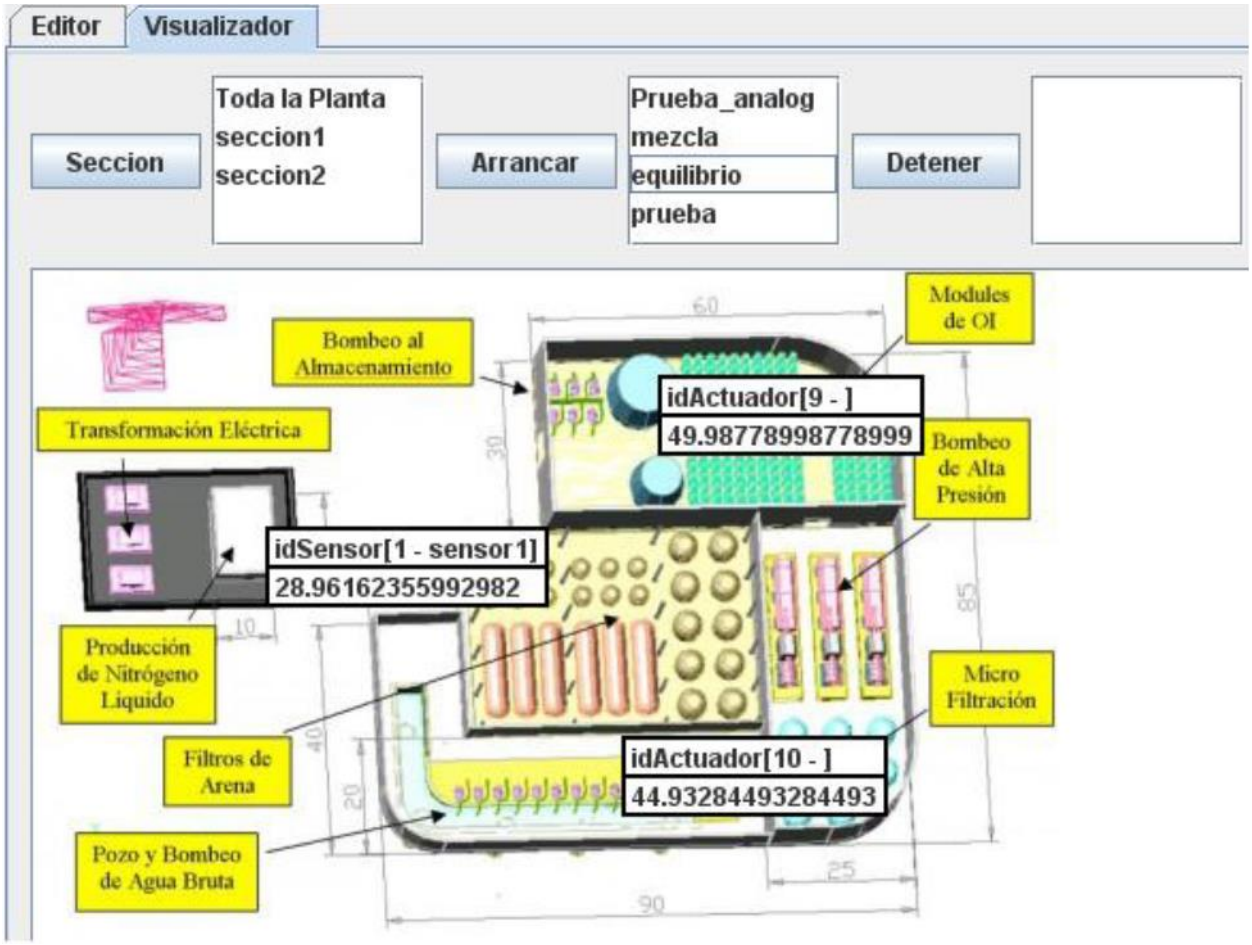

Ilustración 82: Arrancar secuencia de acciones *applet*.

Paso 21. Detener secuencia de acciones

Para este paso usaremos la funcionalidad descrita en la sección **Detener secuencia de acciones**.

