Universidad Politécnica de Valencia

Escuela Técnica Superior de Ingeniería Informática

Ingeniería en Informática

Proyecto Fin de Carrera

Sistema móvil personal de ayuda para el mantenimiento de vehículos y equipos del Consorcio Provincial de Bomberos de Valencia.

Consorcio Provincial de Bomberos de Valencia

Autor: **Javier García San Juan.**
Codirector ETSINF: **Juan Sánchez Díaz**
Codirector CPBV: **Manuel D. Serrat Olmos**

Enero 2012

Agradecimientos

La realización del proyecto final de carrera conlleva el término de una etapa en la que merece la pena echar la vista atrás y recordar todo lo pasado, tanto por el fruto de meses de desarrollo como por la conclusión de una licenciatura a través de años de duro trabajo. Por esto, quiero agradecer con estas palabras a todos aquellos que me han ayudado en este camino.

En primer lugar, reconocer a mis padres el gran esfuerzo realizado en ayudarme de forma moral y económica para mi formación como ser humano y como profesional. Gracias por dedicar gran parte de vuestra vida en guiarme en el camino. A David, mi hermano pequeño, que siempre me ha ayudado con su buen humor en las situaciones difíciles y en general a toda mi familia, que siempre han estado ahí cuando la he necesitado.

También gracias a mi tutores Juan y Manuel por su orientación y sus consejos durante todo el proyecto. Su disponibilidad y el buen trato tenido siempre conmigo han sido de gran valor para mí.

Gratificar a todos los compañeros que he podido cruzarme en mi carrera y que han sabido enseñarme valores como la amistad, compañerismo, trabajo en grupo y solidaridad.

Muchas gracias a todos mis amigos por todos aquellos buenos momentos vividos y los que nos quedan por vivir.

Y por supuesto a Virginia, mi novia, por haber estado ahí todos estos años, tanto en los buenos como en los malos momentos, apoyándome en cada momento de mi vida.

Resumen

El objetivo principal del proyecto es construir un sistema informático que permita al personal del Consorcio Provincial de Bomberos de Valencia disponer de la información y de las guías operativas necesarias para poder realizar el mantenimiento de los vehículos y equipos de cada parque de Bomberos.

El sistema que se desarrollará, formado por varias aplicaciones, debe poder ser ejecutado en su parte clientes sobre el sistema operativo Android en un tablet o en un teléfono inteligente. Los clientes accederán utilizando servicios Web.

Para ello, no sólo habrá que construir las aplicaciones necesarias sino que será necesario definir toda la plataforma hardware de soporte a estas aplicaciones, de manera que el Consorcio pueda planificar adecuadamente sus inversiones alineando este proyecto con otros proyectos tecnológicos previstos. Asimismo, también se definirá la infraestructura software sobre la que se ejecutarán los servicios web a los que las aplicaciones móviles accederán para la materialización de sus funcionalidades y las posibles necesidades de interoperabilidad de esta aplicación con otras existentes.

Concretamente la parte cliente del sistema debe ser capaz de realizar los siguientes escenarios en el sistema:

- Gestión de equipos y vehículos.
- Filtrado de consulta de vehículos y equipos para cada parque.
- Listado de revisiones de mantenimiento de cada equipo y vehículo.
- Consulta del historial de revisiones de cada activo.
- Consulta de las guías operativas para el mantenimiento.

Por último la parte servidora del sistema (REST) deben programarse los siguientes escenarios:

- Consulta a la base de datos existentes de personal, equipos y vehículos.
- Autorizados para la realización de un mantenimiento.
- Revisiones realizadas.

Palabras Clave

Android, servicio web, Rest, aplicación móvil, sistema mantenimiento

Índice General

Índice Figuras

1 Introducción

El siguiente documento tiene como objetivo detallar el proyecto final realizado para el termino de la Ingeniería Informática, que consiste en el desarrollo de una sistema móvil para dispositivos Android para el mantenimiento de vehículos y sistemas en el Consorcio Provincial de Bomberos de Valencia.

El sistema de mantenimiento consiste en una aplicación para una tablet o móvil inteligente dotado con el sistema operativo Android y que tendrá como misión realizar el mantenimiento en los parques de Bomberos, esta aplicación se comunicará a través de servicios web REST (Representational State Transfer) y descargará aquellas tareas que se deban realizar en el parque seleccionado. Esto supondrá diversas mejoras tanto en seguridad como en automatización de procesos para el Consorcio, así como para todas aquellas personas que requieran de sus servicios.

En primer lugar, el personal operativo tendrá un proceso informatizado que gestionará y guiará en el proceso de mantenimiento facilitando la realización de las tareas.

En segundo lugar, el personal operativo puede planificar la realización de las tareas mediante una programación de esta o con el vencimiento en una fecha concreta.

Y finalmente, cualquier usuario del Consorcio Provincial puede consultar tanto las ordenes de trabajo pendientes, como un historial de las revisiones realizadas para un activo en concreto en su parque, gracias a que estos estarán conectados a través de una red empresarial y podrán comunicarse mediante los servicios web.

En lo referente al desarrollo de la memoria y del proyecto se utilizará tecnología orientada a objetos, las fases de análisis y diseño se llevarán a cabo empleando una adaptación de RUP (*Rational Unified Process*) y con la notación del lenguaje unificado de modelado para el modelo de requisitos, diagrama de clases y escenarios.

1.1 Motivación

En primer lugar remarcar que este proyecto ha sido realizado en colaboración con el Consorcio Provincial de Bomberos de Valencia y ha servido para acercarme al mundo laboral aportando mi conocimiento en estos años universitarios, además de ayudar con la gestión del mantenimiento de una "empresa" donde la seguridad en todos los casos es crucial y puede marcar la diferencia en cada salida de emergencia que se pueda producir.

Por otro lado, este proyecto aporta la sistematización e informatización de la gestión de un mantenimiento que anteriormente se hacía sin ningún tipo de programación y de forma independiente en cada parque de Bomberos.

Al mismo tiempo, este sistema de ayuda facilitará el trabajo de mantenimiento, debido a que el personal operativo podrá realizar la gestión de un vehículo o un equipo desde el lugar donde se encuentre este, gracias a que esta aplicación a sido desarrollada para un dispositivo móvil, ya sea tablet o móvil inteligente.

En el Android Market se puede observar que hay gran cantidad de aplicaciones para el mantenimiento de vehículos, sobretodo de coches, pero la aplicación desarrollada no solo va mas allá, manteniendo cualquier tipo de activo del Consorcio Provincial de Bomberos, sino que se le da una vuelta de tuerca pudiendo programarse la gestión de dicho mantenimiento tanto para una fecha concreta como para una tarea determinada, por ejemplo, (cambiar el aceite cada 5000 km del BUP de un parque de bomberos).

De esta manera se ha desarrollado una aplicación especifica para la gestión de mantenimiento de vehículos y equipos para el Consorcio Provincial de Bomberos de Valencia, pero que con unos pequeños cambios podría servir para ayudar en el mantenimiento de cualquier empresa.

1.2 Propósito

El principal objetivo del proyecto es proporcionar un nuevo sistema de acceso al personal operativo del Consorcio Provincial de Bomberos de Valencia para que puedan realizar el mantenimiento de forma mas sencilla y pudiendo obtener los pasos o guías operativas para poder desenvolverse en dichas tareas con la única ayuda de una tablet o teléfono inteligente.

Esta aplicación logra que cualquier miembro del personal operativo pueda interactuar con los datos del vehículo o equipo y su orden de mantenimiento de una forma rápida, sencilla desde el lugar donde se encuentra dicho activo. De esta manera podemos decir que hay tres objetivos bien diferenciados:

- Acceso eficiente: Presentar los datos de forma ordenada y optimizada, además de ser lo mas interactivo posible para facilitar la creación y edición del mantenimiento de los activos.
- Interacción móvil: Posibilidad de realizar el mantenimiento desde el lugar donde se encuentra el vehículo o el equipo, facilitando así su gestión.
- FeedBack: Cuando haya finalizado la gestión del mantenimiento y su orden de trabajo se cierre, se podrá volver a consultar mediante un histórico de revisiones, donde se visualiza todas las operaciones de mantenimiento que se han realizado para el activo seleccionado.

La finalidad personal es el poder aportar mi granito de arena en el lugar de trabajo donde he realizado las prácticas para complementar mis años universitarios y poder ayudar tecnológicamente al personal que se encargue del mantenimiento de cada parque.

Además, el poder continuar mis estudios en Android y profundizar en la tecnología móvil, dado que Android es un sistema operativo donde comencé a estudiar en una asignatura de quinto “ADM”, y que me guío para la realización del proyecto final.

1.3 Estructura del proyecto

La memoria esta organizada principalmente en la forma en la que se ha realizado el proyecto, es decir siguiendo un proceso (RUP) el cual permite que sea un proceso de desarrollo fundamentalmente iterativo, además de gestionar una administración de requisitos necesarios para la realización de este.

El proceso RUP esta centrado en la arquitectura y guiado por los casos de uso. Incluye artefactos (productos tangibles del proceso), que iremos presentando en la memoria según el proceso en el que nos encontremos en cada momento.

Dicho esto, podemos dividir la memoria en 4 capítulos diferenciados por este proceso:

- Fase de inicio: En este capitulo se define y acota el alcance del proyecto con el Consorcio Provincial de Bomberos de Valencia, además se identifica los riesgos asociados al proyecto y se propone una visión muy general de la arquitectura software, es decir, del sistema operativo Android, de la implementación del servicio web REST y la gestión de la base de datos MySQL y PostGress.

- Fase de elaboración: En este capitulo se orienta al desarrollo de la arquitectura, abarcando mas los flujos de trabajos de requisitos, análisis y diseño.

- Fase de construcción: En este capitulo se describe la construcción de la aplicación y del servicio web. Se seleccionan los casos de uso, se refina su diseño y análisis y se procede a su implementación y pruebas.

- Fase de transición: En este capitulo se asegura que el software este disponible para los usuarios finales, se acaban de ajustar errores encontrados en las pruebas de aceptación y se provee del soporte técnico necesario para el uso de la aplicación.

2 Fase de Inicio

En este capitulo se establece el ámbito del proyecto y sus límites, además de poner en manifiesto los casos de uso del sistema, y la muestra general de la arquitectura utilizada para su implementación.

También se indica unas pequeñas estimaciones de costes y tiempos del proyecto que intentaremos seguir durante toda la implementación, y por último se estimaran los posibles riesgos que se pueden dar en el proyecto.

Los productos o "artefactos" que incluiremos en esta fase de inicio del proceso de desarrollo RUP será el modelo de casos de uso, además de la visión del negocio, la cual describe los objetivos y restricciones a alto nivel.

2.1 El proyecto

En este apartado y los siguientes subapartados se describen cuestiones generales acerca del proyecto en curso. En primer lugar, se detalla el contexto del mismo, después se definen los requisitos y por último se marcan las decisiones tomadas en base a las necesidades.

2.1.1 Visión y Análisis del Negocio

El cliente es el Consorcio Provincial de Bomberos de Valencia, es un organismo público cuya misión es la prevención y extinción de incendios así como las tareas de salvamento en la provincia de Valencia. El Consorcio se ha consolidado como uno de los servicios más importantes de España, con una cobertura territorial de 10.671,48 kilómetros cuadrados, 5.819 kilómetros cuadrados de masa forestal, más de 3.000 kilómetros de carreteras, 1.700.829 vehículos y cerca de 184.483 empresas contabilizadas en la provincia de Valencia.

Los 265 pueblos y ciudades de la provincia suman, sin contar la capital, una población de 1.761.154 habitantes (INE mayo 2010), que prácticamente se duplican al llegar la época estival.

Actualmente, y con todos estos datos entre manos, puedo decir que la seguridad y la prevención de accidentes es clave en un organismo como es el Consorcio, por lo que el mantenimiento de todos los vehículos y equipos que se utilizan para los servicios es vital.

Por otro lado, el Consorcio no dispone de un sistema de mantenimiento informatizado y automático para la realización de las tareas de mantenimiento de vehículos y equipos de cada parque. En este caso cada parque, realiza dichas tareas gestionadas por ellos mismo, sin ningún recordatorio que les avise y sin ningún gestión de incidentes en el mantenimiento a través de la central.

Por todo esto, y gracias a la futura adquisición de hardware que facilitará la gestión del mantenimiento mediante dispositivos móviles y que podrán ayudar al personal operativo a realizar las tareas desde donde se localice el activo en cuestión, se decidió como finalización de las practicas de empresa, la realización del proyecto final que suprimirá dicha carencia.

El objetivo del proyecto es conseguir que abarque la gestión de los activos y las tareas de mantenimiento a realizar por los parques de Bomberos en una aplicación Android por el personal operativo del Consorcio, apoyada por un servicio web REST, para la adquisición de los datos almacenados en las bases de datos de la organización.

El servicio web REST también debe ser diseñado desde cero, y que se basará en la escritura y lectura de las bases de datos necesarias para la implementación de este servicio de mantenimiento.

Finalmente, esta aplicación también dispondrá de un historial de todas las revisiones realizadas a través del tiempo, dado que se considero importante que se conozca todo el mantenimiento que haya podido recibir un activo durante su vida útil tanto para sus futuras revisiones como para la gestión de nuevas ordenes de trabajo sobre ese activo.

2.1.2 Modelado de casos de uso

En este diagrama se define una notación gráfica para representar los casos de uso, es decir, el comportamiento del sistema al afrontar una tarea de negocio o un requisito del negocio. Para realizar cualquiera de ellas debe haberse autenticado mediante IMAP (usuario y contraseña del correo corporativo) previamente.

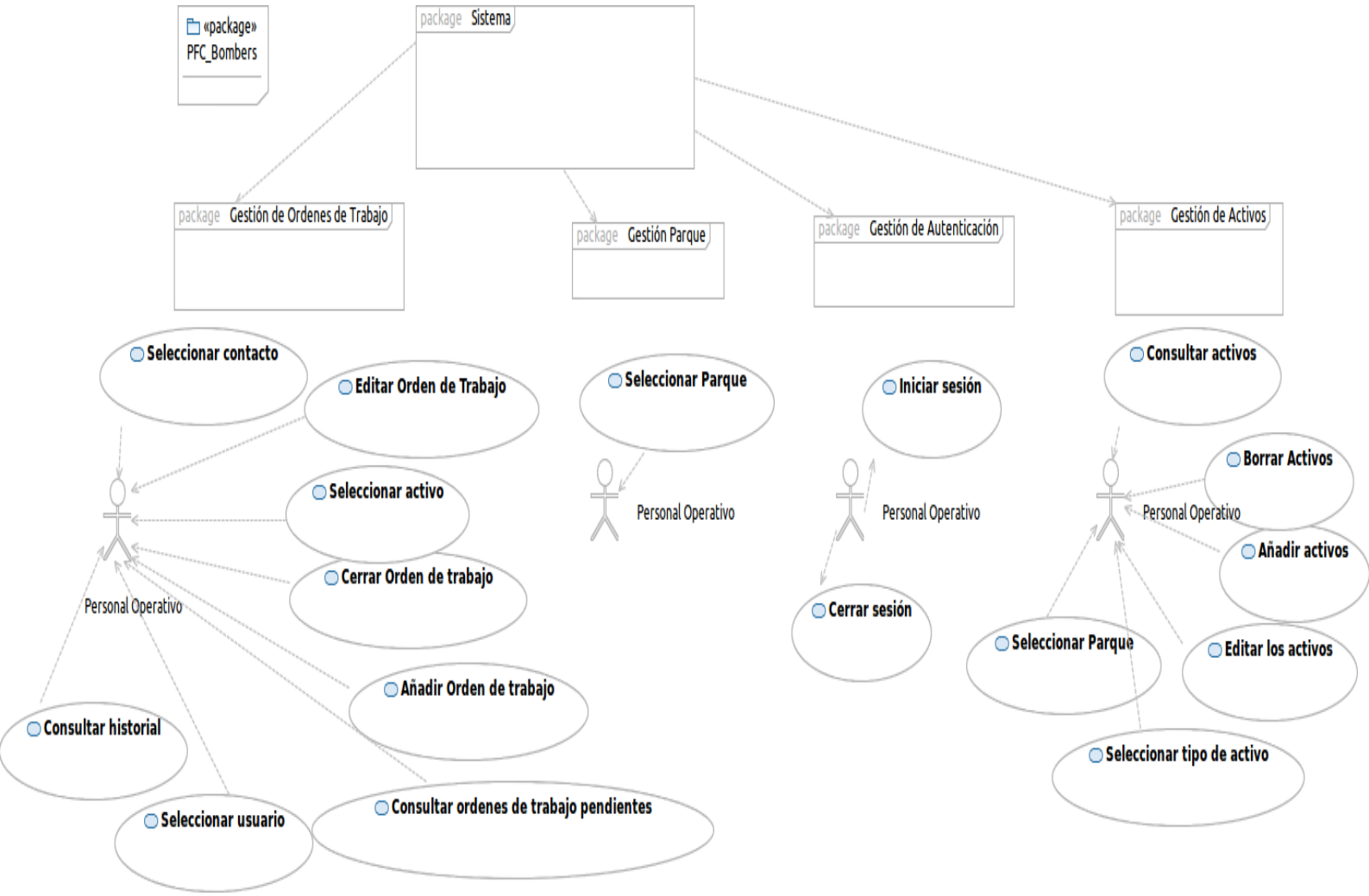

Figura 1: Diagrama de casos de uso

2.1.3 Especificaciones de requisitos

El Consorcio no disponía de ningún sistema de mantenimiento definido, con lo cual se debe definir tanto las decisiones de la interfaz gráfica de la aplicación como los requisitos funcionales. La finalidad es que la aplicación sea lo mas sencilla de utilizar posible para el personal operativo y que el salto tecnológico a los dispositivos móviles no sea un inconveniente sino una ventaja. Los principales requisitos establecidos para realizar la aplicación son:

- Autenticación: El usuario deberá autenticarse con los datos del correo corporativo, mediante un usuario y contraseña.

- Interfaz: Clara y sencilla para el usuario corporativo. Con el aspecto similar a una aplicación Android.

- Seguridad: Tan solo un usuario con correo corporativo puede acceder a la aplicación.

- Ampliación: Código adaptable a futuras modificaciones o ampliaciones.

- Interacción con el dispositivo: Dado que esta diseñado para un dispositivo móvil, la aplicación debe poder interactuar con sus posibilidad, por ejemplo, poder enviar un correo a la central cuando una tarea haya finalizado.

- Gestión de errores: Tratar todos lo errores posibles con diálogos para que el usuario pueda interactuar sin ayuda con el dispositivo móvil.

- Gestión de ordenes de trabajo: Posibilidad de finalizar una tarea como realizada o simplemente devolverla al supervisor porque no corresponde en ese instante.

- Filtrado: Mostrar solo las tareas que correspondan al parque en cuestión.

2.1.4 Especificaciones técnicas

El servicio de mantenimiento desarrollado cuenta con tres capas básicas para poder realizar las especificaciones de requisitos descritas en el apartado anterior.

En primer lugar, tenemos almacenados los datos de la organización en bases de datos almacenada en varios servidores disponibles en el Consorcio.

Por un lado tenemos almacenadas en un servidor las ordenes de trabajo y los activos con MySQL y por otro lado obtenemos el listado de parques de otra base de datos con PostgreSQL y finalmente realizamos la autenticación IMAP contra la base de datos del servidor de correo donde tenemos el listado de personal operativo.

En segundo lugar, la aplicación cuenta con una serie de servicios web REST que han sido desarrolladas para una comunicación mas eficiente con las bases de datos descritas anteriormente y la aplicación Android.

Finalmente, la aplicación trata la información recibida por los servicios web y los muestra por pantalla para que el usuario corporativo pueda gestionar las tareas de mantenimiento.

Figura 2: Especificaciones técnicas

2.1.4.1 Autenticación IMAP

El Consorcio tiene un servidor propio de correo electrónico, lo cual permite proporcionar direcciones personalizadas a todos los empleados. Además es accesible desde Internet, por lo que se puede acceder a el desde cualquier lugar usando un cliente apropiado siempre y cuando permita los envíos con autenticación. Teniendo en cuenta esto, nuestra aplicación se autenticara contra dicho servidor, por tanto, cualquier persona operativa poseedora de un usuario y contraseña de correo podrá acceder a la aplicación, es decir, el personal operativo de la empresa.

2.1.4.2 Base de datos

2.1.4.2.1 PostgreSQL

El Consorcio utilizar este sistema gestor de datos como el corporativo , por lo que la mayoría de aplicaciones deben usarlo. Se encuentra alojado en un servidor (Firenet) y de ahí podemos obtener el listado de parques del Consorcio.

2.1.4.2.2 MySQL

El Consorcio también utiliza este sistema gestor de datos para algunas aplicaciones que no se han adaptado a PostgreSQL porque su esfuerzo no merecía la pena. Aquí será donde tendremos alojados la gestión de los activos (vehículos y equipos), además de las ordenes de trabajo.