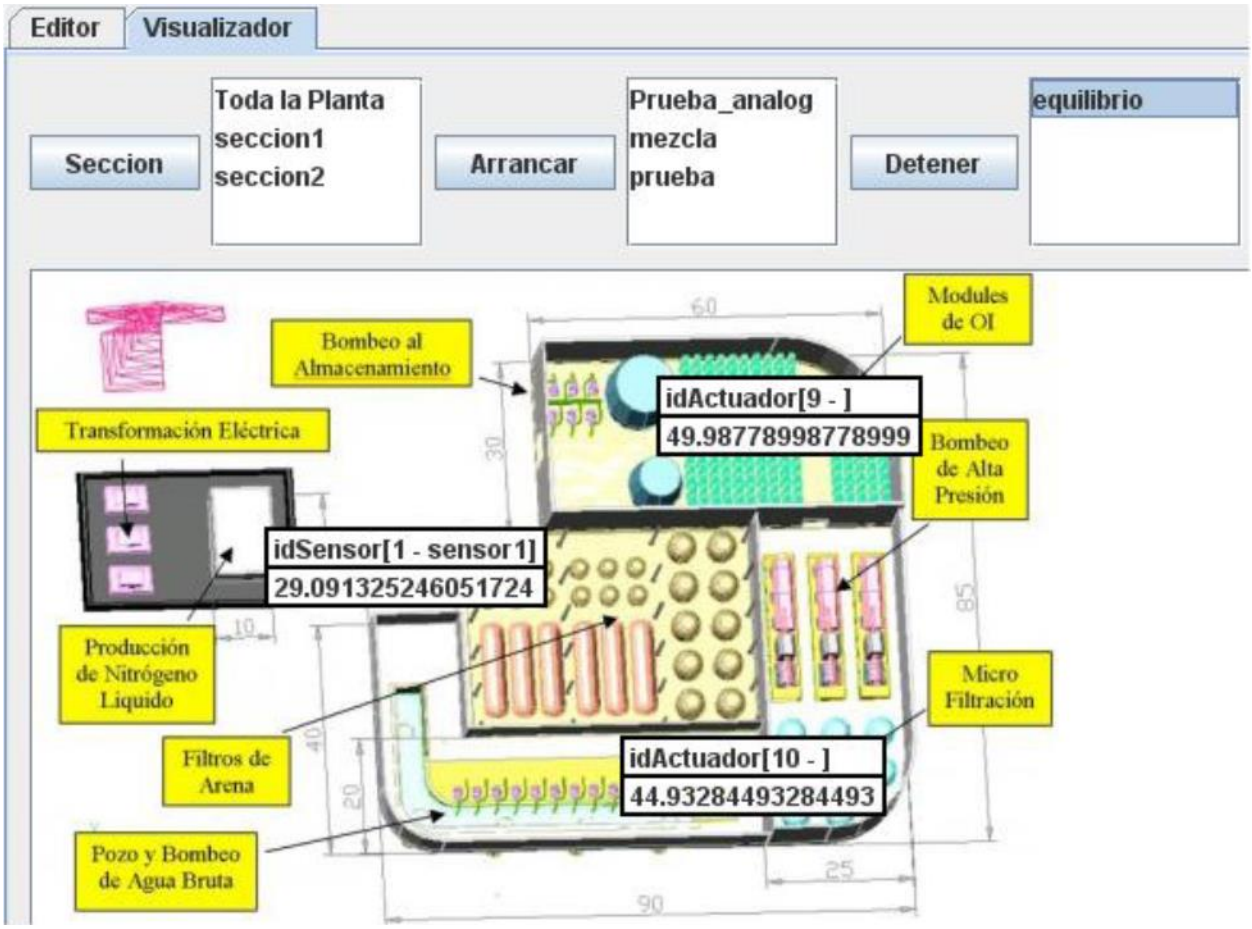

Ilustración 83: Detener secuencia de acciones *applet*.

Paso 22. Modificar valor actuador

Para este paso usaremos la funcionalidad descrita en la sección **Modificar valor actuador**.

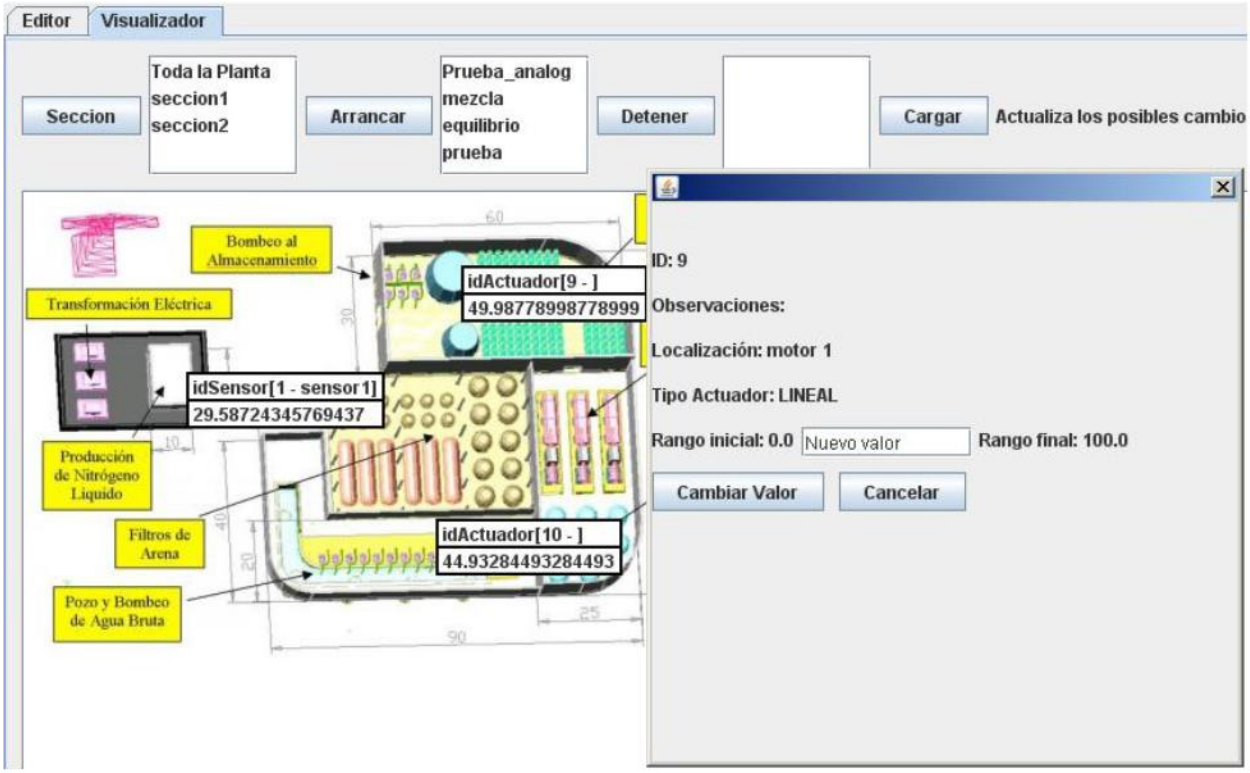

Ilustración 84: Modificar valor actuador *applet*.

Anexo B. Manual para el desarrollador

Este manual describe los pasos a seguir para instalar tanto la aplicación como las herramientas de desarrollo necesarias para continuar con el proyecto. Incluiremos una guía para sistemas *Windows* y *Linux* en su distribución *Ubuntu* (válido para cualquier distribución derivada *Debian*).

Casi todo lo que es necesario para continuar con el desarrollo del proyecto se encuentra en el DVD del proyecto dentro de la carpeta “/telecontrol”. Copia esta y todo su contenido en tu sistema. Indicar que se trata de versiones para sistemas con arquitecturas de 64 bits. A continuación sigue los siguientes pasos tener todo configurado y listo para empezar. También añadiré algunos detalles si prefieres instalar las herramientas desde cero o nuevas versiones.

b.1. Instalación del "*Java Development Kit*" (*JDK*)

El Kit de Desarrollo Java es la versión para los desarrolladores. Incluye el compilador de *Java* (*javac*), *JRE* (entorno de ejecución) y *JVM* (máquina virtual *Java*). La versión instalada en esta guía corresponde a la *JDK 7*.

- En *Ubuntu*:

En una consola y con los permisos adecuados solamente es necesario ejecutar este comando:

```
sudo apt-get install openjdk-7-jdk
```

- En *Windows*:

Podemos buscar en cualquier buscador de internet por “JDK”. También podemos irnos directamente a sitio web de Oracle: http://www.oracle.com/ donde encontraremos el software. Descargamos el JDK y una vez descargado lo instalaremos:

b.2. Instalación de *MySQL*

Lo siguiente será instalar el servidor de base de datos *MySQL* (versión en este proyecto 5.5):

- En *Ubuntu*:

  Ejecutamos el comando:

  ```
  sudo apt-get install mysql-server
  ```

- En *Windows*:

  Descargamos el servidor desde la página de *MySQL*. Debemos tener en cuenta que necesitamos la versión libre cuyo nombre continene “comunity”: http://dev.mysql.com/downloads/windows/installer/ e instalamos.

  Durante el proceso de instalación del servidor de base de datos se requerirá introducir una contraseña para el usuario *root* del mismo. Es importante estar atento a este paso y conservar la contraseña usada que nos permitirá acceder a la administración de la misma.

  Si bien no es necesario añadir nada a la base de datos ya que el motor de persistencia *Hibernate* creará las tablas por nosotros cuando iniciemos la aplicación, recomiendo importar la base de datos desde el script *SQL* que se ha añadido en el DVD del proyecto “/telecontrol/Apps/telecontrol.sql”. De tal manera que no tendremos una base de datos vacía la primera vez que abramos la aplicación. Para ello existen muchas maneras y herramientas. Podríamos importar fácilmente desde la línea de comando pero se recomienda instalar la herramienta de gestión de *MySQL* llamada “*Workbench*”. Esta nos ofrece una cómoda interfaz gráfica desde donde podemos mantener nuestra base de datos. Muy recomendable para el desarrollador.

- En *Ubuntu* puede ser instalada con el siguiente comando:

```
sudo apt-get install mysql-workbench
```

- En *Windows* debería estar instalada si en el paso anterior instalarse el “Windows Installer”. Si no puedes encontrar el *Workbench* en el mismo enlace.

b.3. Obtener el entorno de desarrollo *Eclipse*

Desde su inicio, este proyecto ha sido desarrollado utilizando el entorno de desarrollo Eclipse. Concretamente en su versión para desarrollo en *Java EE*. Al ser java funciona con cualquier plataforma donde esté instalada la máquina virtual. En el DVD de este proyecto (“/telecontrol/eclipse”) se ofrece una copia de *Eclipse Juno* que incluye todos aquello *plugin* utilizados para desarrollar la aplicación.

Si deseas instalar una nueva versión desde cero, se recomienda encarecidamente que instales las ***JBoss Tools*** (Instalará todo lo que necesitas, desde el *plugin* de ***Maven*** hasta ***Git***) y ***JRebel***. Ambas las podrás encontrar en el “*Marketplace*” de Eclipse.

b.4. Obtener el servidor de aplicaciones *JBoss AS 7.1.1*

En el DVD del proyecto encontrarás el servidor ya preparado para funcionar (“/telecontrol/jboss/jboss-as-7.1.1.Final/”). Si deseas descargarlo de nuevo simplemente tendrás que modificar el fichero de configuración tal de manera semejante al presente en el DVD (dentro de Jboss,” \standalone\configuration/standalone.xml/).

Además en este fichero “standalone.xml” deberás indicar la contraseña del usuario “*root*” de la base de datos que has instalado previamente (o un usuario que tenga los permisos adecuados). Busca el subsistema siguiente:

```
<subsystem xmlns="urn:jboss:domain:datasources:1.0">
	<datasources>
		<datasource jta="true" jndi-name="java:jboss/datasources/MySQLDB" pool-
		name="MySQLDB" enabled="true" use-java-context="true" use-ccm="true">
			<connection-url>jdbc:mysql://localhost:3306/</connection-url>
```

```
                <driver>com.mysql</driver>
                <security>
                        <user-name>root</user-name>
                        <password>1234</password>
                </security>
                <statement>
                        <prepared-statement-cache-size>100</prepared-statement-
                cache-size>
                        <share-prepared-statements>true</share-prepared-
                statements>
                </statement>
        </datasource>
        <drivers>
                <driver name="com.mysql" module="com.mysql"/>
        </drivers>
    </datasources>
</subsystem>
```

Y modifica la contraseña en el *tag* “security”.

a.5. Obtener *Maven*

La versión 3.1.1 de *Maven* se encuentra en la carpeta /telecontrol/Apps del proyecto. En la misma carpeta se encuentra el repositorio local de *Maven* donde están todas las librerías necesarias para compilar el proyecto. Dentro de la carpeta “/telecontrol/Apps/apache-maven-3.1.1/conf/” se encuentra el fichero de configuración settings.xml donde se debe indicar los repositorios *Maven* remotos y la ruta al repositorio local. Revisa la ruta de este último para que coincida con la tuya.