2.1.4.3 Servicios Web REST

Debido a que la aplicación se ha diseñado para dispositivos móviles se ha decidido utilizar REST o (Representational State Transfer), que es una técnica de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web, debido a que puede describir cualquier interfaz web simple utilizando XML y HTTP, sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes como el protocolo de servicios web SOAP.

REST tiene varias ventajas que lo hacen ideal para utilizarlo por dispositivos móviles debido a que estos tienen escasos recursos:

- Un protocolo cliente/servidor sin estado: Cada mensaje comprende toda la información para comprender la petición, es decir, que ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. Por tanto, en la implementación de los servicios web se ha decidido que en cada mensaje se mande el usuario y contraseña que se ha autenticado en la aplicación Android y se compruebe mediante IMAP desde el servicio web REST.

- Un conjunto de operaciones (CRUD) bien definidas: HTTP define un conjunto de operaciones que gestionan los servicios web REST como son PUT, GET, POST, y DELETE para la persistencia de datos y con estas podremos gestionar todo el mantenimiento que se haga desde la aplicación Android.

- Una sintaxis universal: En un sistema REST, cada recurso es direccionable únicamente a través de su URI para identificar los recursos.

- El uso de hipermedios: tanto para la información de la aplicación como para las transiciones de estado son típicamente HTML o XML.

- Otro aspecto importante es que se permite no solo la transferencia de datos en formato XML, sino también en formato JSON. En nuestro caso, se ha decidido devolver los datos en este último, dado que es mucho mas compacto y eficiente usándolo de forma efectiva. Esto es debido a que es mucho mas sencillo “parsear” los datos JSON a la estructura que se quiere almacenar.

2.1.4.4 Android

Es el sistema operativo que se ha utilizado para desarrollar la aplicación en los dispositivos móviles y del cual se profundizara en capítulos posteriores. Este sistema operativo se ha utilizado debido a la futura obtención de hardware móvil con este sistema operativo y que complementará, con este proyecto final, la solución obtenida para sistematizar e informatizar el mantenimiento en los parques del Consorcio de Bomberos.

Las principales ventajas que nos aporta Android para ser utilizado para este proyecto son las siguientes:

- Diseño de dispositivo: La plataforma es adaptable a pantallas mas grandes, es decir, nos permite la flexibilidad de poder utilizarlo en teléfonos inteligentes y en tablets.

- Almacenamiento: Nos permite guardar datos, mediante su base de datos SQLite, basada en MySQL.

- Soporte de Java: Código estudiado durante la carrera universitaria y que facilita el conocimiento frente a la programación de la aplicación.

- Entorno de desarrollo: Incluye un emulador para dispositivos el cual nos permite probarlo desde el ordenador, además de herramientas para la depuración y análisis del rendimiento del software.

- Multitarea: Las aplicaciones que no estén ejecutándose en primer plano reciben también ciclos de reloj.

- Soporte para hardware adicional: Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros...

- Software libre: Android es código libre y por lo tanto existe en Internet un punto de vista orientado a compartir el código.

2.1.5 Riesgos asociados

En general para un proyecto informático los riesgos asociados son que el software nunca llegue a funcionar, que no se cumplan los plazos de entrega y que no se cumplan con las funcionalidades esperadas principalmente causado por la alta complejidad o la incertidumbre de cara al comienzo de un proyecto.

Para evitar que estos riesgos se traduzcan en problemas, trabajaremos de forma proactiva, tratando de identificar los problemas potenciales y atajándolos. En nuestro caso, el software es testeado en cada iteración realizada, según el proceso de desarrollo RUP, por tanto evitamos el riesgo de que nuestro software nunca llegue a funcionar.

Por otro lado, para evitar el incumplimiento de plazos hemos realizado un diagrama de Grantt para cumplir cada tarea en la fecha indicada previamente.

Por último, realizamos reuniones con el personal operativo, el cliente potencial del proyecto, en cada iteración para tratar de adaptar el producto lo máximo posible.

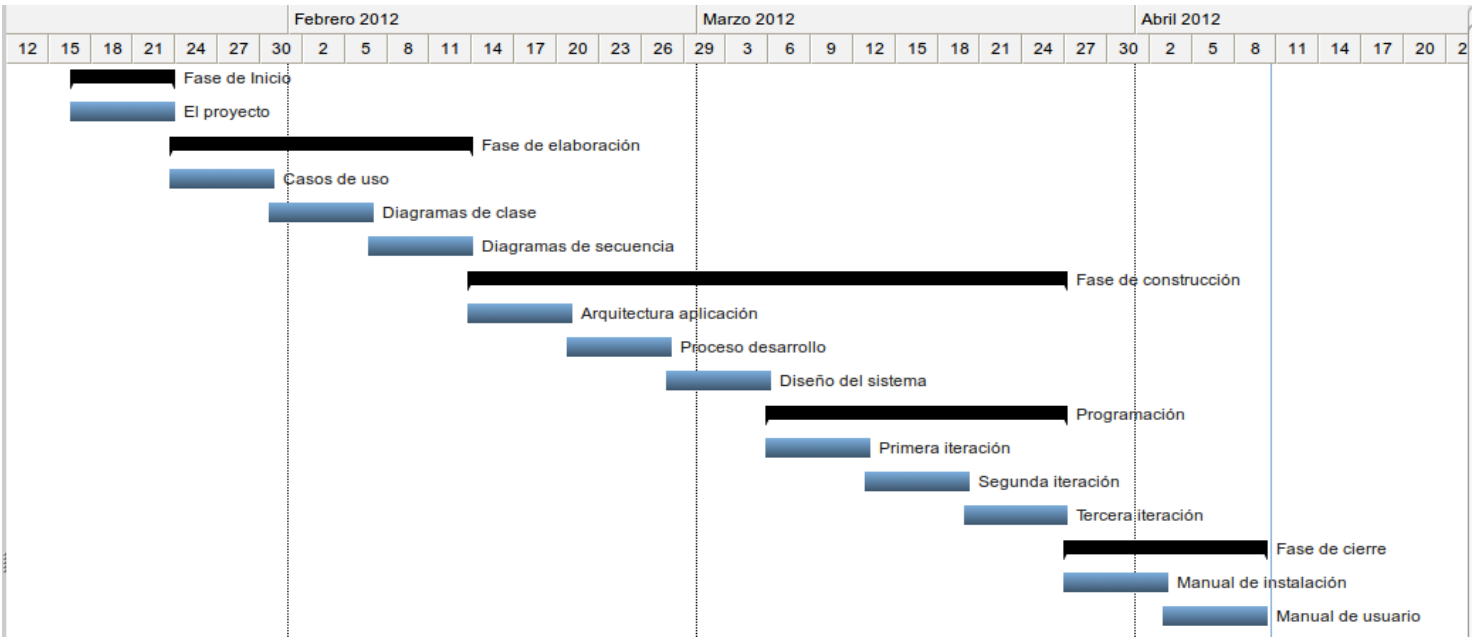

Figura 3: Diagrama de Grantt

3 Fase de elaboración

En este capitulo se analiza el dominio del problema, se establecen los cimientos de la arquitectura, se desarrolla el plan del proyecto y se eliminan los mayores riesgos.

A partir de esta fase se llega al punto de no retorno del proyecto, por tanto esta es la fase en la que se debe indicar si el proyecto es viable para su próximas fases, que serán mucho mas costosas y arriesgadas.

La fase de elaboración debe contener los casos de uso críticos identificados en la fase de inicio y demostrarse que se han evitado los riesgos mas graves.

3.1 Casos de uso

En este apartado se analiza cada uno de los casos de uso que se presentan en el diagrama de casos de uso del capitulo anterior. Con esto describiremos de forma breve cada uno de los casos del diagrama, de esta forma obtendremos una visión mas completa del proyecto.

Caso de uso: Iniciar Sesión

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: -

Postcondicones: El personal operativo queda autenticado en el sistema.

Descripción (Flujo Básico):

1) Indica su nombre de usuario y contraseña.
2) El sistema comprueba su validez comparando con los obtenidos del servidor de correo.
3) El sistema muestra la pagina principal de la aplicación.

Extensiones (Flujo Alternativo):

2.a) Indica su nombre de usuario y contraseña. El nombre de usuario o la contraseña es incorrecto.

2.a.1) El sistema notifica que el nombre de usuario o la contraseña es incorrecto y lo pide de nuevo.

Caso de uso: Cerrar Sesión

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo se ha autenticado en el sistema.

Postcondicones: El personal operativo cierra sesión en el sistema.

Descripción (Flujo Básico):

1) El usuario cierra sesión.
2) El sistema sale de la sesión y se muestra la pagina de autenticación.

Extensiones (Flujo Alternativo):

Caso de uso: Seleccionar Parque

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo se ha autenticado en el sistema.

Postcondicones: El personal operativo selecciona el parque de Bomberos.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de parque.
2) El sistema muestra el listado de parques.
3) El usuario elige el parque donde va a realizar la tarea.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de parques.

Caso de uso: Consultar Activos

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo se ha autenticado en el sistema y ha seleccionado su parque de Bomberos.

Postcondicones: El personal operativo obtiene el listado de activos en el parque.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de activos del parque seleccionado del servicio web.
2) El sistema muestra el listado de activos.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de activos.

Caso de uso: Editar Activo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado el listado de activos.

Postcondicones: El personal operativo ha editado el activos correspondiente.

Descripción (Flujo Básico):

1) El sistema muestra el listado de activos.
2) El usuario elige el activo.
3) El usuario selecciona editar el activo.
4) El sistema muestra el formulario de edición con los cuadros de texto correspondientes.
5) El usuario modifica los datos necesarios para las tareas de mantenimiento.
6) El usuario selecciona confirmar el activo.
7) El sistema valida las entradas del usuario en los cuadros de texto y otros controles.
8) El sistema edita los datos en el servicio web.
9) El servicio web inserta los datos modificados en la base de datos.

Extensiones (Flujo Alternativo):

6.a) El usuario selecciona cancelar.

6.a.1) El sistema sale de la edición de activos sin editar ningún campo.

7.a) Alguna entrada ha sido introducida incorrectamente.
7.a.1) El sistema notifica que la entrada esta vacía o no es correcta.

8.a) No hay conexión.

8.a.1) El sistema notifica que no se puede editar el activo correspondiente.

Caso de uso: Borrar Activo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado el listado de activos.

Postcondicones: El personal operativo borra el activo seleccionado.

Descripción (Flujo Básico):

1) El sistema muestra el listado de activos.
2) El usuario elige el pedido a borrar.
3) El usuario selecciona borrar el activo.
4) El sistema muestra el dialogo de confirmación de borrado.
5) El usuario confirma el borrado.
6) El sistema valida que el activo no este asignado a ninguna orden de trabajo.
7) El sistema borra el activo en el servicio web.
8) El servicio web borra los datos del activo en la base de datos.

Extensiones (Flujo Alternativo):

5.a) El usuario cancela el borrado.

5.a.1) El sistema sale del dialogo de borrado sin borrar el activo.

6.a) No hay conexión.

6.a.1) El sistema notifica que no se puede borrar el activo.

6.b) El sistema encuentra una orden de trabajo activa asignada a ese activo.

6.b.1) El sistema muestra un dialogo de error: El activo esta siendo usado.

Caso de uso: Añadir Activo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado el listado de activos.

Postcondicones: El personal operativo añade el activo.

Descripción (Flujo Básico):

1) El sistema muestra el listado de activos.
2) El usuario selecciona Menú del teclado del dispositivo.
3) El sistema muestra el Menú Añadir.
4) El usuario selecciona Añadir.
5) El usuario añade los datos necesarios para rellenar el activo.
6) El usuario selecciona confirmar.
7) El sistema valida las entradas del usuario en los cuadros de texto y otros controles.
8) El sistema añade el activo en el servicio web.
9) El servicio web inserta los datos modificados en la base de datos.

Extensiones (Flujo Alternativo):

6.a) El usuario selecciona cancelar.

6.a.1) El sistema sale del registro de activos sin añadir ningún activo.

7.a) No hay conexión.

7.a.1) El sistema notifica que no se puede añadir el activo correspondiente.

Caso de uso: Seleccionar Tipo de Activo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado editar o añadir un activo.

Postcondicones: El personal operativo asigna un activo a un tipo de activo.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de tipos de activos.
2) El sistema muestra el listado de tipos de activos.
3) El usuario elige el tipo de activo al que se va a asignar el activo seleccionado.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de tipos de activos.

Caso de uso: Consultar Orden de Trabajo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo se ha autenticado en el sistema y ha seleccionado su parque de Bomberos.

Postcondicones: El personal operativo obtiene el listado de ordenes de trabajo del parque.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de ordenes de trabajo del parque seleccionado del servicio web.
2) El sistema muestra el listado de ordenes de trabajo.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de ordenes de trabajo.

Caso de uso: Editar Orden de Trabajo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado el listado de ordenes de trabajo.

Postcondicones: El personal operativo edita la orden de trabajo correspondiente.

Descripción (Flujo Básico):

1) El sistema muestra el listado de ordenes de trabajo.
2) El usuario elige la orden de trabajo a modificar.
3) El usuario selecciona Editar.
4) El sistema muestra el formulario de edición con los cuadros de texto correspondientes.
5) El usuario modifica los datos necesarios para la orden de trabajo.
6) El usuario selecciona confirmar.
7) El sistema valida las entradas del usuario en los cuadros de texto y otros controles.
8) El sistema edita los datos de la orden de trabajo en el servicio web.
9) El servicio web inserta los datos modificados en la Base de Datos.

Extensiones (Flujo Alternativo):

6.a) El usuario selecciona cancelar.

6.a.1) El sistema sale de la edición de orden de trabajo sin editar ningún campo.

7.a) No hay conexión.

7.a.1) El sistema notifica que no se puede editar la orden de trabajo correspondiente.

Caso de uso: Cerrar Orden de Trabajo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado, con su parque y su listado de ordenes del parque, selecciona una orden para editar.

Postcondicones: El personal operativo cierra la orden de trabajo seleccionada.

Descripción (Flujo Básico):

1) El sistema muestra el formulario de la ordenes de trabajo seleccionada.
2) El usuario selecciona Cerrar.
3) El sistema muestra el dialogo de Cierre de Orden de Trabajo.
4) El usuario selecciona la confirmación.
5) El sistema valida el cierre de la orden de trabajo.
6) El sistema cierra la orden de trabajo en el servicio web.
7) El servicio web cierra la orden de trabajo en la Base de Datos.

Extensiones (Flujo Alternativo):

4.a) El usuario pulsa el selecciona No.

4.a.1) El sistema sale del dialogo de cierre sin cerrar la orden.

6.a) No hay conexión.

6.a.1) El sistema notifica que no se puede cerrar el activo.

Caso de uso: Añadir Orden de Trabajo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado el

listado de ordenes de trabajo.

Postcondicones: El personal operativo añade la orden de trabajo.

Descripción (Flujo Básico):

1) El sistema muestra el listado de ordenes de trabajo.
2) El usuario selecciona Menú del teclado del dispositivo.
3) El sistema muestra el Menú Añadir.
4) El usuario selecciona Añadir.
5) El usuario añade los datos necesarios para rellenar la orden de trabajo.
6) El usuario selecciona confirmar.
7) El sistema valida las entradas del usuario en los cuadros de texto y otros controles.
8) El sistema añade la orden de trabajo en el servicio web.
9) El servicio web inserta los datos modificados en la Base de Datos.

Extensiones (Flujo Alternativo):

6.a) El usuario selecciona cancelar.

6.a.1) El sistema sale del registro de activos sin añadir ninguna orden.

7.a) No hay conexión.

7.a.1) El sistema notifica que no se puede añadir la orden correspondiente.

Caso de uso: Seleccionar Contacto

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado editar o añadir una orden de trabajo.

Postcondicones: El personal operativo asigna un contacto a una orden de trabajo.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de contactos del Consorcio del servicio web.
2) El sistema muestra el listado de contactos.
3) El usuario elige el contacto al que se va a asignar en la orden.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de contactos.

Caso de uso: Seleccionar Usuario

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado editar o añadir una orden de trabajo.

Postcondicones: El personal operativo asigna un usuario a una orden de trabajo.

Descripción (Flujo Básico):

4) El sistema obtiene el listado de usuarios del Consorcio del servicio web.
5) El sistema muestra el listado de usuarios.
6) El usuario elige el usuario al que se va a asignar en la orden.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de usuarios.

Caso de uso: Seleccionar Activo

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo autenticado y con su parque, ha seleccionado editar o añadir una orden de trabajo.

Postcondicones: El personal operativo asigna un activo a una orden de trabajo.

Descripción (Flujo Básico):

7) El sistema obtiene el listado de activos del Consorcio del servicio web.
8) El sistema muestra el listado de activos.
9) El usuario elige el activo al que se va a asignar en la orden.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de activos.

Caso de uso: Consultar Historial

Actor primario: Personal Operativo.

Actores secundarios: -

Precondicones: El personal operativo se ha autenticado en el sistema y ha seleccionado su parque de Bomberos.

Postcondicones: El personal operativo obtiene el listado de ordenes de trabajo para el activo seleccionado en el parque.

Descripción (Flujo Básico):

1) El sistema obtiene el listado de activos del parque seleccionado del servicio web.
2) El sistema muestra el listado de activos.
3) El usuario selecciona un activo.
4) El sistema obtiene el listado de ordenes de trabajo del parque para ese activo seleccionado del servicio web.
5) El sistema muestra el listado de ordenes de trabajo filtrada.

Extensiones (Flujo Alternativo):

1.a) No hay conexión.

1.a.1) El sistema notifica que no se puede obtener el listado de activos.

3.2 Diagrama de clases

Una vez definidos los casos de uso, se procede a diseñar las características mediante diagramas UML. En primer lugar, dado que se trata de una aplicación de tratamiento de datos principalmente, lo primero es definir el modelo de datos, es decir las distintas clases de objetos que se necesitan para implementar todas las características que se han tratado en los casos de uso. El diagrama de clases no sólo se realiza en base a las especificaciones dadas por la aplicación sino también se ha orientado en torno a los servicios web REST diseñados con los que se conectara la aplicación, así como también con las diferentes base de datos donde se almacena toda la información.

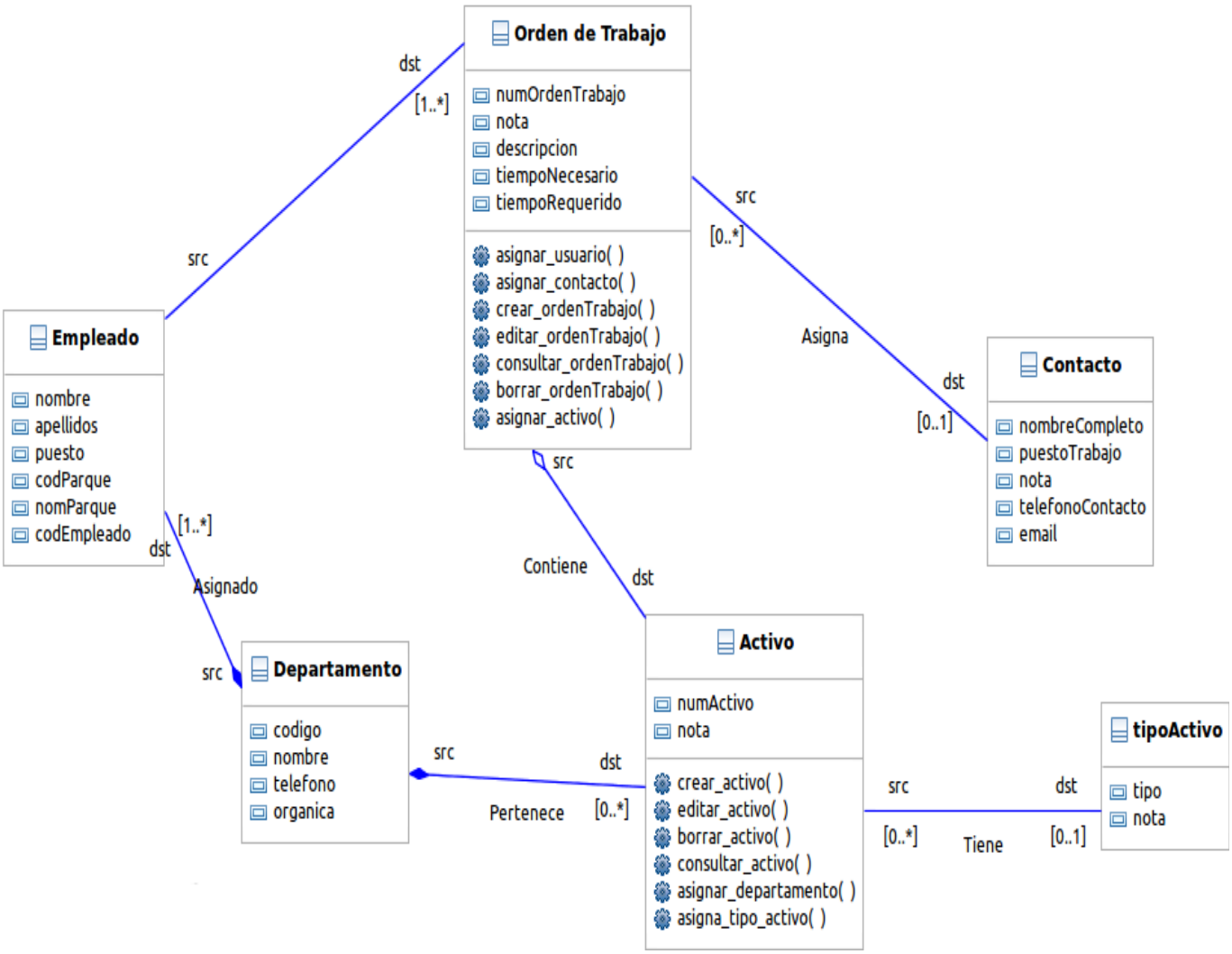

Figura 4: Diagrama de clases

3.3 Diagramas de secuencias

El siguiente paso ha sido la elaboración de los diagramas de secuencias para mostrar la interacción de un conjunto de objetos en una aplicación a través del tiempo modelándose para cada caso de uso. Este contiene detalles de la implementación del escenario, incluyendo objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. Dado que ya tenemos las descripción de los casos de uso como una secuencia de varios pasos, entonces se puede pasar sobre esos pasos para obtener que objetos son necesarios para que se pueda construir dicho diagrama de secuencias.