Si deseas instalar *Maven* de otra manera, ten en cuenta que debes añadir los repositorios remotos presente en *settings.xml* que se encuentra en *Apps*. También hay una serie de librerías que no están presentes en ningún repositorio *Maven*. Estas se encuentran en la carpeta “/telecontrol/Libs”. De manera que deberás añadirlas manualmente a tu repositorio local. Los siguientes comandos te ayudan:

- Librería *JAMOD*:

```
mvn install:install-file -DgroupId=jamod -DartifactId=jamod -Dversion=1.2 -Dpackaging=jar -Dfile=E:\jamod-1.2-SNAPSHOT.jar -DgeneratePom=true
```

- Librería RxTx:

```
mvn install:install-file -DgroupId=org.rxtx -DartifactId=rxtx -Dversion=2.2 -Dpackaging=jar -Dfile=RXTX.jar -DgeneratePom=true
```

En cualquier caso, conviene indicar a *Eclipse* donde está *Maven* y el repositorio local. Dentro de *Eclipse* en el menú superior, pulsamos sobre “Windows->Preferences”. Aparecerá la ventana de configuración de *Eclipse*. No movemos en el directorio izquierdo a “Maven->User Settings” y ahí indicamos la ruta a *Maven*.

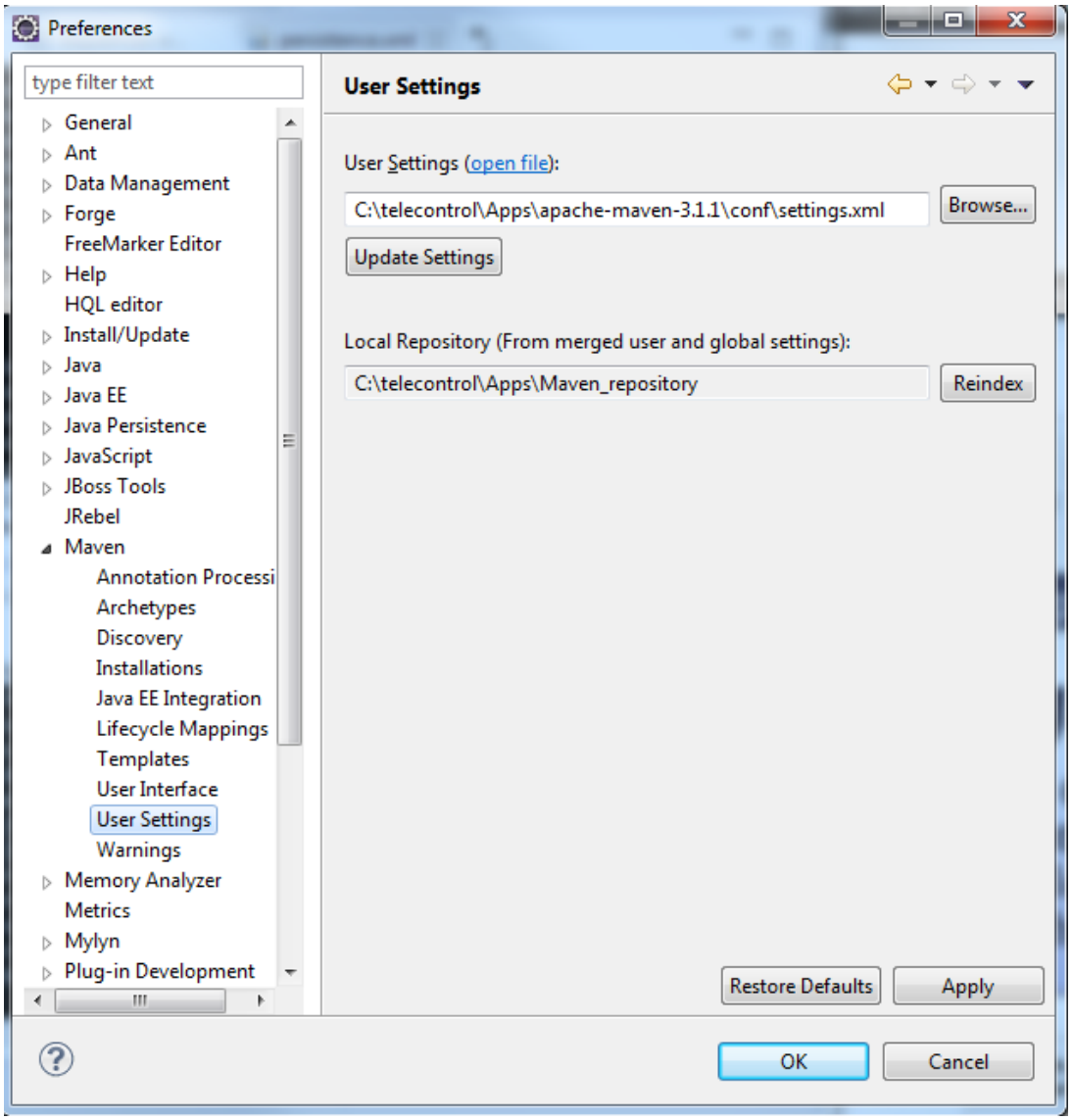

Ilustración 85: Configuración de *Maven* en *Eclipse*.

b.6. Obtener el código fuente

El código fuente del proyecto se encuentra en la carpeta “/telecontrol/workspace” del DVD. Además la carpeta del código fuente es un repositorio *GIT* desde donde podrás consultar versiones anteriores del desarrollo de este proyecto. Se recomienda encarecidamente utilizar además un repositorio remoto donde guardar tus nuevas modificaciones.

Para hacer que eclipse reconozca y cargue el proyecto, simplemente selecciona esta carpeta (/telecontrol/workspace) como la ruta del *workspace* de *Eclipse*.

b.7. Instalar controlador serie en *Java JDK*

Para poder utilizar el módem *GSM* es necesario añadir el controlador *RxTx* en tu instalación *Java*. Por favor busca la instalación de *JDK* en tu sistema y copia los siguientes archivos del driver comprimido (desde “/telecontrol/Libs/”).

- En *Ubuntu*:

El *JDK* de java se suele encontrar en “/usr/lib/jvm/java-7-openjdk-(+architectura)”. Los ficheros de la librería se encuentran en: “/mfz-rxtx-2.2-20081207-linux-(+arquitectura)/”.

- Copia en la carpeta”/bin/” de tu *JDK* el fichero “/RXTXcomm.jar”
- Copia en la carpeta “/lib/” de tu *JDK* la librería “librxtxSerial.so”

- En *Windows*

El *JDK* de java se suele encontrar en “C:\Program Files\Java\jdk1.7.0_(+revisión)”. Los ficheros de la librería se encuentran en: “/mfz-rxtx-2.2-20081207-windows-(+arquitectura)/”.

- Copia en la carpeta “/bin/” de tu *JDK* el fichero “/RXTXcomm.jar”
- Copia en la carpeta “/lib/” de tu *JDK* la librería “rxtxSerial.dll”

Anexo C. Casos de uso completos.

Caso de uso:	**Consultar historico planta**	**ID:**	**1**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Accede a toda la información almacenada referente a la instalación. Esta se debe poder mostrar tanto a nivel de tablas, como a nivel de gráficas..
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El Usuario debe de estar autentificado en el sistema y debe tener permiso de acceso.
Poscondición:
1. El estado de la planta no debe modificarse tras la consulta.
Flujo:
1. El Usuario especifica la acción 2. El Sistema accede a la información almacenada 3. El Sistema solicita al usuario los criterios de consulta 4. El Usuario especifica los criterios de consulta: - los elementos a consultar de la planta - el periodo de tiempo - como desea que se le muestre la información obtenida (tablas o gráficas) 5.El Sistema procesa la consulta y muestra al Usuario la información solicitada
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**Personalizar planta**	**ID:**	**2**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite modificar los parámetros de la instalación
Actores principales:
Administrador
Actores secundarios:
Trigger:
Administrador
Precondición:
1. El Usuario debe de estar autentificado en el sistema y debe tener permiso de acceso.
Poscondición:
1. El estado de la planta no debe modificarse tras la consulta.
Flujo:
1. El Usuario especifica la acción 2. El Sistema accede a la información almacenada 3. El Usuario modifica aquellos parámetros o elementos que quiere personalizar 4. El Sistema muestra al Usuario como quedará la configuración tras los cambios realizados, a la espera de que el Usuario los confirme, cancele o los modifique Los pasos 3-4 se repiten hasta que el Usuario acepte o cancele el proceso de personalización. 5. El Sistema guarda los cambios especificados por el Usuario
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**listar elemento (parametrización)**	**ID:**	**3**
Creado por:	Ángel Merino Sastre	**Fecha:**	20/04/2014
Modif. por:		**Fecha Mod:**	

Descripción:
Permite ver una lista de elementos añadidos al sistema
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
Poscondición:
Flujo:
1. El Usuario selecciona la acción 2. El Sistema comprueba los privilegios del Usuario 3. El Sistema consulta los datos y muesta una tabla al Usuario
Flujo alternativo:
Excepción:
Incluido:
- UC_5 - UC_6 - UC_7 - UC_18 - UC_19
Requisitos especiales:
1. Todos usuarios solo pueden ser vistos por los administradores.
Notas:
- Como elementos entendemos: módulos, sensores, actuadores, usuarios, secciones, alarmas, secuencias diagramas, sms y teléfonos

Caso de uso:	**Añadir elemento (parametrización)**	**ID:**	**4**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite añadir un elemento al sistema.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El elemento a crear no debe ser igual en todos sus datos clave a otro existente en el sistema
Poscondición:
1. Una vez se haya finalizado este proceso el Usuario podrá acceder al Sistema.
Flujo:
1. El Usuario especifica los datos del elemento a añadir. 2. El Sistema comprueba que dicho elemento no exista en el Sistema. 3. El Sistema almacena el elemento y su información en el sistema.
Flujo alternativo:
Excepción:
1. El elemento a crear ya existe en el sistema. - Abortar y mostrar mensaje describiendo situación al usuario
Incluido:
Requisitos especiales:
1. Los usuarios solo pueden ser añadidos por los administradores
Notas:
- Como elementos entendemos: módulos, sensores, actuadores, usuarios, secciones, alarmas, secuencias diagramas, sms y telefonos

Caso de uso:	**Eliminar elemento (Parametrización)**	**ID:**	**5**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Elimina un elemento del sistema.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
- El elemento no debe tener ninguna dependencia con otro elemento en el sistema. (e.j: un sensor no puede estar conectado a un módulo)
Poscondición:
1. El elemento desaparece del sistema pero no sus registros
Flujo:
1. El Administrador especifica el elemento a eliminar de la lista (UC_3) 2. El Sistema comprueba la existencia de dicho elemento y que no tenga dependencias (relaciones) con otros elementos 3. El Sistema Elimina toda la información relativa elemento
Flujo alternativo:
1. El elemento tiene dependencias con otros elemento -1.1 Informar y eliminar todas las dependencias -1.2 Informar y cancelar eliminación
Excepción:
Incluido:
Requisitos especiales:
1. El usuario actual no puede eliminarse a sí mismo. 2. Los usuarios solo pueden ser eliminados por los administradores
Notas:
- Como elementos entendemos: módulos, sensores, actuadores, usuarios, secciones, alarmas, secuencias diagramas, sms y teléfonos