Para diseñar los diagramas de secuencias se ha utilizado un programa llamado "Quick Sequence Diagram Editor" que consiste en una herramienta de desarrollo construida en Java 5 para generar UML de forma profesional con diagramas de secuencias mediante sencillas lineas de código.

Sus principales ventajas para utilizar esta herramienta, han sido principalmente que puede ser exportado a una imagen para introducirlo de forma sencilla en la memoria del proyecto y que cambia de forma automática mediante la introducción del código .

Diagrama de secuencia: Iniciar Sesión

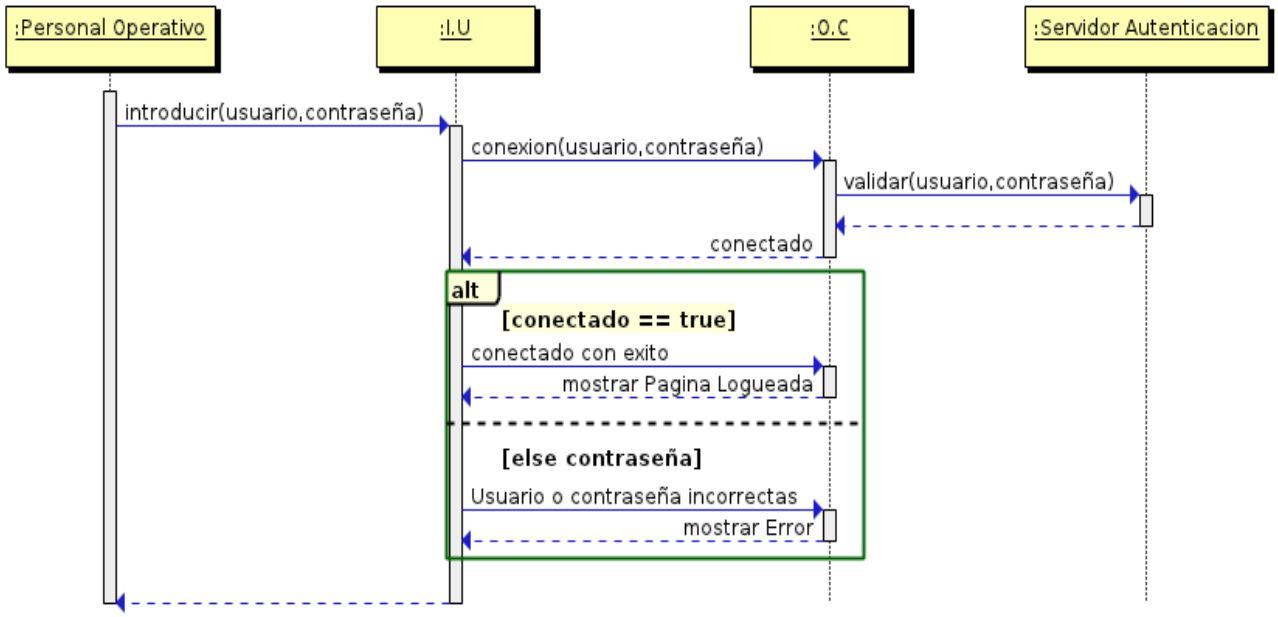

Figura 5: Iniciar sesión

Diagrama de secuencia: Seleccionar Parque

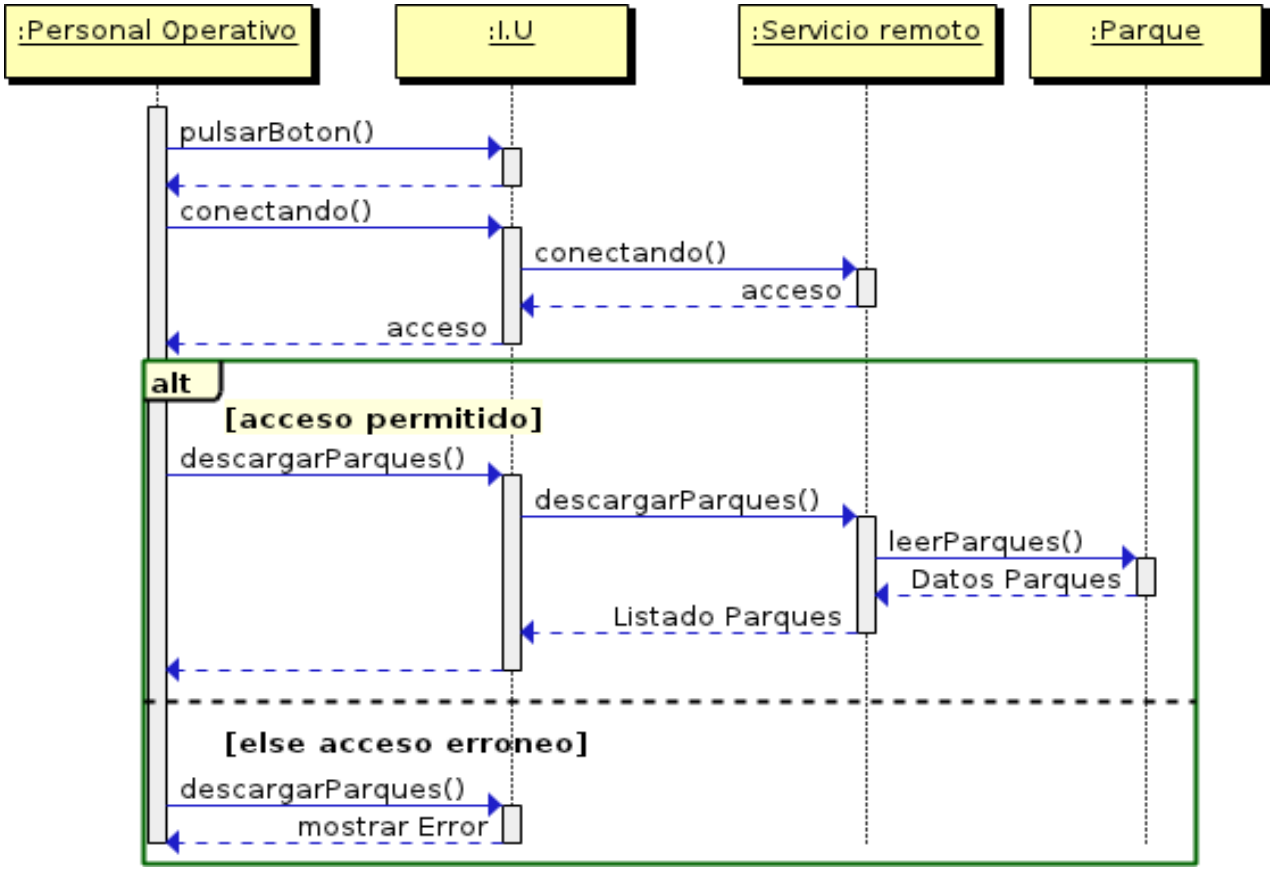

Figura 6: Seleccionar parque

Diagrama de secuencia: Consultar Activo

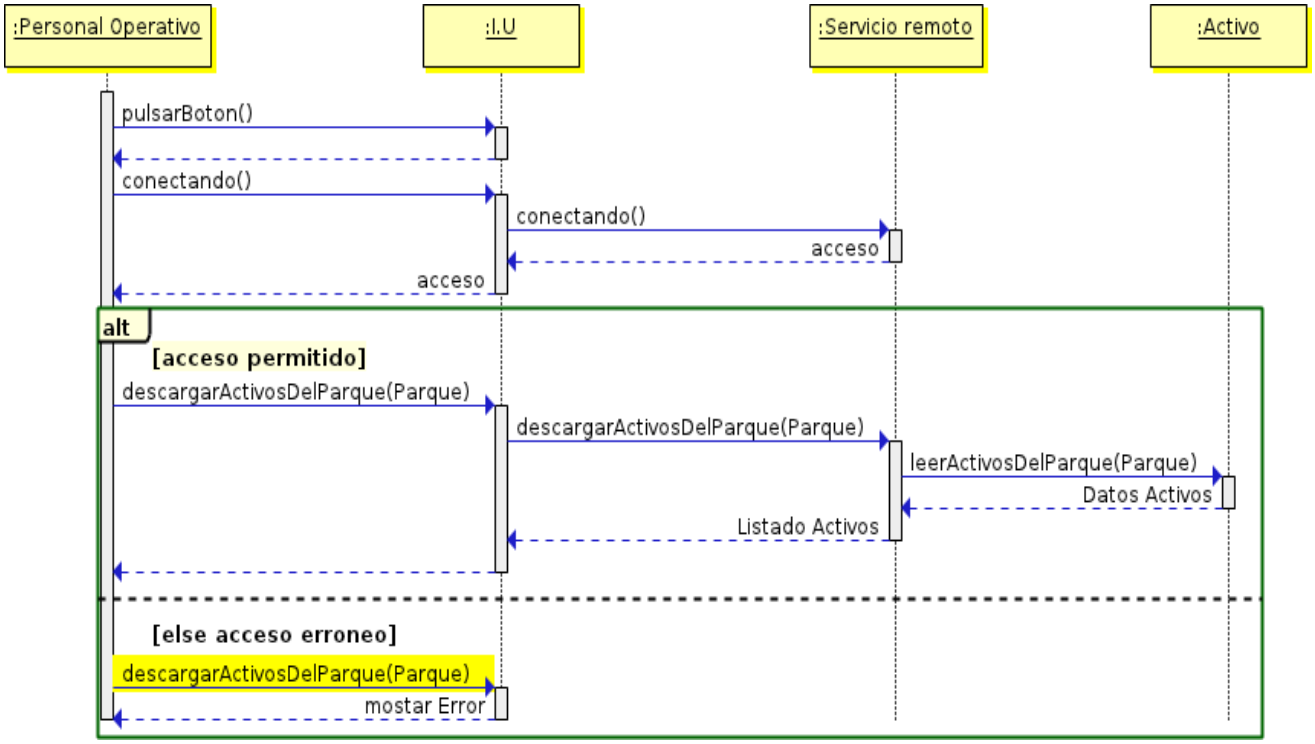

Figura 7: Consultar activo

Diagrama de secuencia: Editar Activos

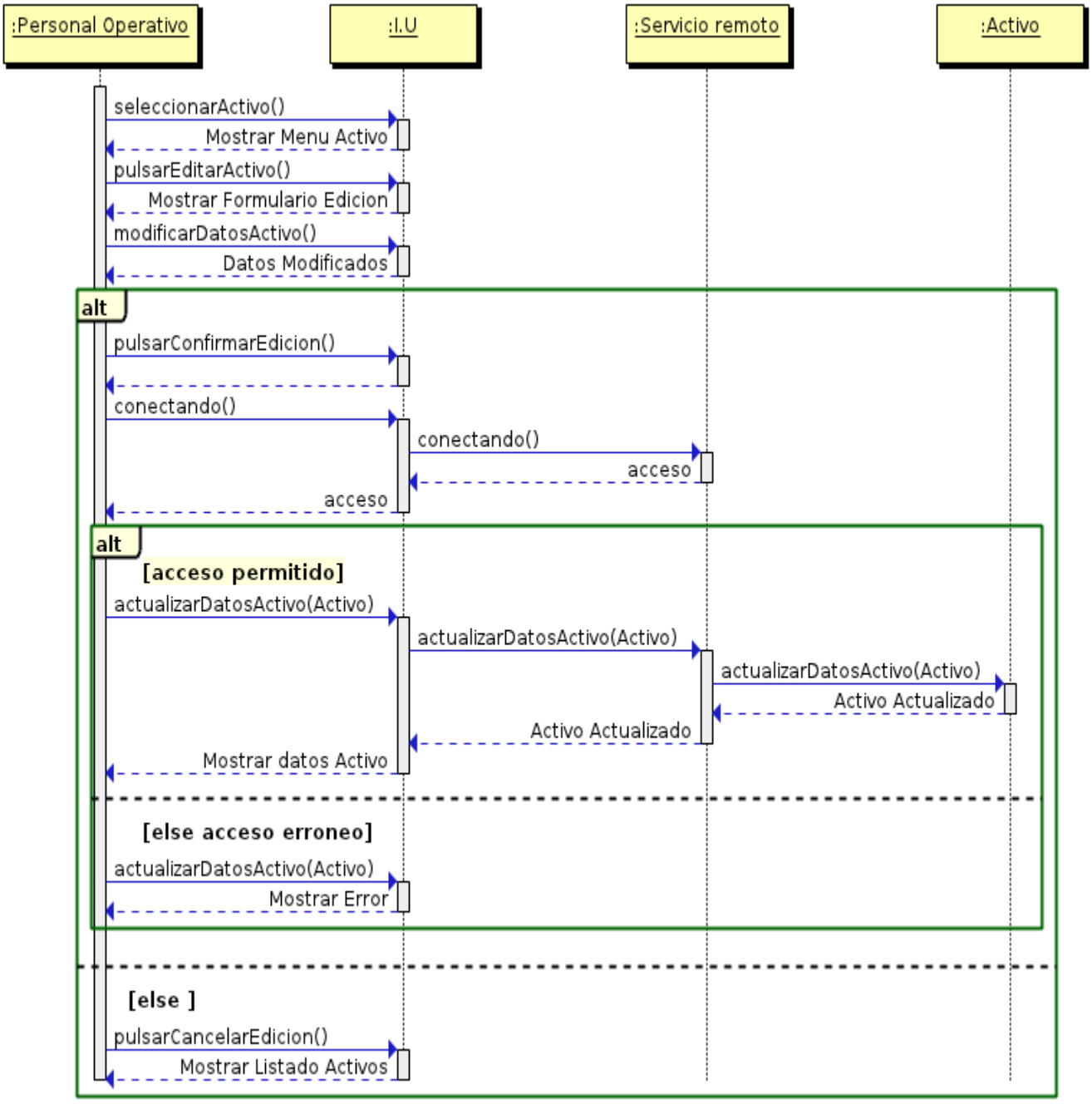

Figura 8: Editar activo

Diagrama de secuencia: Borrar Activos

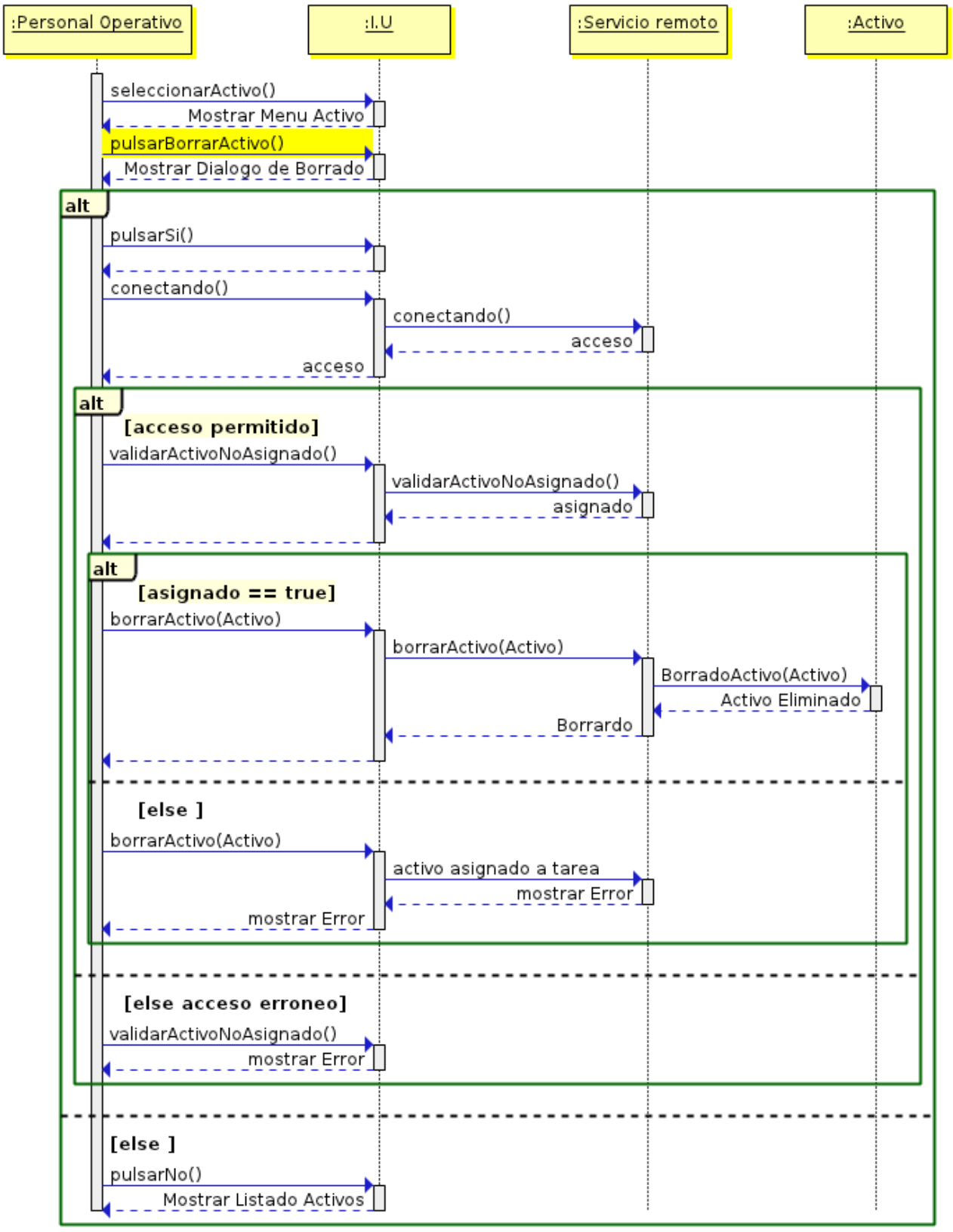

Figura 9: Borrar activo

Diagrama de secuencia: Añadir Activos

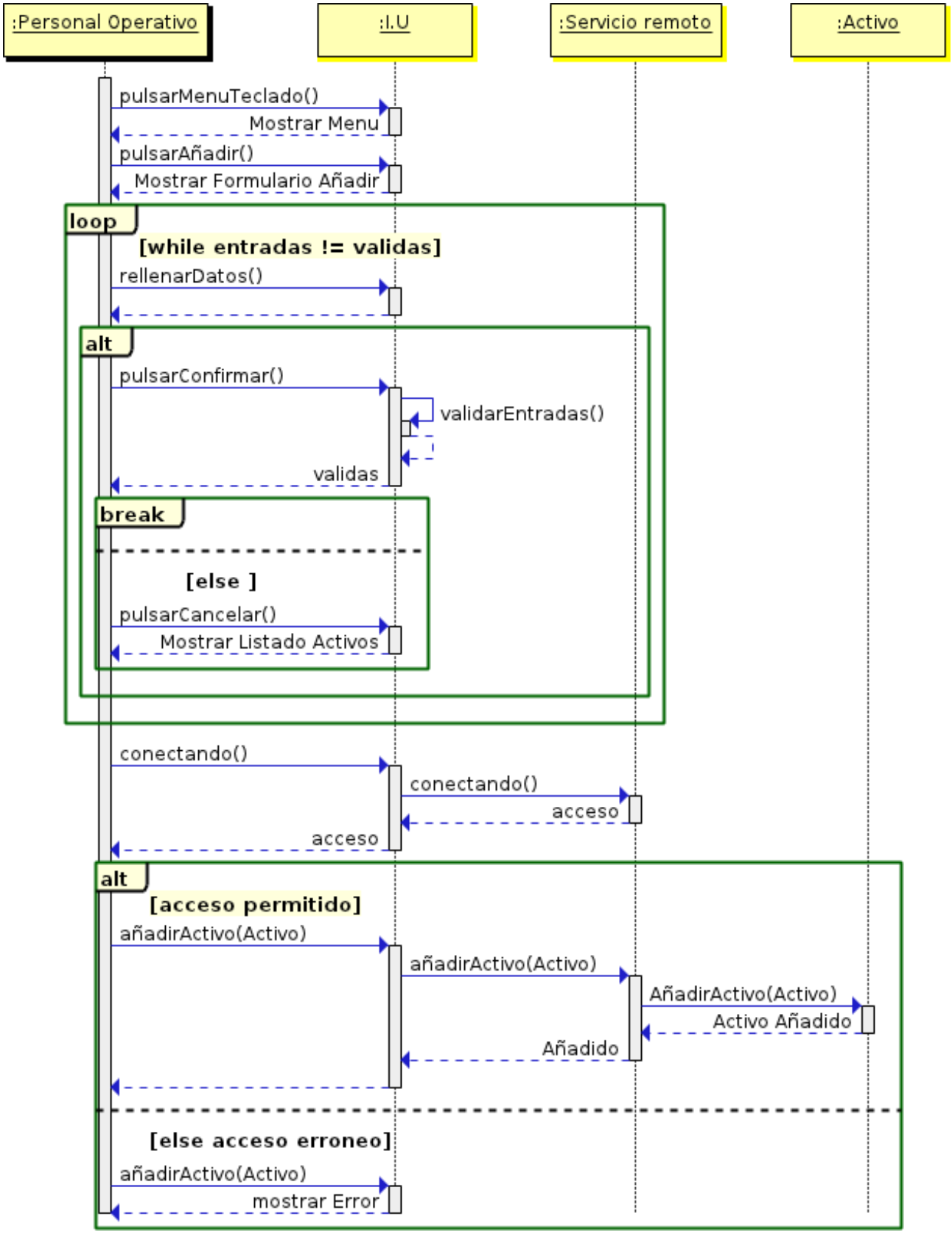

Figura 10: Añadir activo

Diagrama de secuencia: Seleccionar Tipo Activo

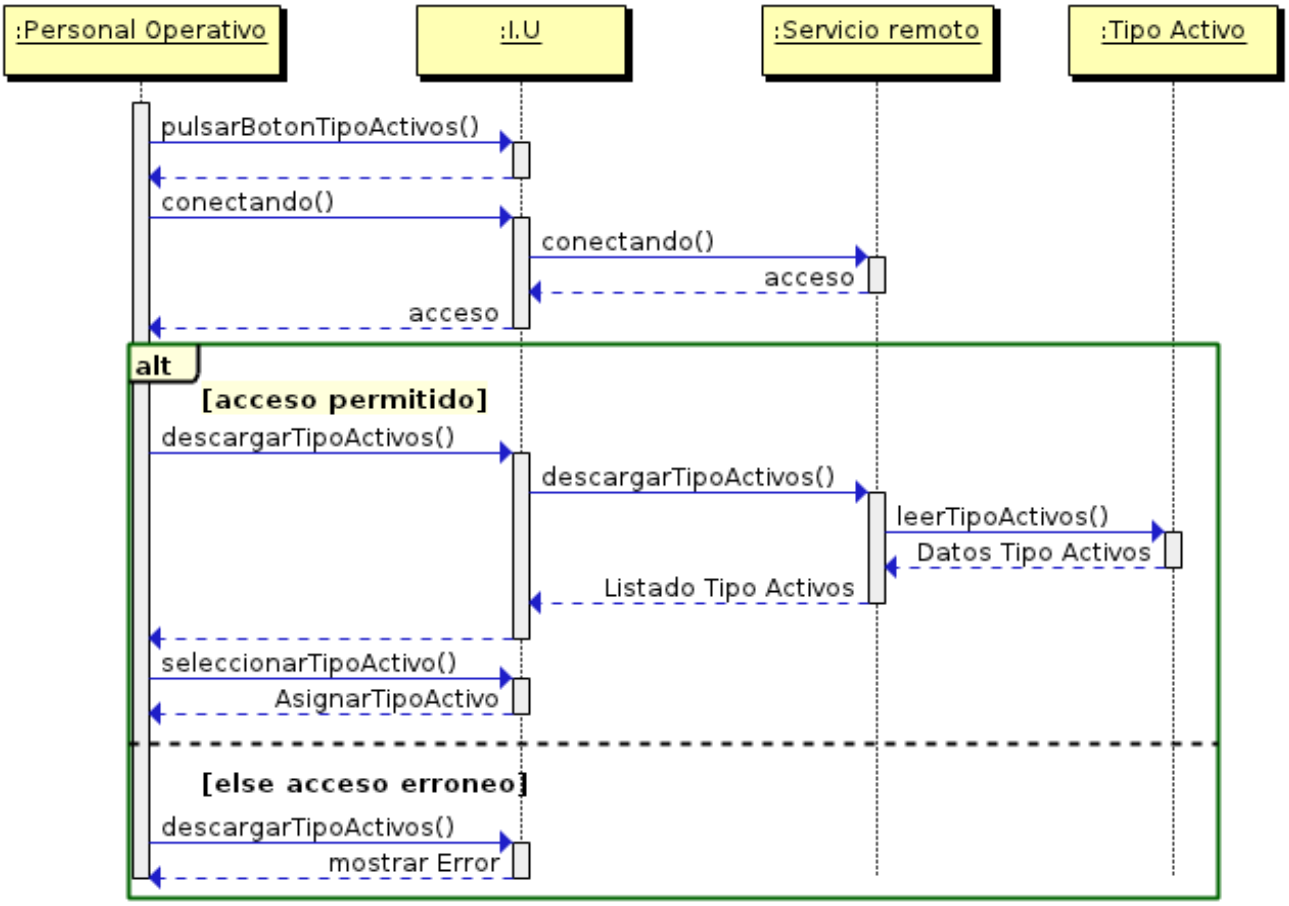

Figura 11: Seleccionar tipo activo

Diagrama de secuencia: Consultar Orden de Trabajo

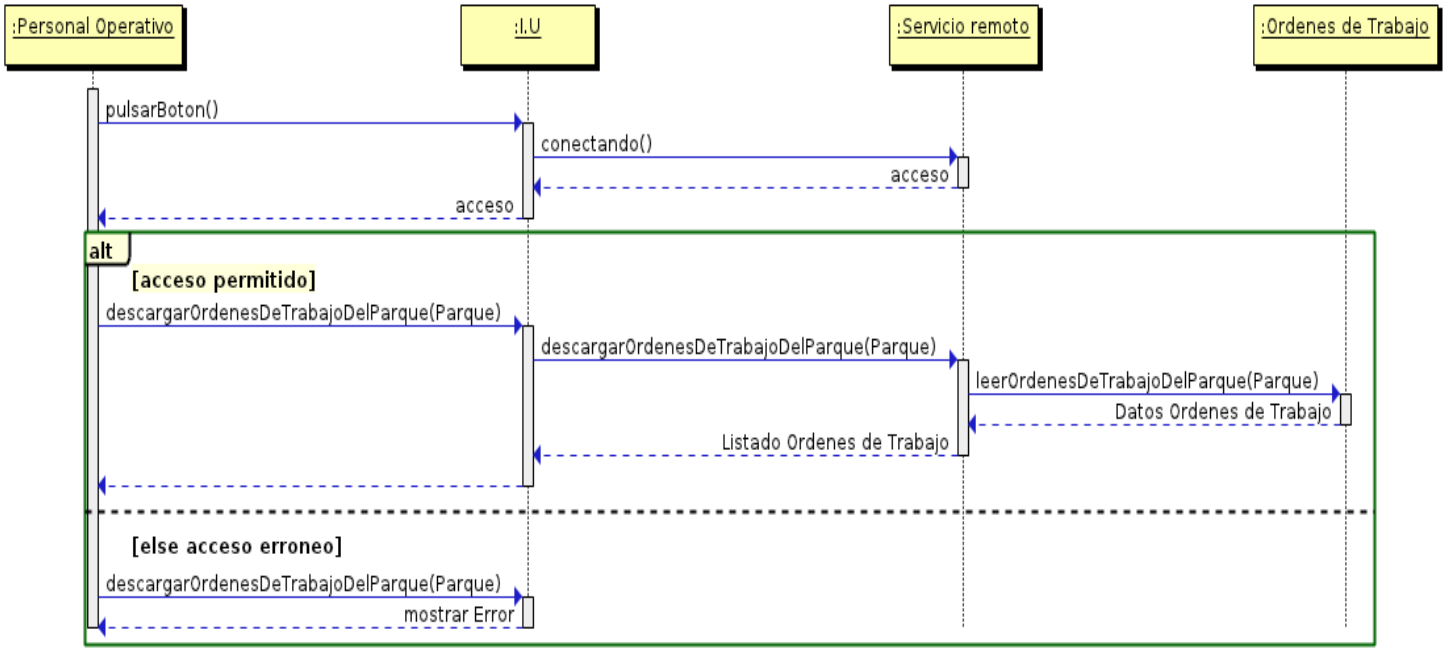

Figura 12: Consultar orden de trabajo

Diagrama de secuencia: Editar Orden de Trabajo

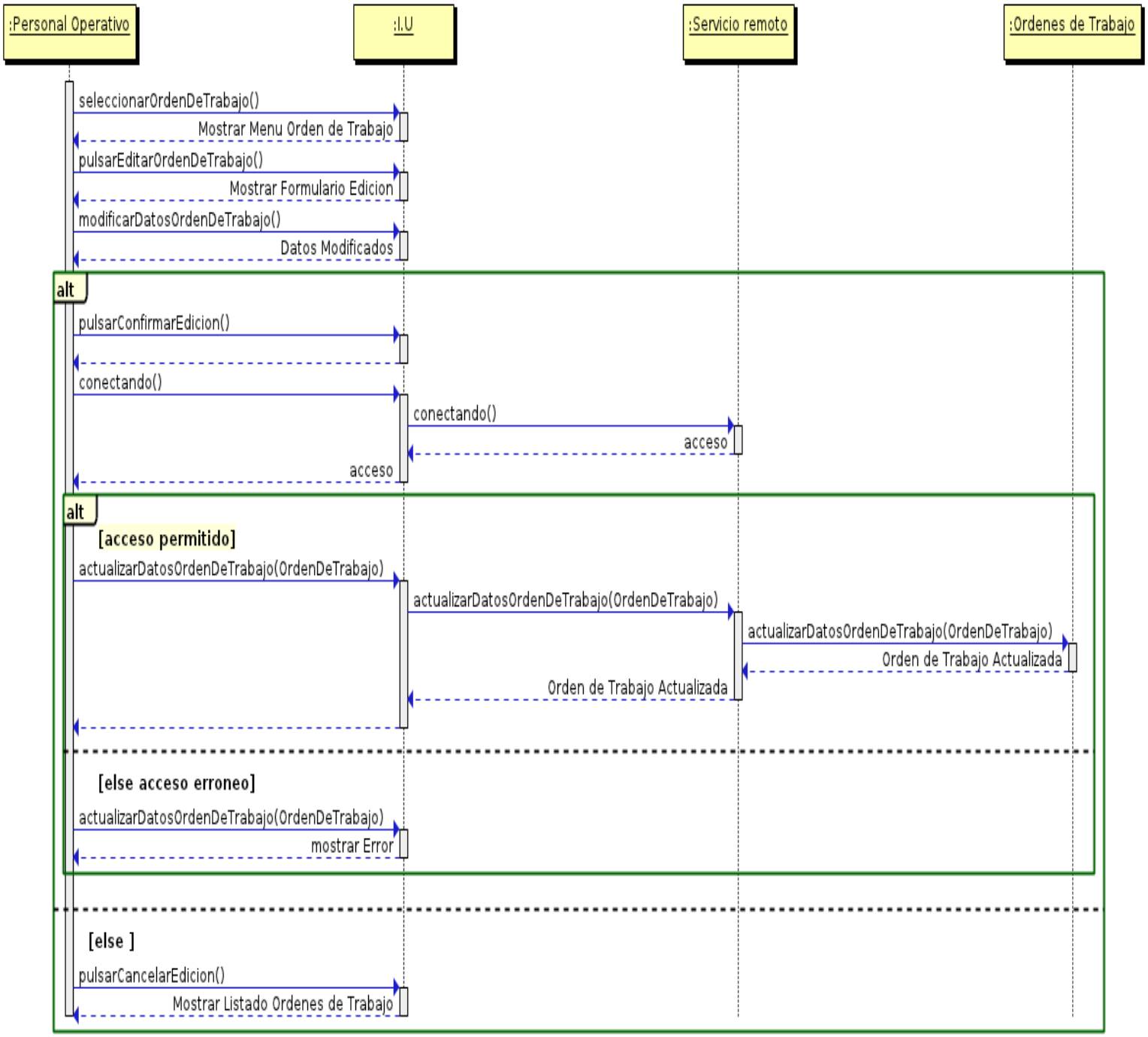

Figura 13: Editar orden de trabajo

Diagrama de secuencia: Cerrar Orden de Trabajo

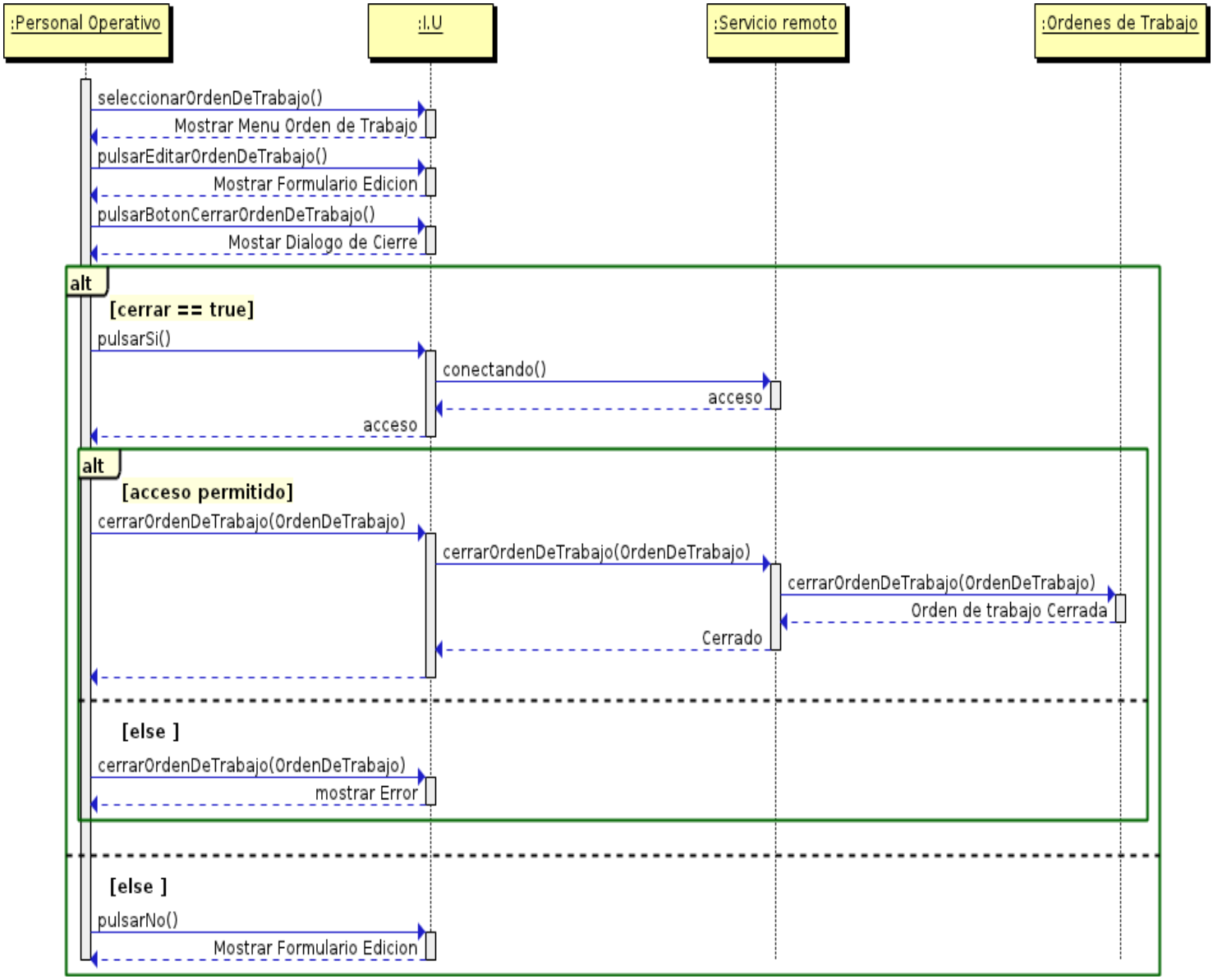

Figura 14: Cerrar orden de trabajo

Diagrama de secuencia: Añadir Orden de Trabajo

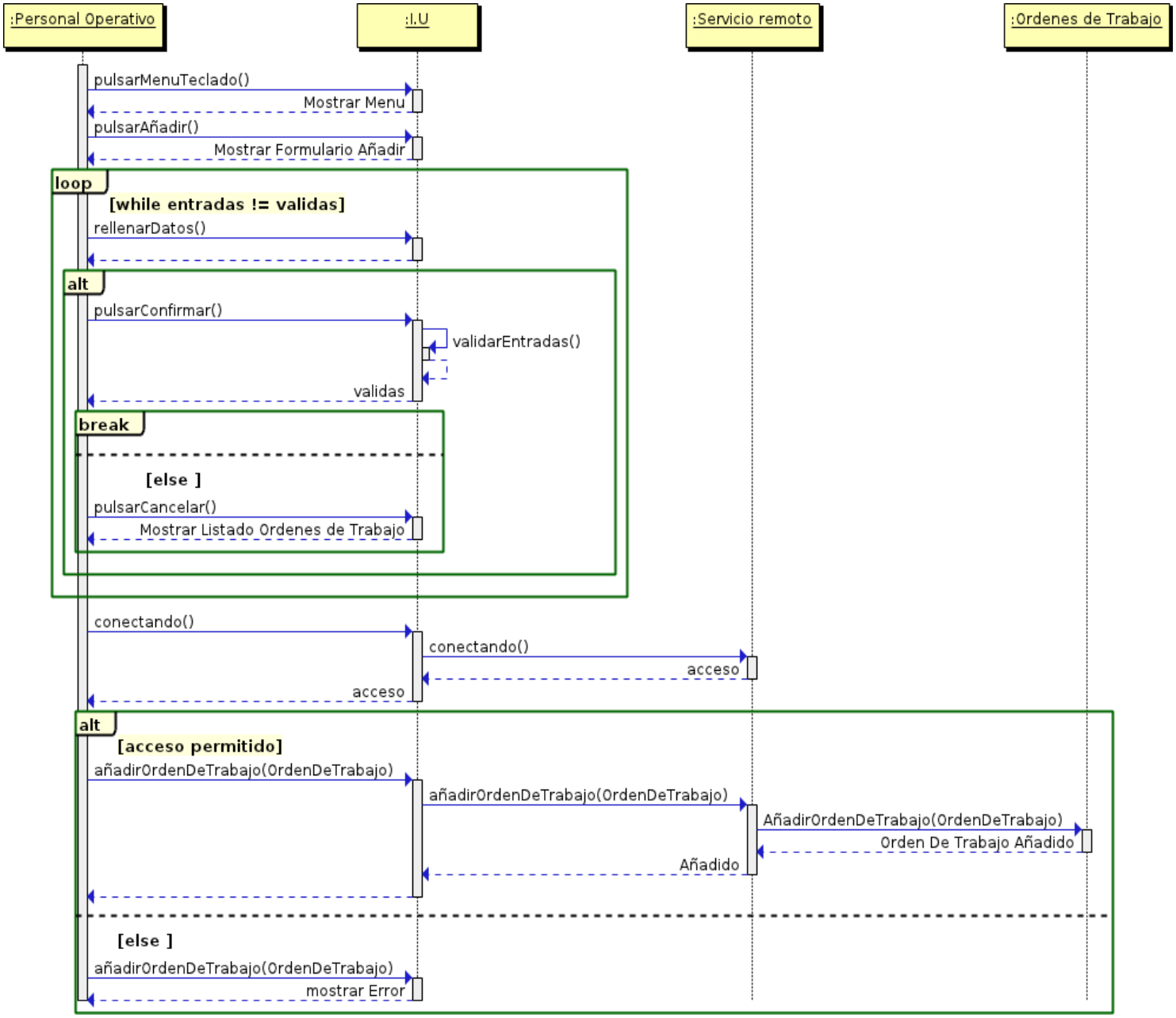

Figura 15: Añadir orden de trabajo

Diagrama de secuencia: Seleccionar Contacto

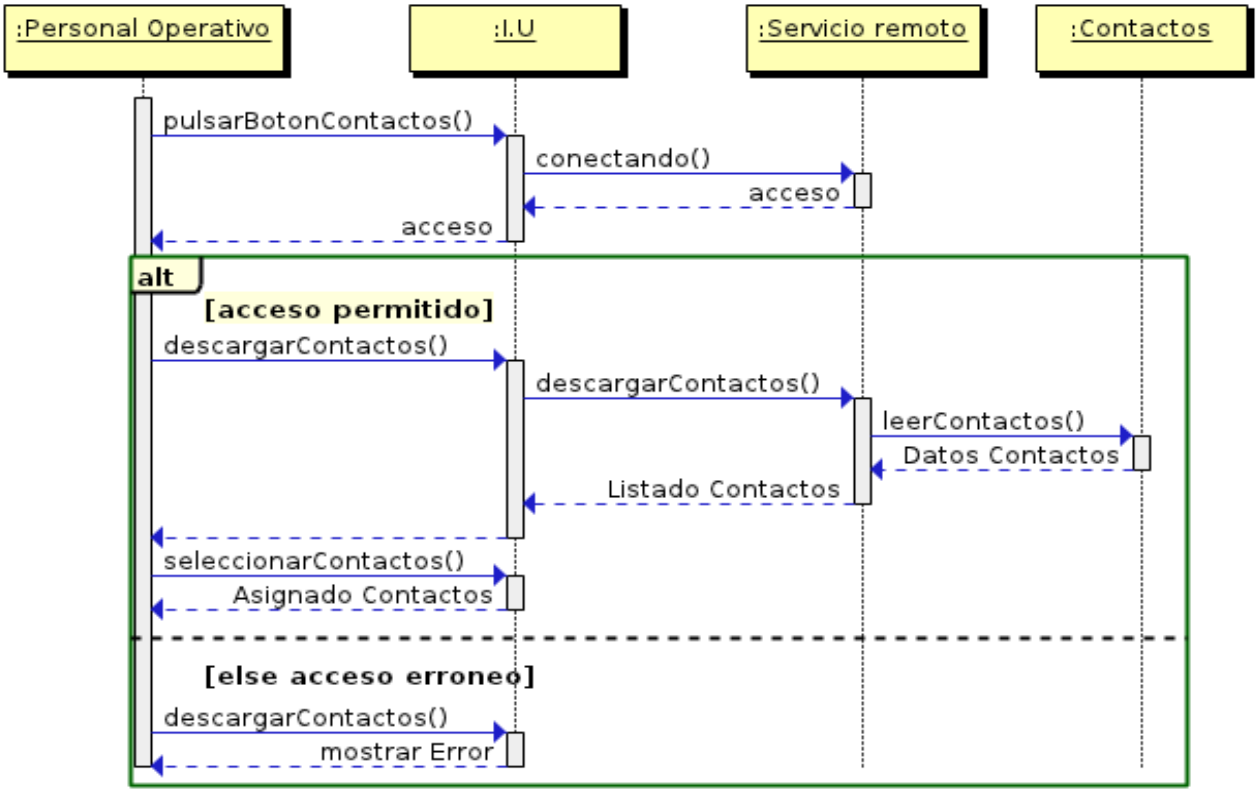

Figura 16: Seleccionar contacto

Diagrama de secuencia: Seleccionar Usuario

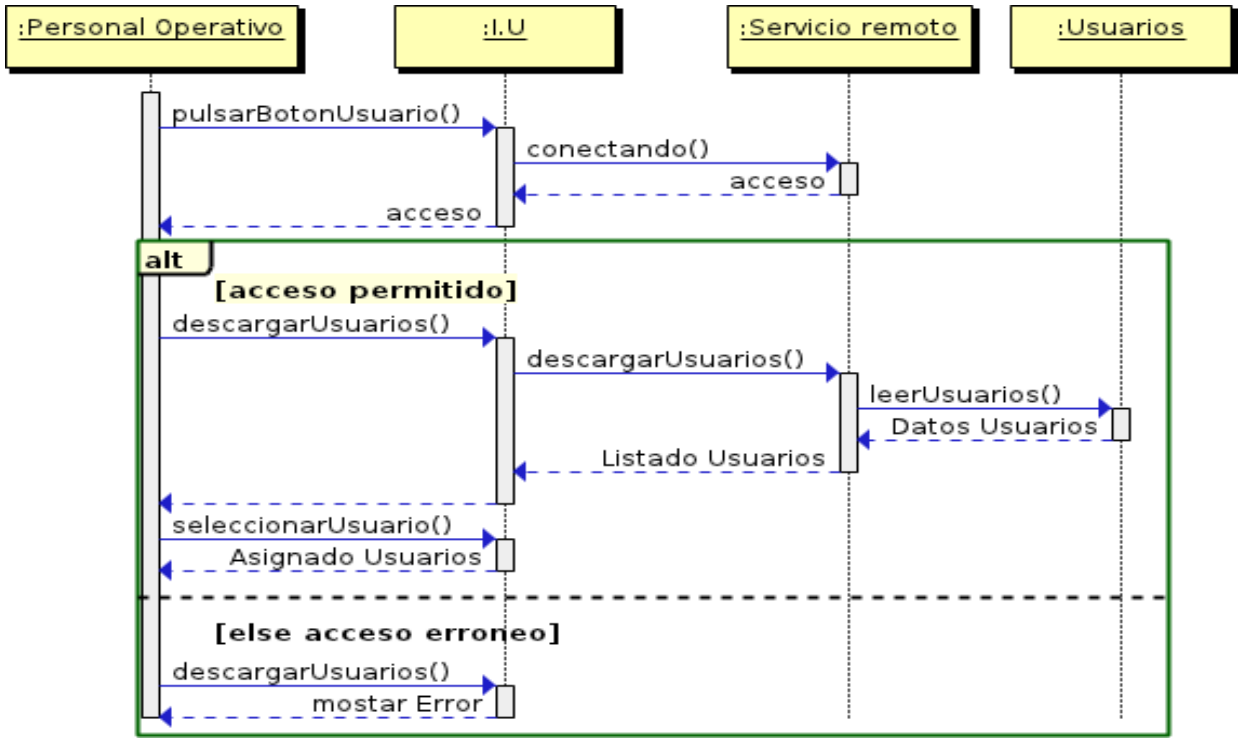

Figura 17: Seleccionar usuario

Diagrama de secuencia: Seleccionar Activo

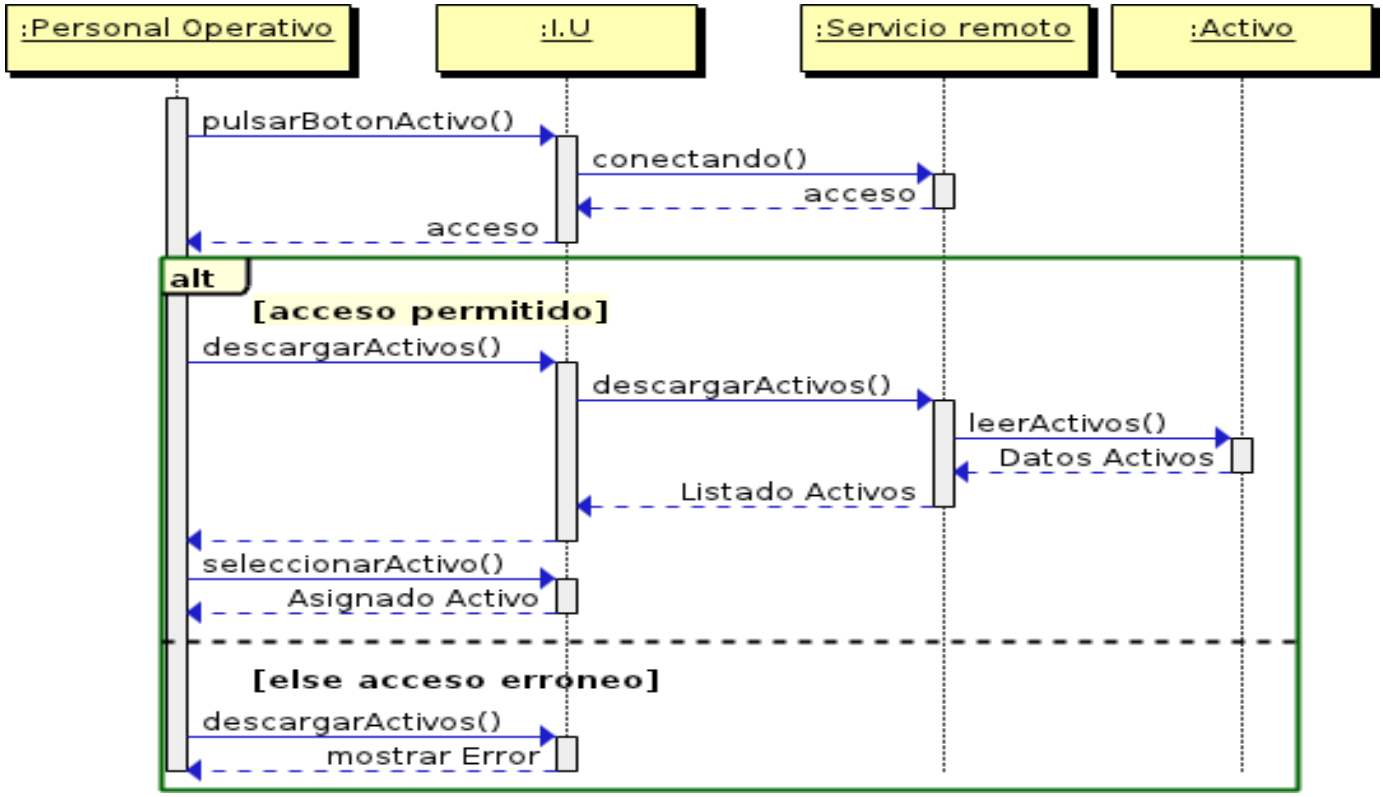

Figura 18: Seleccionar activo

Diagrama de secuencia: Consultar Historial

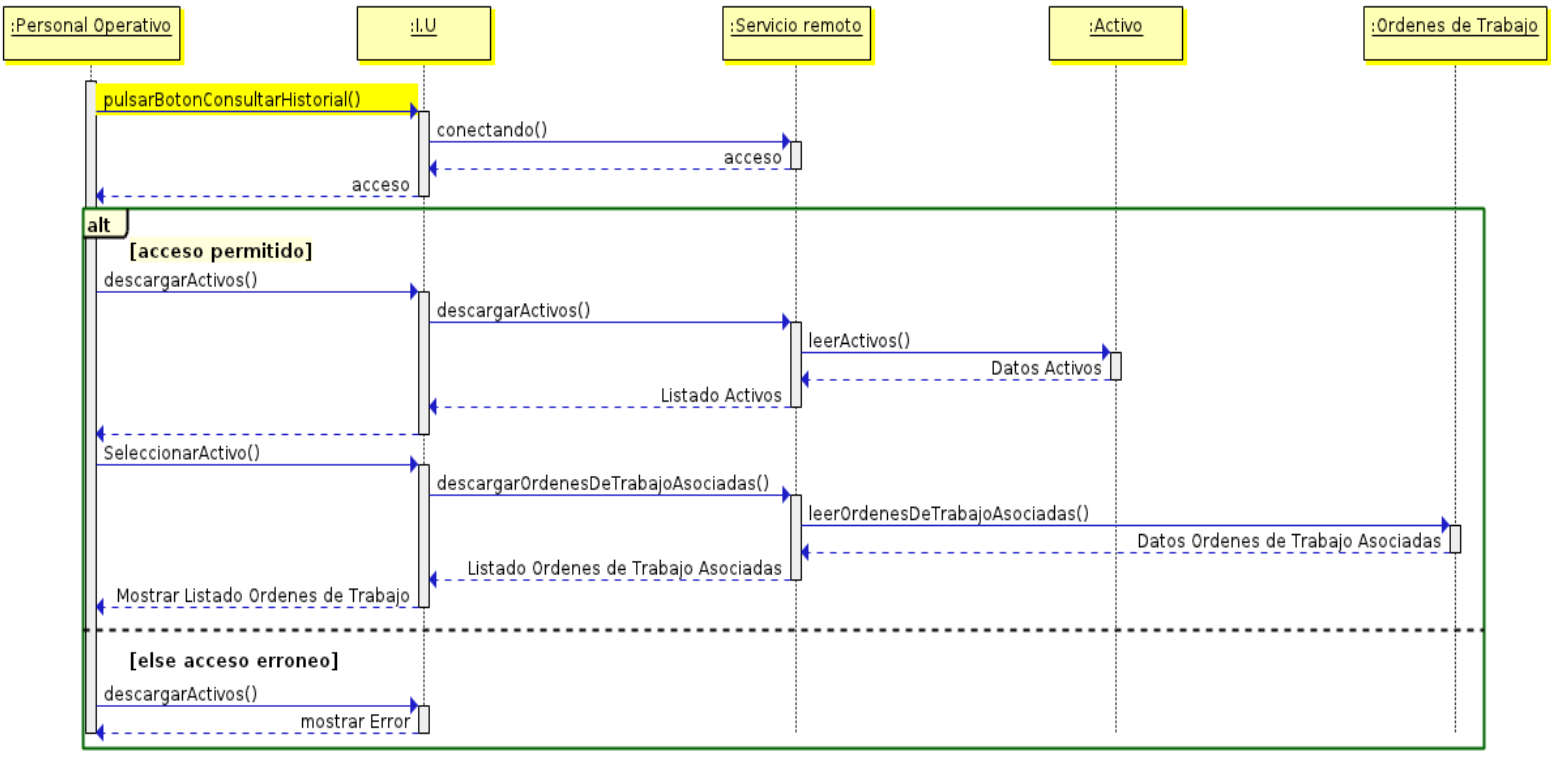

Figura 19: Consultar historial

4 Fase de Construcción

En la siguiente fase se pretende alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase y siguiendo con el proceso de implementación RUP acabaremos de implementar los artefactos necesarios, como el diagrama de componentes y de despliegue para modelar las distintas partes del sistema y el modelo relacional para la base de datos. Además los integraremos y testearemos obteniendo una versión beta del producto que se pueda poner en manos de los usuarios.

Por otra parte, cabe decir que en la fase de construcción se ha llevado a cabo un proceso iterativo, implementado bajo el sistema operativo Android por parte del cliente e implementado en Java por parte del servicio web (servidor) REST.

En primer lugar, el cliente se ha desarrollado bajo la versión 2.2 bajo la API 8 del sistema operativo comentando anteriormente bajo el entorno de desarrollo Eclipse y con el Plug-in Android SDK.