Caso de uso:	**Modificar elemento (Parametrización)**	**ID:**	**6**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Modifica la información de un elemento del sistema.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
Poscondición:
Flujo:
1. El Usuario especifica el elemento a modificar de la lista. (UC_3) 2. El Sistema comprueba la existencia de dicho elemento. 3. El Usuario modifica la información y confirma. 4. El Sistema aplica los cambios.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
1. Los campos clave del elemento contenidos en una dependencia deben ser actualizados. 2. Los usuarios solo pueden ser modificados por si mismos o por los administradores
Notas:
- Como elementos entendemos: módulos, sensores, actuadores, usuarios, secciones, alarmas, secuencias diagramas, sms y teléfonos

Caso de uso:	**Consultar elemento (Parametrización)**	**ID:**	**7**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Muestra información detallada de un elemento
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
Poscondición:
1. El elemeto consultado no debe ser modificado.
Flujo:
1. El usuario especifica el elemento a consultar de la lista. (US_3) 2. El Sistema comprueba la existencia de dicho elemento y consulta la información 3. El Sistema muestra toda la información relativa al elemento.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
- Como elementos entendemos: módulos, sensores, actuadores, usuarios, secciones, alarmas, secuencias, sms y teléfonos

Caso de uso:	**Arrancar monitorización**	**ID:**	**8**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Comienza a capturar datos, procesar alarmas y secuencias, arranca el módulo sms y almacena logs.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El Usuario debe de estar autentificado en el sistema y debe tener permiso de acceso. 2. La planta debe estar parada. 3. Debe existir al menos un dispositivo desde donde capturar datos.
Poscondición:
Flujo:
1. El Usuario especifica la acción 2. El Sistema consulta si la captura de datos esta inactiva 3. El Sistema comprueba los dispositivos de captura 4. El sistema arranca la captura de datos. 5. El sistema marca el sistema como activo y muestra los datos al usuario.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
- Renombrada desde "Arrancar captura históricos planta"

Caso de uso:	**Detener monitorización**	**ID:**	**9**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Detienes los proceso de captura y almacenamiento de dato así como el subsistema de SMS
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El Usuario debe de estar autentificado en el sistema y debe tener permiso de acceso.
2. La planta debe estar activa
Poscondición:
Flujo:
1. El Usuario especifica la acción 2. El Sistema consulta si la captura de datos esta activa 3. El Sistema detiene de manera segura los procesos de captura 4. El sistema muestra al usuario el resultado de la detención
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
- Renombrada desde "Detener captura históricos planta"

Caso de uso:	**log in**	**ID:**	**10**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Autentificarse y registrarse en el sistema
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El Usuario no se encuentra activo.
Poscondición:
1. El usuario se encuentra activo en el sistema y se le permite el acceso
Flujo:
1. El Usuario intenta acceder al sistema. 2. El Sistema comprueba si es usuario está activo. 3. El usuario introduce sus datos de acceso. 4. El sistema comprueba los datos de acceso introducidos. 5. El usuario para a modo activo en el sistema
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**log out**	**ID:**	**11**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Salir del sistema
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El Usuario se encuentra activo.
Poscondición:
1. El usuario pasa a ser inactivo en el sistema.
Flujo:
1. El Usuario indica al sistema salir. 2. El Sistema realiza las acciones necesarios para pasar el usuario a inactivo. 3. El usuario sale del sistema.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**Consultar históricos**	**ID:**	**12**
Creado por:	Adrian Peñate Sánchez	**Fecha:**	29/10/2007
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
El sistema ofrece al usuario los elementos a sobre los cuales se puede consultar datos registrados.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. Deben existir registros de algún elemento del sistema
Poscondición:
Flujo:
1. El Usuario indica la acción. 2. El Sistema busca todos aquellos elementos de los cuales se puedan mostrar registros. 3. El Sistema lista tales elementos al usuario en tablas
Flujo alternativo:
Excepción:
Incluido:
- CU_13
Requisitos especiales:
Notas:

Caso de uso:	**Visualizar históricos**	**ID:**	**13**
Creado por:	Adrian Peñate Sánchez	**Fecha:**	29/10/2007
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
El sistema ofrece al usuario los elementos a sobre los cuales se puede consultar datos registrados.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. Deben existir registros de algún elemento del sistema
Poscondición:
Flujo:
1. El Usuario indica la acción. 2. El Sistema busca todos aquellos elementos de los cuales se puedan mostrar registros. 3. El Sistema lista tales elementos al usuario en tablas
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**Conectar dispositivo a módulo (parametrización)**	**ID:**	**14**
Creado por:	Adrian Peñate Sánchez	**Fecha:**	29/10/2007
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Un dispositivo queda asociado a un módulo y a una entrada o salida concreta del mismo.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. Deben existir dispositivos y módulos en el sistema 2. El módulo seleccionado debe tener entradas o salidas libres.
Poscondición:
1. El dispositivo queda asociado a una y solo una salida o entrada. 2. La entrada o salida del módulo queda como no disponible.
Flujo:
1. El Usuario selecciona un elemento de las listas sensores o actuadores y de los módulos disponibles (UC_3). 2. El Usuario selecciona una entrada en el módulo. 3. El usuario añade la información adicional y solicita la información. 4. El sistema comprueba los datos y realiza la asociación.
Flujo alternativo:
Excepción:
Incluido:
- CU_3
Requisitos especiales:
Notas:
1. Los dispositivos son sensores o actuadores. 2. Los sensores sólo pueden ser asociados con entradas del módulo y los actuadores con salidas