En segundo lugar, el servidor o servicio web REST se ha implementado principalmente bajo el lenguaje de programación Java, además de utilizar el entorno de desarrollo Eclipse y un servidor web o contenedor de servlets Apache Tomcat. En nuestro caso, utilizamos la versión 7 de Apache Tomcat que tiene como principales características que esta implementado en Servlet 3.0, JSP 2.2 y EL 2.2, además de contener mejoras para detectar y prevenir “fugas de memoria” en las aplicaciones web, una mayor limpieza de código y soporte para contenidos externos.

La aplicación cliente desarrollada principalmente se encarga de gestionar los activos y las tareas de mantenimiento para dichos activos en cada parque de bomberos para un usuarios (personal operativo) autenticado previamente, obteniendo los datos del servicio web REST, el cual se encargar de posteriormente, almacenar y gestionar dichos datos con las bases de datos del Consorcio Provincial de Bomberos de Valencia.

Para esta fase, y una vez tenidos los conceptos básicos claros se ha programado tanto la aplicación cliente como el servicio web (servidor) siguiendo el estilo de programación por capas.

4.1 Arquitectura de la aplicación

La programación por capas, se puede definir como una arquitectura cliente-servidor donde el objetivo primordial es la separación lógica de negocio de la lógica de diseño, es decir, separando la capa de datos con la capa de presentación al usuario.

En nuestro caso utilizaremos la programación en tres capas, la mas utilizada, donde cada aplicación se divide en tres capas bien diferenciadas:

- Capa de Presentación: Es la que ve el usuario, presentando la aplicación, le comunica la información y captura la información que el usuario le indica realizando una validación previa, es decir, en nuestro proyecto será la interfaz gráfica de la aplicación
- Capa de Negocio: Es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados y con la capa de datos para solicitar al gestor (en nuestro caso el servicio web REST) almacenar o recuperar datos de él.
- Capa de Datos: Es donde residen los datos y es la encargada de acceder a los mismo. En el proyecto estará formado por el servicio web REST al que accede la aplicación móvil cuando necesitar información, para realizar almacenamiento de datos o para actualización de datos.

Por otra parte, y para finalizar con el apartado de la arquitectura en la que se ha basado dicha programación, cabe destacar que la arquitectura de la solución es por tres capas y dos niveles. Las tres capas han sido detalladas previamente, pero los dos niveles serán necesarios porque la solución reside en dos componentes, por un lado tenemos la presentación + lógica que reside en la aplicación móvil y por tanto en los dispositivos móviles donde se utilizará la aplicación y, por otro lado, tenemos la lógica + datos del servicio web REST que se implementa en un servidor de la sede central del Consorcio Provincial de Bomberos de Valencia y que gracias a toda la interconexión existente se podrán comunicar y realizar las peticiones pertinentes.

Finalmente se puede observar que la principal ventaja para este tipo de arquitectura de programación es la independencia que existe entre niveles, y que permite hacer cambios en un nivel, sin tener que afectar al resto.

4.2 Proceso de Desarrollo