Caso de uso:	**Listar pasos (Parametrización)**	**ID:**	**15**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Un dispositivo queda asociado a un módulo y a una entrada o salida concreta del mismo.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. Deben existir dispositivos y módulos en el sistema 2. El módulo seleccionado debe tener entradas o salidas libres.
Poscondición:
1. El dispositivo queda asociado a una y solo una salida o entrada. 2. La entrada o salida del módulo queda como no disponible.
Flujo:
1. El Usuario selecciona un elemento de las listas sensores o actuadores y de los módulos disponibles (UC_3). 2. El Usuario selecciona una entrada en el módulo. 3. El usuario añade la información adicional y solicita la información. 4. El sistema comprueba los datos y realiza la asociación.
Flujo alternativo:
Excepción:
Incluido:
- CU_3
Requisitos especiales:
Notas:
1. Los dispositivos son sensores o actuadores. 2.Los sensores sólo pueden ser asociados con entradas del módulo y los actuadores con salidas

Caso de uso:	**Añadir paso (Parametrización)**	**ID:**	**16**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Añade un paso una secuencia
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El caso de uso ocurre cuando añadimos, modificamos una secuencia. Usa UC_4, UC_6. 2. La secuencia debe ser suministrada por el usuario.
Poscondición:
1. El paso queda asociado a la secuencia de manera temporal hasta que la secuencia se guarde. 2. Si el proceso se añadir o modificar secuencia se cancela, el paso se elimina.
Flujo:
1. El Usuario añade o modifica una secuencia. 2. El sistema muestra el formulario con los distintos campos y elementos del paso. 3. El usuario suministra y envía la información del paso. 4. El sistema añade temporalmente el paso a la secuencia hasta que esta se guarda.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. Los pasos son: de espera temporal, espera condicional, espera temporal, espera condicional, actuación, salto, salto condicional, arranque/detención de secuencia.

Caso de uso:	**Eliminar paso (Parametrización)**	**ID:**	**17**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Elimina un paso de una secuencia.
Actores principales:
Usuario
Actores secundarios:
Trigger:
Usuario
Precondición:
1. El caso de uso ocurre cuando añadimos, modificamos una secuencia. Usa UC_4, UC_6. 2. La secuencia debe ser suministrada por el usuario.
Poscondición:
1. El paso queda eliminado de la secuencia de manera temporal hasta que la secuencia se guarde. 2. Si el proceso se añadir o modificar secuencia se cancela, el paso se elimina.
Flujo:
1. El usuario añade o modifica una secuencia. 2. El sistema muestra la lista de pasos asociados a esa secuencia. 3. El usuario selecciona el paso que desea eliminar. 4. El sistema elimina el paso de manera temporal.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. Los pasos son: de espera temporal, espera condicional, espera temporal, espera condicional, actuación, salto, salto condicional, arranque/detención de secuencia.

Caso de uso:	**Ejecutar secuencia**	**ID:**	**18**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Ejecuta una secuencia de pasos.
Actores principales:
Usuario, sistema.
Actores secundarios:
Trigger:
Usuario, sistema por una alarma.
Precondición:
1. La secuencia no debe estar ejecutándose.
Poscondición:
1. La secuencia queda marcada como en ejecución hasta que finalice sus pasos.
Flujo:
1. El usuario elige una secuencia de la lista. (CU_3) 2. El sistema lanza la ejecución de los pasos y marca la secuencia como activa.
Flujo alternativo:
1. La secuencia es activada por la activación o desactivación de una alarma.
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**Detener secuencia**	**ID:**	**19**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Detiene una secuencia de pasos.
Actores principales:
Usuario, sistema.
Actores secundarios:
Trigger:
Usuario, sistema.
Precondición:
1. La secuencia debe estar en ejecución.
Poscondición:
1. La secuencia queda marcada como en detenida.
Flujo:
1. El usuario elige una secuencia de la lista de secuencias activas. (CU_3) 2. El sistema lanza la ejecución de los pasos y marca la secuencia como activa.
Flujo alternativo:
1. La secuencia es desactivada por la activación o desactivación de una alarma.
Excepción:
Incluido:
Requisitos especiales:
Notas:

Caso de uso:	**Listar sensores SMS**	**ID:**	**20**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite a los usuarios del sistema recibir información sobre los sensores vía SMS
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado o el teléfono dado de alta en el sistema.
Poscondición:
Flujo:
1. El usuario manda un SMS autentificándose y pidiendo un listado de los sensores. 2. El sistema comprueba la validez de los datos. 3. Se obtiene la lista de todos los sensores activos. 4. Se manda la información al usuario mediante SMS.
Flujo alternativo:
1. La secuencia es desactivada por la activación o desactivación de una alarma.
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. En caso de que el teléfono desde el que se manda el SMS esté dado de alta en la base de datos, la autentificación es innecesaria.

Caso de uso:	**Listar elemento SMS (Parametrización)**	**ID:**	**21**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite a los usuarios del sistema recibir información sobre los elementos del sistema vía SMS
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado o el teléfono dado de alta en el sistema.
Poscondición:
Flujo:
1. El usuario manda un SMS identificándose y pidiendo un listado de los elementos seleccionados. 2. El sistema comprueba la validez de los datos. 3. Se obtiene la lista de todos los elementos. 4. Se manda la información al usuario mediante SMS.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. Los elementos se refieren a: sensores, actuadores, secuencias y alarmas activas. 2. En caso de que el teléfono desde el que se manda el SMS esté dado de alta en la base de datos, la autentificación es innecesaria.

Caso de uso:	**Modificar el valor de un actuador SMS**	**ID:**	**22**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite a los usuarios del sistema modificar el valor de salida de un actuador del sistema vía SMS
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado o el teléfono dado de alta en el sistema.
Poscondición:
Flujo:
1. El usuario manda un SMS identificándose e indicando una orden para un actuador. 2. El sistema comprueba la validez de los datos. 3. manda un SMS al usuario informando del estado de la planta en estos momentos. 4. El usuario responde al SMS verificando la operación. 5. Se varía el valor del actuador en cuestión.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. Los elementos se refieren a: sensores, actuadores, secuencias y alarmas activas. 2. En caso de que el teléfono desde el que se manda el SMS esté dado de alta en la base de datos, la autentificación es innecesaria.