La implementación de la aplicación ha sido desarrollada de forma iterativa y creciente siguiendo el framework (entorno de trabajo) RUP (Rational Unified Process). La idea principal es desarrollar la aplicación de forma incremental, sacando ventaja de lo que se ha aprendido a lo largo del desarrollo e incrementándolo en forma de versiones posteriores o entregables. Dicho aprendizaje se puede obtener por dos caminos, mientras se esta desarrollando la iteración o cuando se prueba dicha versión con el cliente, obteniendo así los requisitos para la siguiente versión. Los pasos para tener éxito en este proceso de desarrollo son comenzar con una implementación simple de los requerimientos e iterativamente mejorar la funcionalidad en las distintas versiones hasta que el sistema completo esté implementado.

El proceso en sí mismo consiste de una etapa de inicialización donde se crea la versión del sistema y un producto con el que el usuario pueda interactuar con el proceso, ofreciendo una muestra de los aspectos del problema y obteniendo una solución simple. Para guiar el proceso de iteración se crea una lista de control de proyecto, que contendrá un historial de todas las tareas realizadas y futuras.

Además contiene una etapa de iteración donde involucra el rediseño e implementación de las tareas de la lista de control de proyecto. La meta del diseño es ser simple y modular para soportar el rediseño de la etapa. El análisis de una iteración se basa principalmente en la retroalimentación con el cliente y en las funcionalidades a introducir de la lista de control de proyecto.

En nuestra aplicación se desarrollaron tres iteraciones básicas y bien diferenciadas para abordar el proyecto de mantenimiento de equipos y sistemas del Consorcio:

- Primera Iteración: Diseño y construcción del servicio web REST, lo que conlleva que el servicio web pueda realizar las principales funciones (CRUD), es decir, crear, obtener, actualizar y borrar las distintas clases que necesitemos para el proyecto. Introducción a la programación con Android. Además se realiza la conexión entre los distintos sistemas que utilizaremos para realizar dicho proyecto, es decir, la conexión cliente-servicio web REST para consultar la información para las tareas de mantenimiento y la conexión servicio web-bases de datos para el almacenamiento y obtención de la información necesaria. Por otra parte, se comienza con la

implementación de la aplicación de forma inicial, tan solo la consulta de la información obtenida del servicio web REST para poder observar que los datos obtenidos son los correctos en cada caso. Finalmente en esta primera iteración, ser realiza un primer boceto del diseño de la interfaz gráfica de la aplicación móvil, con lo que ya podremos realizar la implementación de la aplicación en iteraciones posteriores.

- Segunda iteración: En primer lugar y tras la aceptación del diseño del cliente se realiza la implementación final de la interfaz gráfica de la aplicación móvil. Seguidamente, y por lo que respecta el servicio web REST, se acaban de añadir todas las clases necesarias para la gestión del mantenimiento en los dispositivos móviles. Por otro lado, en el cliente se implementa una primera versión de la gestión de activos y tareas en la aplicación eso conlleva las funciones básicas (CRUD) que se implementan para la aplicación móvil. Por último, se acuerda con el cliente, realizar una autenticación para que tan solo el personal operativo del Consorcio pueda acceder a la aplicación, por lo que se decide realizar un pequeño modulo de autenticación que consistirá en la llamada al servidor de correo IMAP del Consorcio, la gestión de errores en dicho login y su interfaz gráfica correspondiente.

- Tercera iteración: En esta iteración realizamos la implementación de la selección del parque de bomberos donde se va a proceder a realizar la gestión del mantenimiento, a su vez, obtenemos una tarea ligada a la selección del parque y que consiste en el filtrado de activos y ordenes de trabajo por parque seleccionado, por lo que obtendremos tan solo los activos y tareas que realmente nos interesan. A continuación realizamos el diseño e implementación del historial de revisiones que nos aporta información de consulta para los activos que nos interesan, es decir, los del parque correspondiente. Por otro lado, realizamos el diseño e implementación de la gestión de las guías operativas en la que se podrá consultar ficheros pdf para facilitar el mantenimiento. Finalmente, se implementa el cierre de una orden de trabajo cuando se termina un mantenimiento, esta orden pasará del listado de ordenes del historial de revisiones.

Para realizar todos el proyecto se ha utilizado en entorno de trabajo Eclipse, tanto para el servicio web REST como para el cliente mediante el plugin Android SDK.

4.3 Diseño del Sistema

En este apartado se define los componentes, módulos y datos del sistema para satisfacer los requerimientos definidos previamente. Además el diseño de sistema es la primera fase en la cual se selecciona la aproximación básica para resolver el problema, se decide la estructura y el estilo global.

Para continuar con el diseño orientado a objetos se realiza un diagrama de componentes que representa cómo un sistema de software es dividido en componentes y muestra las dependencias entre estos componentes.

4.3.1 Diagrama de Componentes

El diagrama de Componentes es un tipo de diagrama UML que se diseña para obtener una visión estática y dinámica de los componentes físicos y sus dependencias. Estos componentes incluyen archivos, cabeceras, bibliotecas compartidas, módulos, ejecutables o paquetes.

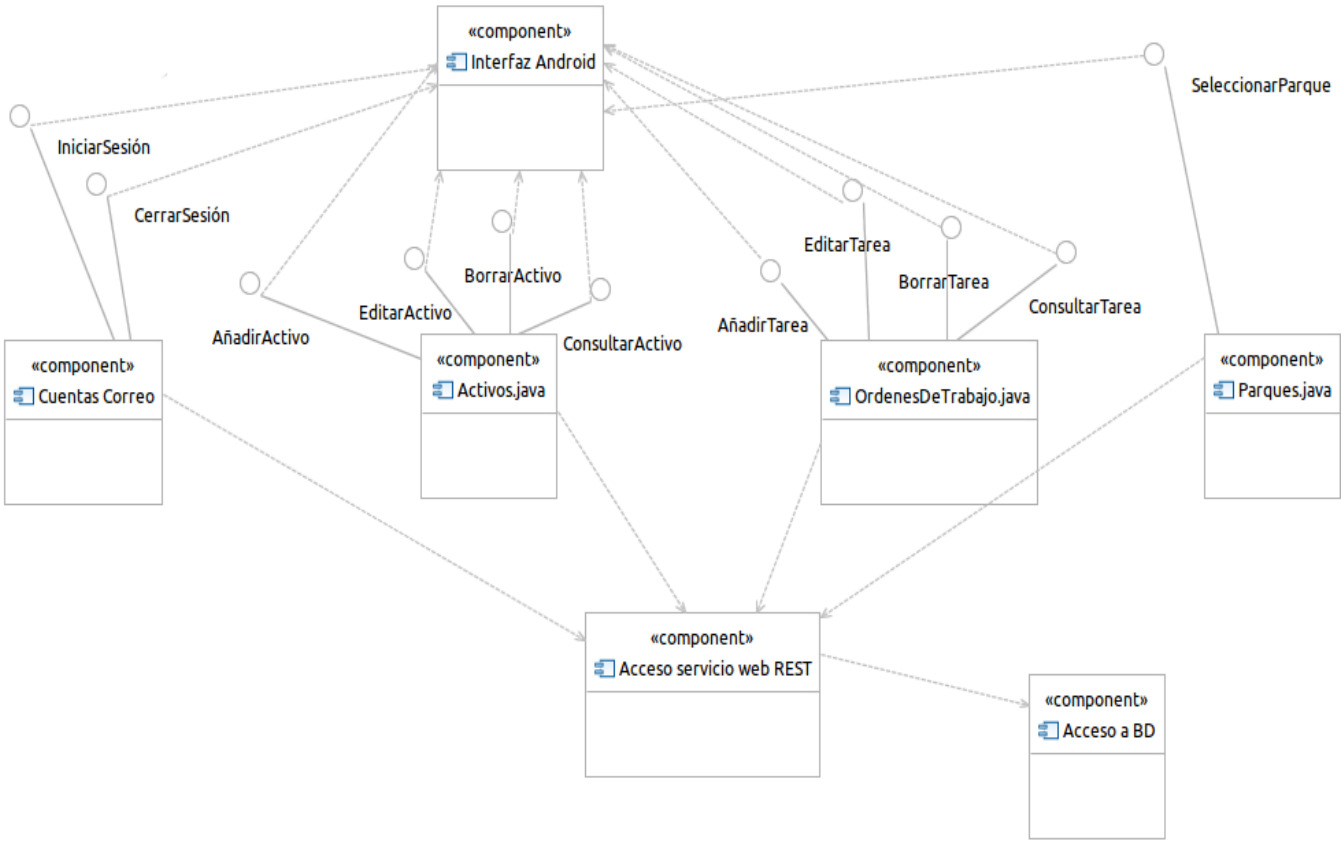

Figura 20: Diagrama de componentes

En el diagrama de componentes diseñado se han implementado interfaces dado que ofrece la ventaja de romper la dependencia directa entre componentes, en este caso entre las clases java y la interfaz Android. Una interfaz contiene una o varias operaciones y se utiliza para especificar los servicios de una clase o de un componente, además se conecta al componente que la implementa a través de una relación de realización, y al componente que utiliza sus servicios con una dependencia.

4.3.2 Diagrama de Despliegue

El diagrama de despliegue es un tipo de diagrama UML que utiliza para modelar el hardware utilizado en las implementaciones de sistemas y las relaciones entre sus componentes.

Para el diseño del diagrama de despliegue utilizamos nodos, componentes y asociaciones con el que se representa la topología del hardware sobre el que se ejecuta el sistema.

En el proyecto se ha implementado un diagrama de despliegue para modelar el sistema cliente-servidor, los cuales son un extremo del espectro de los sistemas distribuidos y requieren la toma de decisiones sobre la conectividad de red de los clientes a los servidores y sobre la distribución física de los componentes software del sistema a través de los nodos.

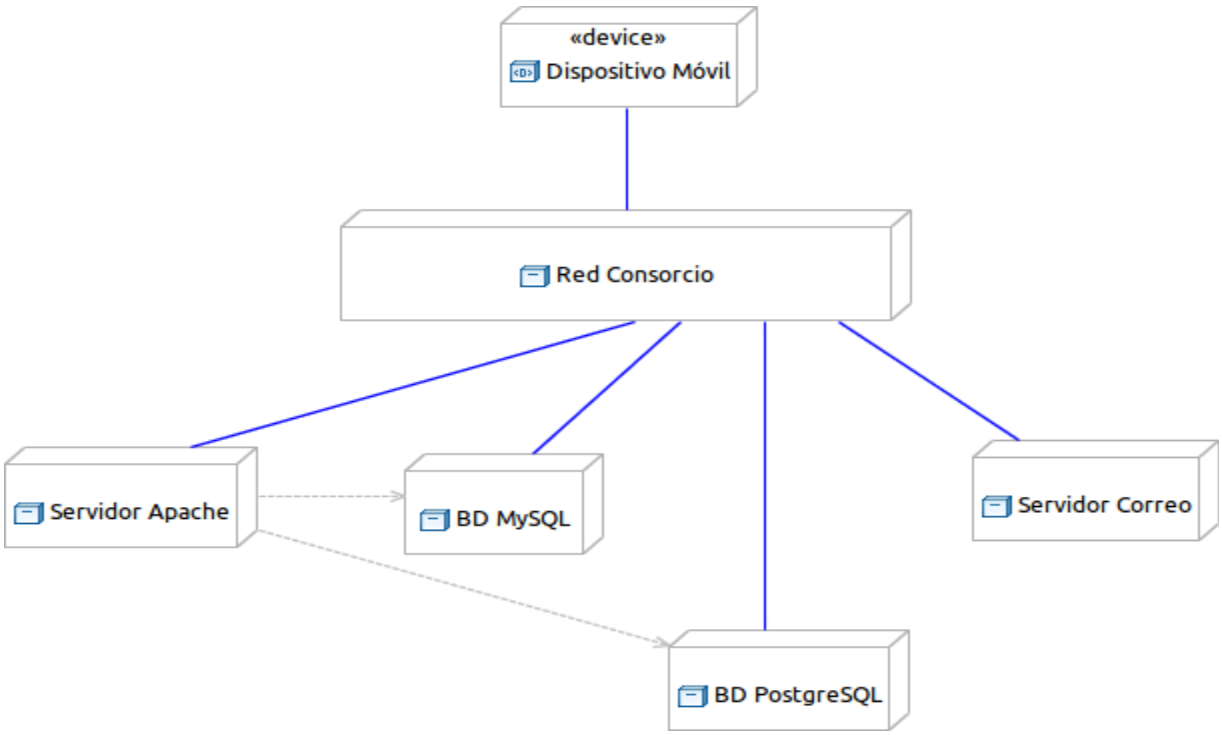

Figura 21: Diagrama de despliegue

En el diagrama de despliegue diseñado se ha implementado la arquitectura hardware en la que se trabajara, por un lado tendremos el dispositivo móvil conectado y configurado debidamente a la red con la aplicación de mantenimiento instalada previamente, por otro lado, disponemos de los distintos servidores a los que se conectara el dispositivo móvil para gestionar el mantenimiento en cada parque.

Además se ha indicado en el diagrama de despliegue el servidor de correo, que resulta de gran utilidad para la implementación de la autenticación en la aplicación Android, dado que todo personal operativo tiene cuenta de correo en la compañía.

Finalmente se observa que el servicio web (en el servidor apache) depende de las bases de datos, tanto de MySQL como de PostgreSQL, de las cuales se obtendrá todos los datos necesarios para la realización con éxito del mantenimiento.

4.4 Programación de la Aplicación

En el siguiente apartado se va a exponer todo el desarrollo y las tecnologías utilizadas para la elaboración de este proyecto analizando los lenguajes y el entorno de desarrollo que se han utilizado para implementar el sistema de mantenimiento, tanto por parte del servidor como por parte del cliente.

Todo esto se detallara profundizando en las iteraciones que se han explicado anteriormente escribiendo la memoria tal como se ha realizado el proyecto, es decir, siguiendo el entorno de trabajo RUP.

4.4.1 Primera Iteración

En esta primera iteración se han analizado todas las tecnologías que se han elegido para implementar el sistema de mantenimiento, tales como los Servicios web REST, Android, Eclipse, Java, el servidor Apache Tomcat...

Seguidamente, se ha implementado una conexión básica para poder gestionar toda la información necesaria para el sistema.

A continuación, realizaremos un primer boceto de la aplicación Android con el plugin Android SDK, que nos muestre un listado de la información contenida en las bases de datos del Consorcio. Además se diseña y se describe la interfaz gráfica de la aplicación Android, para poder realizar la programación en iteraciones sucesivas.

4.4.1.1 Servicio Web REST

Dado que ya se ha comentado las ventajas de utilizar Servicios web REST en el primer capitulo del proyecto, se va a complementar con una pequeña introducción y definición de estos servicios para que el lector pueda comprender finalmente esta utilidad para el proyecto de mantenimiento.

4.4.1.1.1 Introducción a REST

La Transferencia de Estado Representacional (REST) ha ganado enteros como servicio web hasta llegar a ser una alternativa, mas simple, a SOAP y a servicios web basados en el lenguaje de descripción de servicios web (WSDL). Grandes proveedores como Google, Facebook... ya han migrando a esta tecnología para usar un modelo mas fácil, orientado a los recursos.

REST define un conjunto de principios arquitectónicos por los cuales se diseñan servicios web haciendo foco en los recursos del sistema, incluyendo cómo se accede al estado de dichos recursos y cómo se transfieren por HTTP hacia clientes escritos en diversos lenguajes. REST emergió en los últimos años como el modelo predominante para el diseño de servicios logrando un impacto tan grande en la web que prácticamente logró desplazar a SOAP y las interfaces basadas en WSDL por tener un estilo bastante más simple de usar.

REST fue presentado por Roy Fielding en el año 2000 en la Universidad de California, durante la charla académica "Estilos de Arquitectura y el Diseño de Arquitecturas de Software basadas en Redes", analizando un conjunto de principios arquitectónicos de software para usar a la Web como una plataforma de Procesamiento Distribuido.

Años después de su presentación comenzaron a aparecer frameworks REST y se llega a convertir en una parte integral en Java 6 a través de JSR-311.

4.4.1.1.2 Métodos HTTP

Rest utiliza métodos HTTP de manera explícita de manera que resulta una asociación de uno-a-uno entre las operaciones crear, leer, actualizar y borrar y los métodos HTTP:

- **POST** crea o actualiza un recurso en el servidor.

- **GET** define un acceso a la lectura del recurso. El recurso no se cambia mediante una petición GET.
- **PUT** cambia el estado de un recurso o lo actualiza.
- **DELETE** elimina un recurso del servidor.

En nuestro proyecto los servicios web están diseñados para responder a las peticiones HTTP con la gestión de recursos que concuerden con la ruta (o el criterio de búsqueda) en la petición, y devolver estos resultados o una representación de los mismos en la respuesta gestionando a su vez la base de datos donde se almacenan dichos recursos.

Todos estos valores son enviados con el formato JSON (o XML) debido a la problemática que pueden provocar las herramientas de cache web y los motores de búsqueda realizando cambios no intencionales en el servidor si estos valores fueran enviados de cualquier manera.

4.4.1.1.3 Rest y Jersey en Java

Java define el estandar Rest a través de JAX-RS (La api de Java para los servicios web RESTful) en JSR 311. JAX-RS utiliza anotaciones para definir la relevancia de clases REST.

Utilizando el "web.xml" se registra un servlet provisto mediante Jersey y también define la ruta bajo el cual la aplicación estará disponible:

http://your_domain:port/display-name/url-pattern/path_from_rest_class

Este servlet que analiza la solicitud HTTP entrante y selecciona la clase correcta y el método para responder a esta solicitud. Esta selección se basa en la anotación en la clase y los métodos.

Las anotaciones mas importantes en JAX-RS y que utilizaremos en el proyecto para generar nuestro servicio web REST son:

@PATH(your_path) → Establece la ruta de la URL base. La dirección URL base se basa en el nombre de la aplicación, servlet y modelo de dirección URL del archivo de configuración web.xml.

@POST → Indica que el método responderá a una petición POST HTTP.

@GET → Indica que el método responderá a una petición GET HTTP.

@PUT → Indica que el método responderá a una petición PUT HTTP.

@DELETE → Indica que el método responderá a una petición DELETE HTTP.

@Produces(MediaType.TEXT_PLAIN [, more-types]) → Define que tipo de MIME es entregado por un método anotado. En el ejemplo se produce "texto plano". En nuestro caso se utiliza también "application/xml" y "application/json".

@Consumes(type [, more-types]) → Define que tipo de MIME es consumida por este método.

@PathParam → Es usado para inyectar valores de la URL en un parámetro del método. En el caso del proyecto es utilizado para obtener el recurso mediante su id.

Para esta especificación utilizaremos la librería Jersey el cual contiene un servidor y un cliente REST, pero en el proyecto se desarrolla un servidor Jersey y un cliente adaptado a aplicaciones móviles llamado Spring.

4.4.1.1.4 Instalación

Durante la implementación del servicio web REST se han utilizado diferentes librerías que han sido agregadas al proyecto, entre ellas cuentas las librerías **Spring** (framework de código abierto para el desarrollo de aplicaciones Java), **Jersey** (framework para el desarrollo del servicio web REST), **Json** (librería para la utilización del formato ligero de intercambio de datos que recibe su nombre), **postgresql** (librería java para la conexión del servicio web REST con la base de datos del Consorcio postgreSQL) y **mysql-connector** (librería java para la conexión del servicio web REST con la base de datos del Consorcio MySQL).

Además se genero un servlet Apache Tomcat en el entorno de desarrollo de pruebas Eclipse antes de ser subido a producción. También se barajo la posibilidad de la alternativa de Google App Engine para "correr" el servidor en la red, pero no se vio la necesidad de ello.

4.4.1.1.5 Creación del servicio web Rest

Con Eclipse se crea un nuevo proyecto llamado “Dynamic Web Proyect” bajo el servlet Apache Tomcat instalado previamente, en el cual se depura el servicio hasta que se pueda subir el definitivo a producción en un servidor Apache del Consorcio.

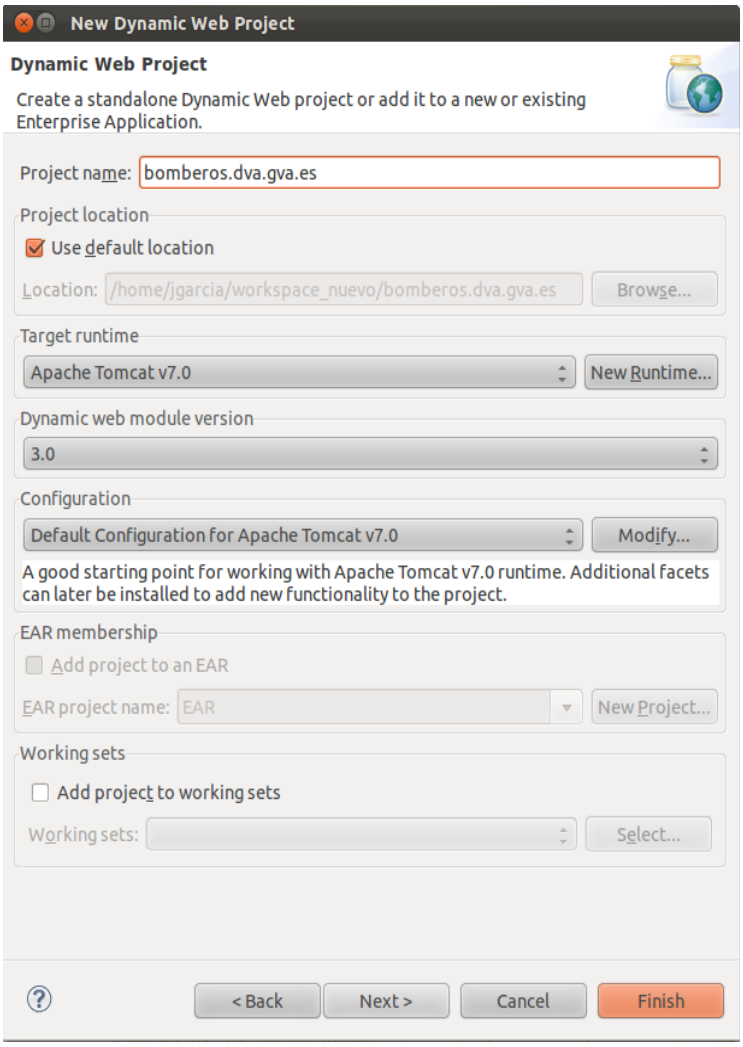

Figura 22: Pantalla instalación Servicio Web

Seguidamente, se copian todos los jars que se ha comentado con anterioridad y que son necesarios para implementar el servicio web en el apartado “WEB-INF/lib”.

4.4.1.1.6 Arquitectura servicio web RESTful

En esta sección se crea un CRUD (Crear, Leer, Actualizar, Borrar) para cada

uno de los datos que queremos gestionar del servicio web. Esto permite mantener un listado de cada orden de trabajo, activo... que se requiera en la aplicación móvil.

En primer lugar se modifica en el "web.xml" los siguientes parámetros:

```
<display-name>bombers.dva.gva.es</display-name>
<param-value>bombers.dva.gva.es.resources</param-value>
<url-pattern>/rest/*</url-pattern>
```

Con este resultado hemos modificado la ruta bajo la cual el dispositivo móvil podrá acceder a los diferentes elementos necesarios, y además se ha indicado mediante <param-value> donde residen el modelo de los datos.