Caso de uso:	**Ejecutar secuencias acciones SMS**	**ID:**	**23**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite a los usuarios del sistema ejecutar secuencias de acciones vía SMS.
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado o el teléfono dado de alta en el sistema.
Poscondición:
Flujo:
1. El usuario manda un SMS identificándose y especificando qué secuencia de acciones es la que quiere ejecutar. 1. El sistema comprueba que la secuencia de acciones especificada existe y no esté actualmente ejecutándose. 3. Se manda un SMS al usuario informando del estado de la planta en estos momentos. 4. El usuario responde al SMS verificando la operación. 5. Se empieza a ejecutar la secuencia de acciones en el sistema.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. En caso de que el teléfono desde el que se manda el SMS esté dado de alta en la base de datos, la autentificación es innecesaria.

Caso de uso:	**Detener secuencias acciones SMS**	**ID:**	**24**
Creado por:	Andrés del Pino Bolaños	**Fecha:**	17/06/2009
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite a los usuarios del sistema detener secuencias de acciones activas vía SMS.
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado o el teléfono dado de alta en el sistema. 2. La secuencia debe estar activa.
Poscondición:
Flujo:
1. El usuario manda un SMS autentificándose y especificando que secuencia de acciones es la que quiere detener. 1. El sistema comprueba que la secuencia de acciones especificada existe y esté actualmente ejecutándose. 3. Se manda un SMS al usuario informando del estado de la planta en estos momentos. 4. El usuario responde al SMS verificando la operación. 5. Se empieza a ejecutar la secuencia de acciones en el sistema.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:
1. En caso de que el teléfono desde el que se manda el SMS esté dado de alta en la base de datos, la autentificación es innecesaria.

Caso de uso:	**Mostrar planta**	**ID:**	**25**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Muestra la información de la planta
Actores principales:
Usuario.
Actores secundarios:
Trigger:
Usuario.
Precondición:
1. El usuario debe estar identificado.
Poscondición:
Flujo:
1. El Usuario especifica la acción 2. El sistema muestra la información de la planta en forma de formulario.
Flujo alternativo:
Excepción:
Incluido:
UC_26
Requisitos especiales:
Notas:

Caso de uso:	**Modificar planta**	**ID:**	**26**
Creado por:	Ignacio Solinis Camalich	**Fecha:**	28/04/2006
Modif. por:	Ángel Merino Sastre	**Fecha Mod:**	20/04/2014

Descripción:
Permite modificar la información y configuración de la planta de la planta
Actores principales:
Administrador
Actores secundarios:
Trigger:
Administrador
Precondición:
1. El usuario debe estar identificado y ser administrador.
Poscondición:
Flujo:
1. El administrador accede a la modificación dentro de “mostrar planta” UC_25. 2. El sistema muestra la un formulario con la información de la planta. 3. El administrador hace los cambios considerados y confirma. 4. El sistema valida los parámetros dados. 5. El sistema guarda y aplica la nueva configuración.
Flujo alternativo:
Excepción:
Incluido:
Requisitos especiales:
Notas:

7. Bibliografía

[1] Ignacio Solinis Camalich. Proyecto de fin de carrera: “Sistema de telecontrol de plantas desaladoras”. Facultad de Informática. ULPGC. 2006.

[2] Adrián Peñate Sánchez. Proyecto de fin de carrera: “Un sistema de monitorización y telecontrol para instalaciones del Servicio de Alojamiento Universitario de la ULPGC”. Facultad de Informática. ULPGC. 2009.

[3] Andrés del Pino Bolaños. Proyecto de fin de carrera: “Herramienta de gestión para planta de tratamiento de aguas y sistemas de energía eléctrica en paneles solares fotovoltaicos”. Facultad de Informática. ULPGC. 2010.

[4] http://struts.apache.org/

[5] Martin Fowler. “Refactoring. Improving the Design of Existing Code”. Addison-Wesley. ISBN 0-201-48567-2. 2002.

[6] http://en.wikipedia.org/wiki/Code_refactoring

[7] http://en.wikipedia.org/wiki/Code_smell

[8] http://es.wikipedia.org/wiki/Desarrollo_iterativo_e_incremental

[9] http://es.wikipedia.org/wiki/Maven

[10] Jason van Zyl, Brian Fox, John Casey, Bruce Snyder, Tim O’Brien, Eric Redmond. “Maven: The Definitive Guide”. O’Reilly. ISBN 978-0-596-51733-5. 2008.

[11] http://docs.oracle.com/javaee/

[12] http://es.wikipedia.org/wiki/Java_EE

[13] http://parasitovirtual.wordpress.com/2010/12/13/introduccion-a-la-plataforma-java-ee/

[14] http://es.wikipedia.org/wiki/JBoss

[15] http://infinispan.org/

[16] http://en.wikipedia.org/wiki/HornetQ

[17] http://es.wikipedia.org/wiki/Jakarta_Tomcat

[18] Andrew Lee Rubinger, Bill Burke. “Enterprise JavaBeans 3.1”. O’Reilly. ISBN 978-0596158026. 2010.

[19] Christian Bauer, Gabi King. “Java Persistence with Hibernate”. O’Reilly. ISBN 978-1932394887. 2010.

[20] Richard Monson-Haefel, David A. Chappell. “Java Message Service”. O’Reilly. ISBN 978-0596522049. 2001.

[21] Robert Cecil Martin. “UML for Java programmers”. Prentice-Hall. ISBN 007-6092024644. 2002.

[22] Jim Arlow, Lla Neustadt. “UML2”. Anaya Multimedia.ISBN 978-8441520332. 2006.

[23] http://smslib.org/doc/

[24] Willian F. Opdyke, Ph.D Thesis: “Refactoring Object-Oriented Frameworks”, 1993. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.17.688