A continuación se pasa a la creación de los diferentes modelos mediante el llamado "Singleton" (clase para la cual sólo una instancia puede ser creada y que proporciona un acceso global a esta instancia) que sirve como el proveedor de datos para el modelo. Se ha utilizado la implementación basado en un enumerado.

```
public class Singleton {
        private static Singleton instance = new Singleton();
private Singleton() {
}
public static synchronized Singleton getInstance() {
        if(instance == null)
                instance = new Singleton();
        return instance;
        }
}
```

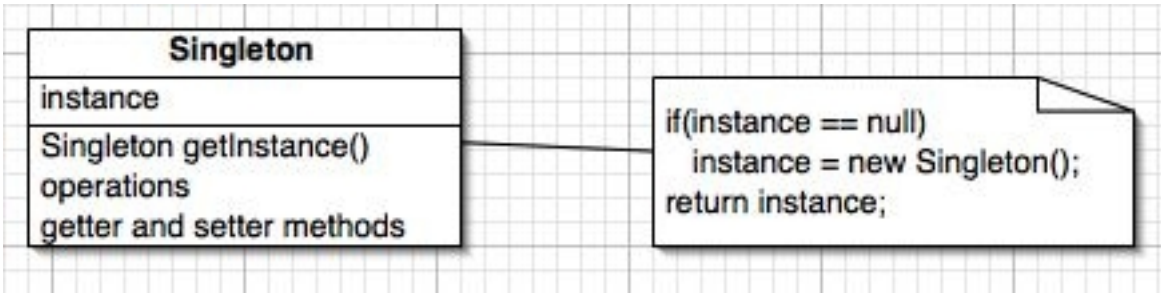

Figura 23: Diagrama de Clase "Singleton"

Por otro lado, la distribución se ha realizado en cuatro paquetes para que su estructura sea mas sencilla e intuitiva:

- Dao → En este paquete se crea una clase para cada dato a gestionar. Su función es la programación intermedia entre los datos recibidos de la aplicación Android con la base de datos correspondiente.
- Model → En este paquete se implementa una clase para cada dato y se crean sus variables.
- Mysql → En este paquete se diseña el CRUD con las bases de datos (MySQL y PostgreSQL), se gestiona la conexión y las consultas con estas.
- Resources → En este paquete se crea una clase para cada dato y otra clase para el listado de dicho dato. Con esto, podemos obtener el listado de todos los datos para su consulta, o individualmente podemos realizar la gestión de un dato en concreto.

4.4.1.1.7 Implementando el CRUD de Restful

En el apartado "Resources" citado anteriormente se crea una clase donde se gestiona el listado de los datos llamada "ListadoDatoResource". En esta clase se podrá realizar el listado de todos los datos mediante GET:

```
// Devuelve la lista de Activos almacenados en la BD
@GET
@Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
public List<Bienes> getBienes() {
      List<Bienes> Bienes = new ArrayList<Bienes>();
      Bienes.addAll( BienesDao.instance.getModel().values() );
      return Bienes;
}
```

En esta función se observa que produce elementos tanto xml como json que después podrán ser utilizados para su gestión. Por otro lado, la función realiza la tarea de crear un listado de Activos añadiéndolos a un List desde su instancia de clase Singleton.

Por otro lado, en esta clase también se puede añadir un dato a la base de datos mediante el método POST:

```
@POST
@Path("addNew")
@Consumes(MediaType.TEXT_PLAIN)
@Produces(MediaType.APPLICATION_JSON)
public String postClichedMessage(String message)  {
      JSONObject json = null;
      try {
            json = new JSONObject(message);
      } catch (JSONException e1) {
            e1.printStackTrace();
      }
      BienesDao.PostBienes(json);
      return "Orden de Trabajo guardada";

}
```

Esta función se encarga de recibir una cadena y convertirla al formato ligero json, tras esto, hace la llamada a la base de datos para guardar el activo recibido, además de añadirse en la clase Singleton del servicio web.

```
@Path("{ordenTrabajo}")
public OrdenTrabajoResource getOrdenTrabajo(
            @PathParam("ordenTrabajo") String id) {
      return new OrdenTrabajoResource(uriInfo, request, id);
}
```

Además se realiza las funciones de cuenta de los datos y la llamada a la clase "DatoResource" donde se gestiona cada dato de forma individual mediante un identificador único, es decir, para una orden de trabajo con id = 2, la dirección URL de consulta seria: http://192.168.100.143:8080/bombers.dva.gva.es/rest/orden_trabajo/2

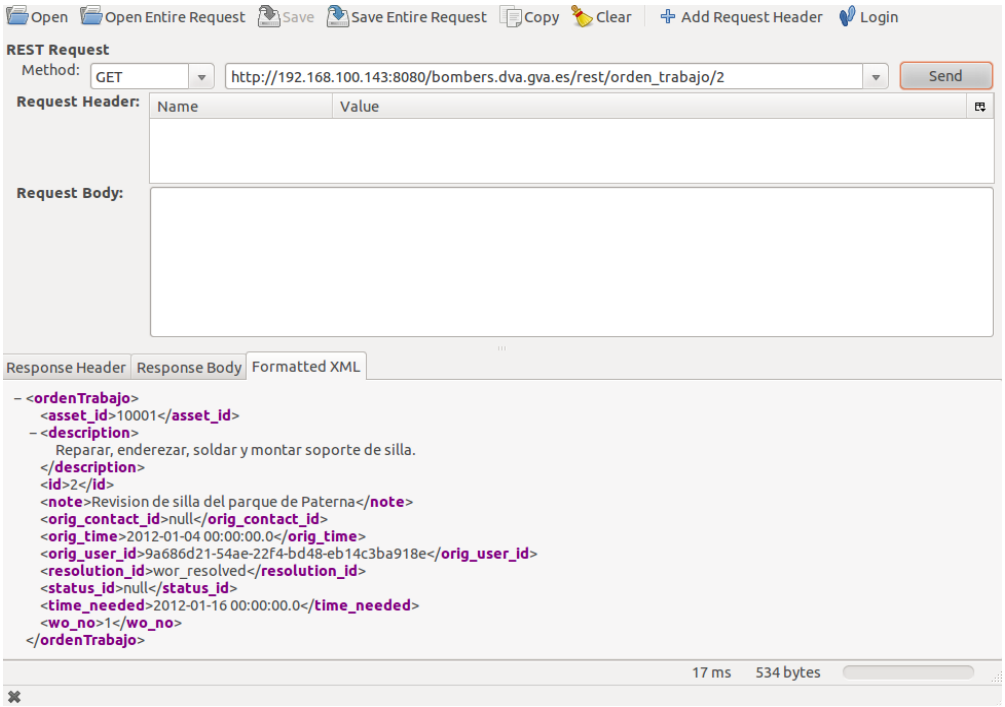

Figura 24: Consulta GET con Rest Client de Firefox

A continuación se implementará la parte donde se gestiona cada dato del servicio web de forma individual y filtrado por el identificador único, Esta clase "DatoResource" tendrá un método GET similar a la mostrada anteriormente pero en este caso tan solo mostrará un solo dato.

Finalmente, en esta misma clase se realiza una método PUT para la actualización del dato que solicita la aplicación y un método DELETE para el borrado de dicho dato.

Una vez se ha implementado las dos clases para cada datos, una para el listado y otra para cada dato individualmente, se puede probar la aplicación, se compila en Eclipse y se ejecuta en el navegador las URL para mostrar los items:

- http://localhost:8080/bombers.dva.gva.es/rest/orden_trabajo/ → Muestra un listado en formato xml de todos los datos de la clase "orden de trabajo".
- http://localhost:8080/bombers.dva.gva.es/rest/orden_trabajo/{id} → Muestra una "orden de trabajo", si el {id} existe.
- http://localhost:8080/bombers.dva.gva.es/rest/orden_trabajo/count → Muestra el numero de "ordenes de trabajo" existentes.

4.4.1.2 Programación con Android

4.4.1.2.1 Introducción a Android

En este apartado se van a realizar una breve introducción acerca de la programación de Android con el entorno de desarrollo de Eclipse. **Android** es un sistema operativo para dispositivos móviles basado en Linux. Ha sido desarrollado por la Open Handset Alliance, liderada por Google. Fue implementado por Android inc. Una firma comprada por Google en 2005 y que en la actualidad alcanza una cuota de mercado del 50,9 % durante el cuarto trimestre de 2011, mas del doble que el segundo sistema operativo (iOS de Iphone).

A causa de su código abierto, Android tiene una gran comunidad de desarrolladores para extender la funcionalidad de sus dispositivos. A la fecha se han sobrepasado las 400.000 aplicaciones disponibles en la tienda oficial: Google Play.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo

de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (surface manager), un framework OpenCore, una base de datos relacional SQLite, una API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, SSL y una biblioteca estándar de C Bionic.

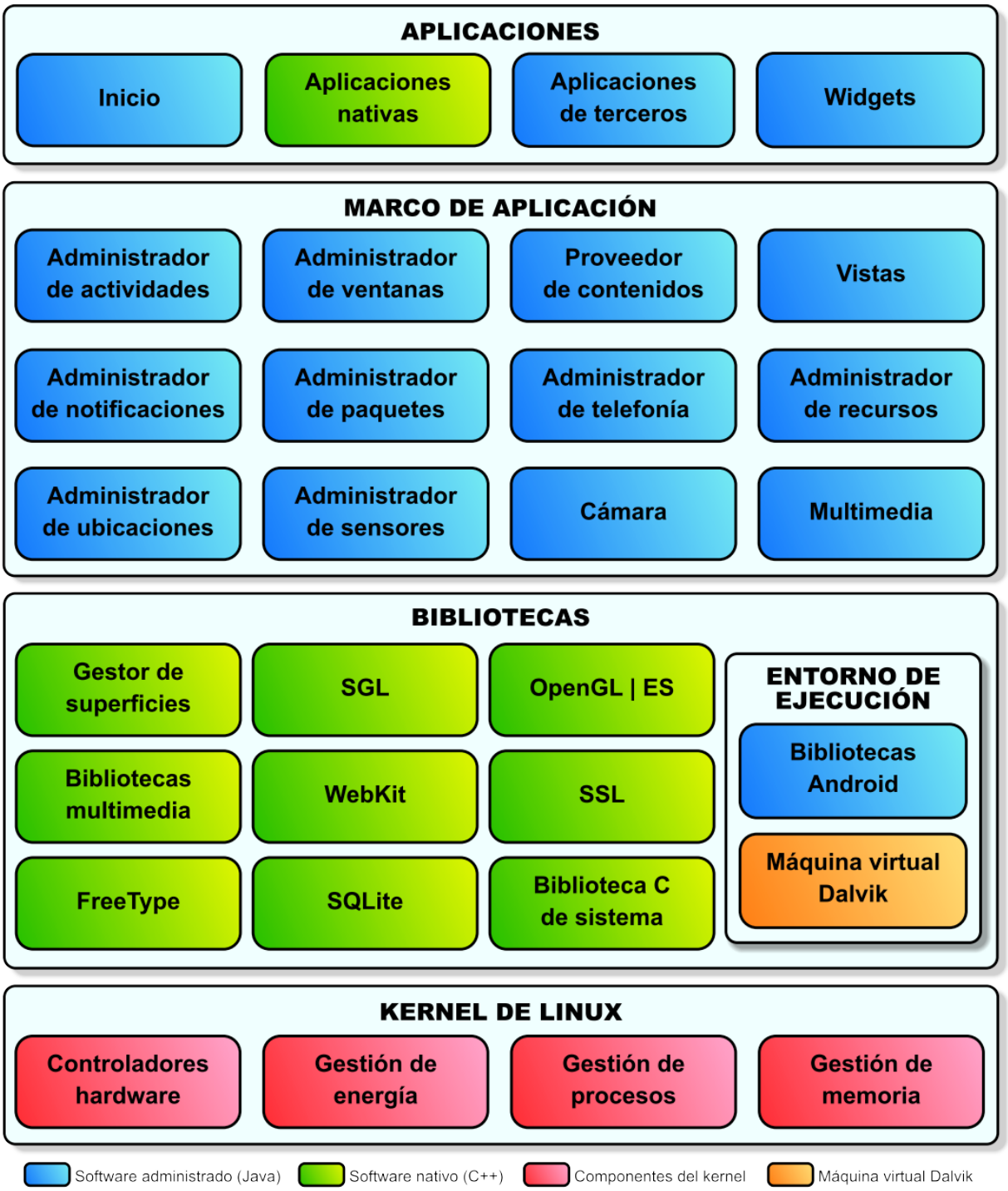

Figura 25: Arquitectura Android

En la figura anterior se muestra la arquitectura principal del sistema operativo Android y que se pasa a comentar de forma mas detallada a continuación:

- Aplicaciones: Conjunto de programas desarrollados que incluyen cliente de correo, proveedor de SMS, calendarios, ...además de todas las aplicaciones disponibles en Google Play.
- Marco de aplicación: Los desarrolladores tienen acceso total a los APIS usados por la aplicación base. La arquitectura se basa en la reutilización de componentes.
- Bibliotecas: Conjunto de librerías en C/C++ usadas por varios componentes del sistema.
- Entorno de ejecución: Set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente.
- Kernel de Linux: Android depende de Linux para los servicios base tales como la seguridad, gestión de energía, gestión de procesos...

Android ha tenido numerosas actualizaciones desde su liberación. Comentar la última versión 4.0 (Ice Cream Sandwich) que salio hará unos pocos meses al mercado y que incluye entre sus principales ventajas:

- Unificación del uso en cualquier dispositivo móvil.
- Interfaz renovada y mas moderna con la nueva fuente llamada “Roboto”.
- Aceleración por hardware. Interfaz manejable desde la GPU.
- Posibilidad de finalizar una tarea desplazándola fuera de la lista.
- Android Beam, que nos ofrece la posibilidad de compartir contenido entre telefonos.
- Reconocimiento de voz del usuario.
- Reconocimiento facial, para cambiar vistas.
- Soporte nativo para el uso tanto del contenedor MVK como del lápiz táctil.

4.4.1.2.2 Entorno de desarrollo Android

En el siguiente capitulo se detalla los pasos a seguir para desarrollar aplicaciones así como todo lo necesario para la instalación de todos sus componentes.

En primer lugar se necesita descargar Eclipse, en este proyecto se utiliza la versión "Indigo" 3.7 de Eclipse for Java Developers, para el sistema operativo apropiado. Tras su instalación se descarga al plugin SDK de Android con su última versión, además necesitamos el plugin de Android para Eclipse llamado *Android Development Tools* (ADT) y que se descargará desde Eclipse, accediendo al menu "*Help/ Install New Software...*" e indicando la URL: "https://dl ssl.google.com/android/eclipse/"

Se debe seleccionar e instalar el paquete completo Developer Tools, formado por Android DDMS y Android Development Tools..

Seguidamente se debe acceder a la sección de Android e indicar la ruta en la que se ha instalado el SDK: "*home/jgarcia/Desarrollo/android-sdk-linux*"

A continuación se deben descarga los targets necesarios que no son mas que las librerías necesarias para desarrollar en cada una de las versiones de Android. Para ello, se debe, desde Eclipse acceder al menú "*Window / Android SDK and AVD Manager*", y en la sección Avaliable Packages seleccionar los paquetes deseados:

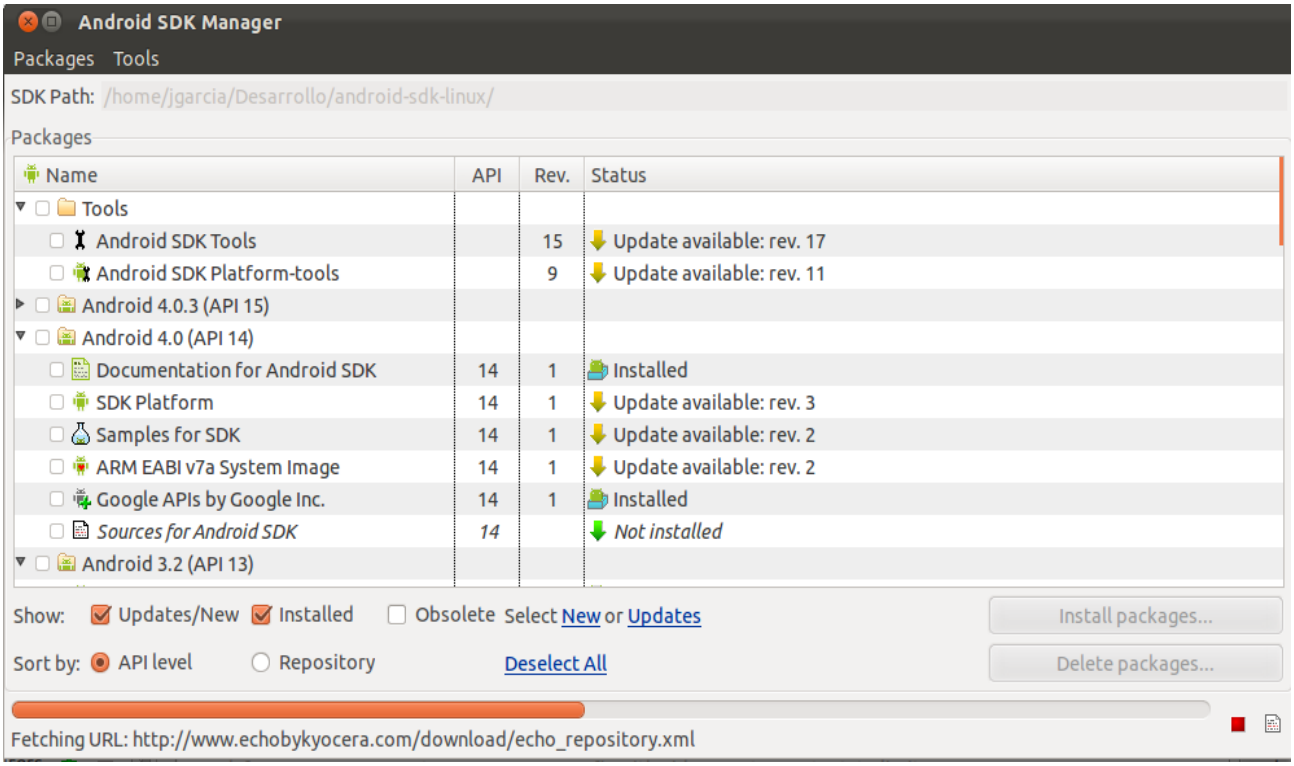

Figura 26: Android SDK Manager

Por otro lado, para probar y depurar las aplicaciones Android no tendremos que hacerlo siempre en el dipositivo físico sino que disponemos también de un emulador llamado Android Virtual Device (AVD). Para ello, se accede al AVD Manager y en Virtual Devices se añaden tantos emuladores como se quiera. Para configurar el AVD tan sólo tendremos que indicar un nombre descriptivo, el target de Android que utilizará, y las características de hardware del dispositivo virtual, como por ejemplo su resolución de pantalla, el tamaño de la tarjeta SD, o la disponibilidad de GPS.

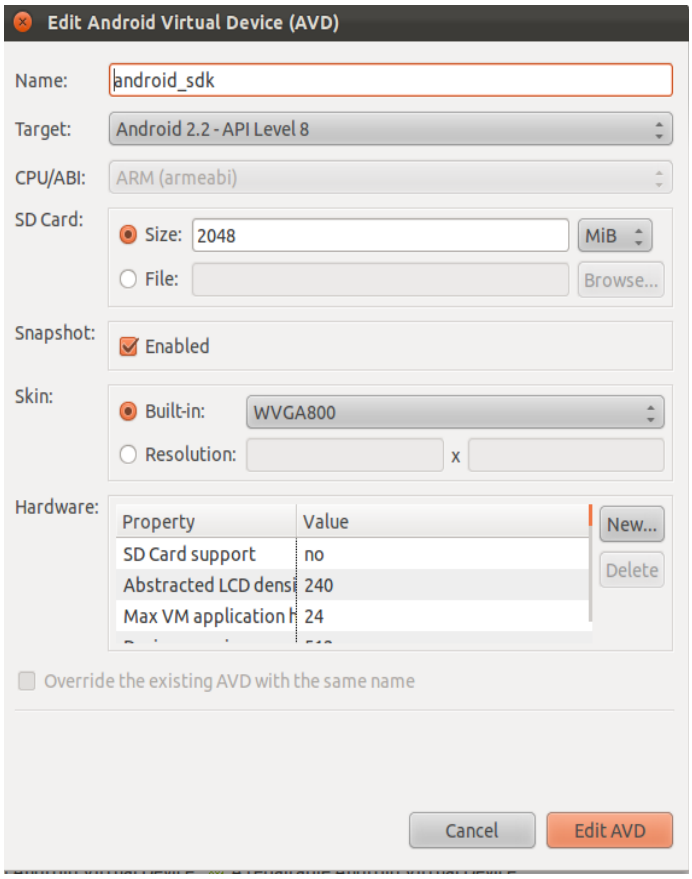

Figura 27: Emulador Dispositivo

Finalmente, creamos un nuevo proyecto de tipo *Android Project*. Indicamos su nombre, el target deseado, el nombre de la aplicación, el paquete java por defecto para nuestras clases y el nombre de la clase (*Activity*) principal.

Podemos emular un proyecto configurando una nueva entrada tipo *Android Applications* en la ventana de *Run Configurations*. Al ejecutar el proyecto, se abrirá un nuevo emulador Android creado previamente y se cargará automáticamente nuestra aplicación.

4.4.1.2.3 Estructura de un proyecto Android

Tras la creación de un proyecto Android se genera automáticamente la estructura de carpetas para poder generar la aplicación. Esta estructura será común a cualquiera aplicación Android:

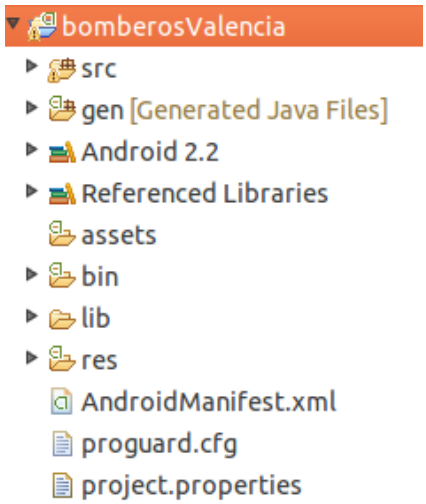

Figura 28: Estructura Proyecto Android

- bomberosValencia/src/ → Contiene todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, conexión con el servicio web REST, clases, manejo de errores, etc. Eclipse, inicialmente, creará el código básico de la pantalla principal de la aplicación. El proyecto contendrá la siguiente estructura dentro de esta carpeta:
  - Acceso a Datos → En este paquete se crea una clase para cada dato a gestionar. Su función es la programación intermedia entre los datos de la aplicación Android con el servicio web REST.
  - Autenticación → En este paquete se genera la autenticación IMAP.
  - Bienes → En este paquete se gestionan los activos.
  - Clases → En este paquete se implementa una clase para cada dato y otra para el listado de cada dato y se crean sus variables.
  - Manejo de Errores → En este paquete gestionan los mensajes de error.
  - Orden de Trabajo → En este paquete se gestionan las tareas a realizar.
  - QuickActions → En este paquete se diseñan las quickActions que explicaremos mas adelante.
  - Selección Parque → En este paquete se gestiona la selección del parque.
- bomberosValencia/res/ → Contiene todos los recursos necesarios para el

proyecto que deberán dividirse entre las siguientes carpetas:

- /res/drawable/. Contienen las imágenes de la aplicación. Se puede dividir en /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para utilizar diferentes recursos dependiendo de la resolución del dispositivo.
- /res/layout/. Contienen los ficheros de definición de las diferentes pantallas de la interfaz gráfica.
- /res/anim/. Contiene la definición de las animaciones utilizadas por la aplicación.
- /res/menu/. Contiene la definición de los menús de la aplicación.
- /res/values/. Contiene otros recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.

- bomberosValencia/gen/ → Contiene una serie de elementos de código generados automáticamente al compilar el proyecto. No debe ser modificada puesto que se actualiza sola cada vez que se haga algún cambio dentro de la carpeta res. Sirve, por lo tanto, como interfaz entre la carpeta res y el código fuente contenido en src.

Por último, cabe destacar el fichero AndroidManifest.xml, que contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono, …), sus componentes (pantallas, mensajes, etc), o los permisos necesarios para su ejecución.

En este fichero se deben definir todos los permisos necesarios para la aplicación, que le serán mostrados al usuario en el momento de la instalación:

```
<uses-permission android:name="android.permission.INTERNET" >
```

También deben especificarse todos los nombres de las actividades que aparecen en la aplicación Android, si ninguna de estas cosas se definen, la ejecución mostrará una excepción saliendo del programa:

```
<activity
        android:label="@string/app_name"
        android:name=".BomberosActivity" >
</activity>
```

Finalmente, se puede definir que aplicación será la primera a mostrar:

```
<intent-filter >
        <action android:name="android.intent.action.MAIN" />
         <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

4.4.1.2.4 Componentes de una aplicación Android

En este apartado se muestra los distintos tipos de componentes con los que se construye el proyecto Android.

- Activity → Representan el componente principal de la interfaz gráfica de una aplicación Android. Crea la IGU de la aplicación, una de ella será la principal, el cambio entre Activitys se realiza mediante Intents.
- Intent → Elemento básico de comunicación entre los distintos componentes Android. Se pueden entender como mensajes o peticiones enviados entre los distintos componentes de la aplicación o entre distintas aplicaciones.
- View → Son los componentes básicos con lo que se ha construido la interfaz del proyecto y que se comentaran en apartados posteriores.

Ciclo de Vida: Cuando una se lanza pasa a ser la cima de la pila. La cima de la pila anterior pasa a segundo plano hasta que la nueva Activity termine. Cuando el usuario pulsa el botón atrás, la actividad en segundo plano pasa a la cima y se activa.

Estados:

- **Ejecución**: En primer plano, interactúa con el usuario.

- **En pausa**: Visible, pero no en la cima, no interactúa con el usuario.

- **Parada**: Completamente oculta, mantiene su estado, será matada por el sistema cuando se necesite algo de memoria.

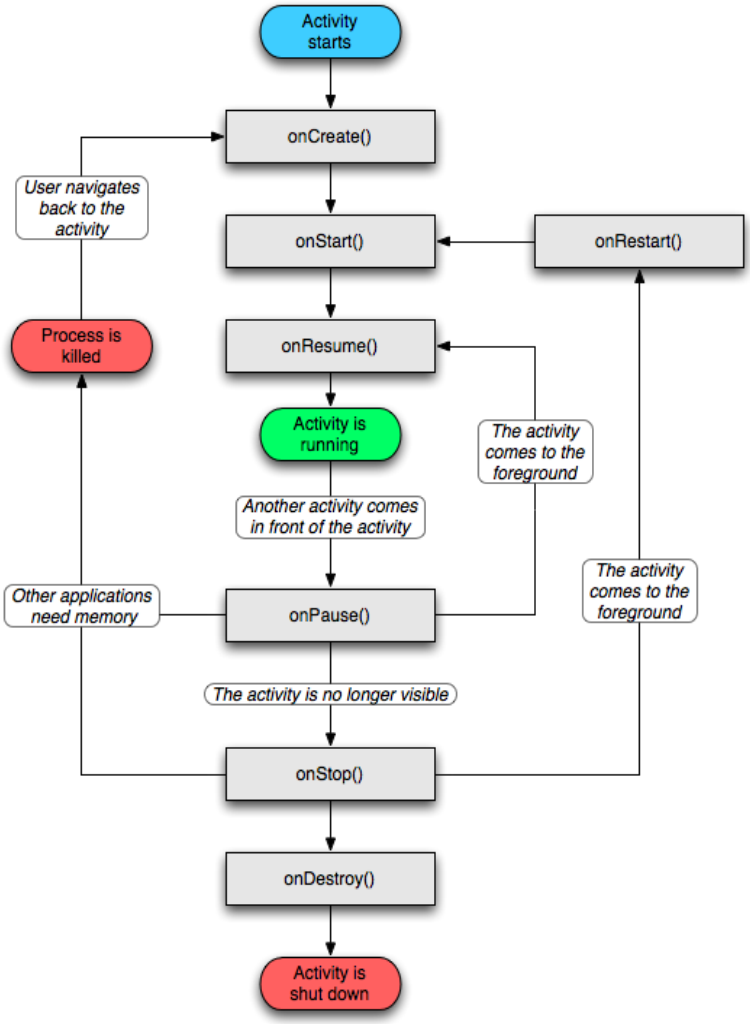

Figura 29: Ciclo Vida Android

4.4.1.3 Conexión cliente Android - servicio Web

En este apartado se van a realizar una breve descripción de la conexión entre el cliente Android y el servicio web REST, como ya se ha comentado las llamadas al servicio no se harán a través de una única URL, sino que se determinara la acción a realizar según la URL accedida y la acción HTTP utilizada para realizar la petición (GET, POST, PUT, DELETE).

El cliente Android ha sido diseñado mediante Spring, utilizando el cliente de Spring Rest llamado RestTemplate, y se han implementado múltiples representaciones que utilizan HttpMessageConverter para comunicarse con los servicios web REST. Por otra parte, la conexión del servicio web ya ha sido comentado en un apartado anterior.

Tanto solicitudes como respuestas en HTTP están basadas en texto brutos comunicados mediante un navegador y su servidor, que gracias a HttpMessageConverter convertiremos en objetos. Hay varios ejemplos de este tipo, pero en este proyecto se ha utilizado MappingJacksonHttpMessageConverter debido a que es capaz de leer/escribir en formato JSON usando "ObjectMapper Jackson ". Convierte datos a Mime Type application/json.

Las funciones que se utilizaran para la implementación CRUD en la parte cliente y así, poder comunicarse serán:

- getForObject → Ejecuta el método HTTP GET y obtiene la respuesta como un objeto.
- PostForObject → Ejecuta el método HTTP POST con un cuerpo de la solicitud de tipo JSON.
- Put → Ejecuta el método HTTP PUT con un cuerpo de la solicitud JSON.
- Delete → Ejecuta el método HTTP DELETE para un URI determinado.

4.4.1.4 Conexión servicio Web – Base de Datos

El objetivo de esta sección es la creación de conexiones con Java para las bases de datos del Consorcio, tanto para MySQL con la librería mysql-connector como para postgreSQL con la librería postgresql.

En primer lugar, lo que debemos hacer es instanciar la clase del driver:

- Class.*forName*("com.mysql.jdbc.Driver");
- Class.*forName*("org.postgresql.Driver");

Las tres clases que se manejaran en el servicio web serán Connection para realizar la conexión a la base de datos tanto para la conexión con MySQL como en postgreSQL:

- c = DriverManager.*getConnection*("jdbc:postgresql://servidor/base_datos","usuario", "contraseña");
- c = DriverManager.*getConnection*("jdbc:mysql://servidor/base_datos?""user=usuario&password=contraseña");

Statement que será la que contenga la sentencia SQL:

- statement = c.createStatement();

y ResultSet que será la que contenga el resultado:

- resultSet = statement.executeQuery("select * from workorder ORDER BY orig_time ASC");

Una vez tengamos el contenido en ResultSet ya se es capaz de poder trabajar sobre dichos datos en el servicio web para su futura gestión.

4.4.1.5 Interfaz gráfica Android

Para continuar con la primera iteración se detalla los elementos que se pueden mostrar por pantalla con Android, además se realiza una primera consulta para comprobar que los datos se muestren correctamente por pantalla.

Todo objeto que queremos que se muestre por pantalla debe heredar de la clase View descrita anteriormente. Podemos definir nuestras propias clases que hereden de View o utilizar las clases predeterminadas. Estos elementos se pueden organizar y ordenar dentro de objetos Layout.

Los layouts son elementos no visuales destinados a controlar la distribución, posición y dimensiones que se insertan en su interior, entre ellos:

- LinearLayout: Apila uno tras otro todos sus elementos hijos de forma horizontal o vertical según se establezca su prioridad. También se puede establecer sus propiedades width y height para determinar sus dimensiones.
- FrameLayout: Coloca todos sus controles hijos alineándolos con su esquina superior izquierda. Suele utilizarse para mostrar un solo control.
- TableLayout: Permite distribuir sus elementos hijos de forma tabular, definiendo filas y columnas necesarias, y la posición de cada componente

dentro de cada tabla.

- RelativeLayout: Permite especificar la posición de cada elemento de forma relativa a su elemento padre o a cualquier otro elemento incluido en el propio layout.

Una vez tengamos los layouts definidos en la clase View, podemos añadir los diferentes controles que nos ofrece Android para el diseño de interfaces y que, a continuación, se comentan los mas utilizados en este proyecto:

- Button: Elemento que dibuja un botón para la interacción con el usuario. Entre sus características esta la capacidad de añadir texto, cambiar de color... Una propiedad a destacar es onClick, que puede generar un evento cuando el usuario pulsa el botón.
- ImageButton: Control que define una imagen a mostrar en vez de un texto, para lo que se debe definir un "src" (donde esta almacenada la imagen).
- ImageView: Control que permite mostrar imágenes.
- TextView: etiquetas para mostrar texto al usuario, se puede dar el formato deseado mediante su propiedades.
- EditText: Componente de edición de texto.

Finalmente se detallan los controles de selección utilizados en este proyecto y como aplicar los adaptadores que son comunes a todos ellos:

- Adaptadores: Representa el modelo de datos donde accederán a los datos que contienen a través de un adaptador.

```
final String[] datos =new String[]{"Elem1","Elem2","Elem3"};


ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
                                        android.R.layout.simple_spinner_item, datos);
```

- Spinner: Listas desplegables de Android, mostrando una especie de lista emergente al usuario con todas las opciones disponibles y al seleccionarse una de éstas se queda fijado el control. Para enlazar un adaptador:

```
final Spinner Opciones = (Spinner)findViewById(R.id.Opciones);
```

```
adaptador.setDropDownViewResource
        ( android.R.layout.simple_spinner_dropdown_item);
Opciones.setAdapter(adaptador);
```

- ListView: Muestra al usuario una lista de opciones seleccionables directamente sobre el propio control, sin listas emergentes.

Finalmente y para entender completamente el proyecto, se explica el ArrayAdapter diseñado para el proyecto:

```
class IconAdapter extends ArrayAdapter<String> {
        IconAdapter() {
                super(Activity.this, R.layout.listitem, datos);
        }
        public View getView(int position, View convertView, ViewGroup parent) {
           View row = convertView;
           if (row == null) {
                row = super.getView(position, convertView, parent);
           }
           TextView firsLine = (TextView)row.findViewById(R.id.firstLine);
           firstLine.setText(datos[position].getTitulo());

           TextView secondLine = (TextView)row.findViewById(R.id.secondLine);
           secondLine.setText(datos[position].getSubtitulo());
           return(row);
        }
}
```

Primero se implementa el constructor para nuestro adaptador al que se le pasa la actividad que será el contexto del listado.

Posteriormente, se redefine el método encargado de generar y rellenar con los datos todos los controles necesarios de la interfaz gráfica de cada elemento de la lista.

El método getView() se llamará cada vez que haya que mostrar un elemento de la lista. Primeramente, se crea e inicializa una fila de nuestro layout XML.

Tras esto, tan sólo se tendrá que obtener la referencia a cada una de las etiquetas como siempre y asignar su texto correspondiente según los datos del array y la posición del elemento actual (parámetro position del método getView()).

4.4.1.5.1 Listado Consulta

Una vez se ha explicado todos los componentes que se van a utilizar en el proyecto, ya se es capaz de definir un listado de consulta, mediante el adaptador "IconAdapter" y con la conexión entre la base de datos, servicio web y aplicación Android que se ha explicado anteriormente, se obtiene el siguiente resultado:

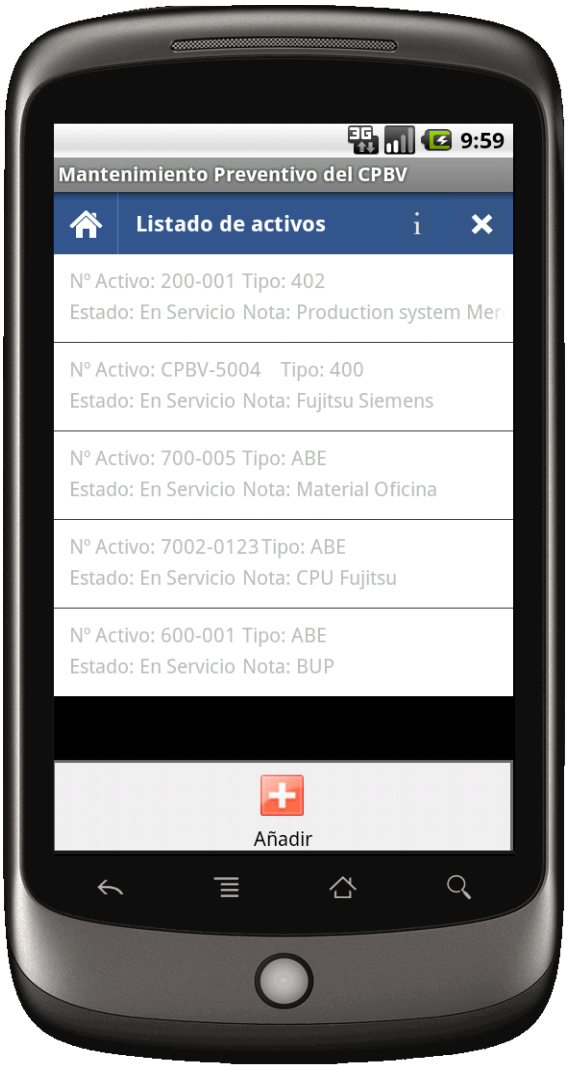

Figura 30: Listado consulta activos

4.4.2 Segunda Iteración

Tras la aceptación de los bocetos por parte del cliente, en este apartado se va continuar con la fase de construcción, en concreto, con la segunda iteración del proyecto.

En esta segunda fase se termina la interfaz gráfica, que después se desglosará en el apartado del manual de usuario e instalación.

4.4.2.1 Clases de la aplicación

Se añaden todas las clases en el servicio web REST que se utilizaran para gestionar el mantenimiento, cabe destacar:

- Activo: Todos aquellos equipos y vehículos del Consorcio y que se hace inventario según el parque al que pertenezcan.
- Parque: Los parques de Bomberos del Consorcio Provincial de Bomberos de Valencia incluyendo Central.
- Orden de Trabajo: Todas aquellas tareas de mantenimiento que se definen sobre un activo de un parque.
- Usuario: El personal operativo que se encargar de realizar las tareas de mantenimiento.

4.4.2.2 Gestión de activos y tareas

Este apartado se explica con mas detalle en el manual de usuario donde se podrá visualizar las interfaces utilizadas para su gestión, en cambio, en esta sección se explica la implementación de los **QuickActions** en el listado de activos y tareas, que permite un uso mucho mas intuitivo, al pulsar sobre cualquier dato del listado, aparece un menú que muestra las opciones que se tienen sobre ese dato en forma de listado.

QuickActions no están incluidos en el plugin estándar de Android SDK, por lo que hay que crearlos manualmente. Se ha utilizado un layout simple y plano basado en el QuickActions de contactos o de twitter de Android.

En primer lugar, se crean los "action items" que son todas aquellas opciones que tendrá el menú QuickActions y se asocia un titulo y un icono para que sea mas visual:

```
ActionItem addItem =
new ActionItem(ID_EDIT, "Editar",
getResources().getDrawable( R.drawable.ic_up_azul));
```

A continuación, se implementa la instancia de QuickActions y se añaden los Items al QuickAction:

```
final QuickAction mQuickAction;
mQuickAction.addActionItem(addItem);
```

Seguidamente, se implementa el método que muestra que elemento de acción es pulsado:

Figura 31: QuickAction en Android

```
mQuickAction.setOnActionItemClickListener(new
        QuickAction.OnActionItemClickListener() {
        @Override
        public void onItemClick(QuickAction quickAction, int pos, int actionId) {
        }
});
```

Finalmente, se añade el código para mostrar el menú QuickActions cuando se pulsa la fila correspondiente:

```
row.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                mQuickAction.show(v);
                        }
});
```

4.4.2.3 Autenticación IMAP

Para continuar con esta segunda iteración se acuerda con el cliente realizar una pantalla de autenticación para permitir solo al personal operativo acceder a las tareas de mantenimiento del Consorcio, que consiste en la autenticación contra el correo de la compañía y una gestión de errores de login y contraseña. Para realizar esta implementación se necesita instalar la librería JavaMail.
JavaMail es una expansión de Java que facilita el envío y recepción de e-mail desde código Java.

En el proyecto tan solo se realiza el proceso de conexión con el servidor de correo mediante la autenticación y se consulta si se ha realizado la autenticación con éxito para permitir el acceso al personal autorizado.

```
Properties props = new Properties();

props.setProperty("mail.store.protocol", "imap");
props.setProperty("mail.imaps.host", "servidor_correo_bomberos");
props.setProperty("mail.imaps.port", "puerto_servidor_correo");
props.setProperty("mail.imaps.socketFactory.class",
                                    "javax.net.ssl.SSLSocketFactory");
props.setProperty("mail.imaps.socketFactory.fallback", "false");
Session imapSession = Session.getInstance(props);
Store store = imapSession.getStore("imap");
store.connect("servidor_correo_bomberos", usuario, contrasenya);
if (store.isConnected()) {
        conectado = true;
}
return conectado;
```

4.4.2.4 Almacenamiento en Android

En aplicaciones típicas de escritorio, el sistema operativo ofrece el sistema de ficheros para compartir datos entre aplicaciones, en cambio, en Android, los ficheros son privados por aplicación.
Para compartir información, se utilizan los Content Providers, que no es mas que un mecanismo que proporciona Android para compartir información entre aplicaciones.

Finalmente en la segunda iteración se describe las diferentes formas de almacenamiento de datos en Android y la gestión que se realiza para guardar los datos del usuario y su código de parque asociado mediante las preferencias en

Android: “SharedPreferences”.
Android ofrece varias posibilidades de almacenamiento de datos según que caso:

Preferences:

Es una técnica ágil para guardar datos simples de la aplicación, estos se almacenan en pares key/value y es usado típicamente para guardar las preferencias de la aplicación (fuentes, colores...). En el caso del proyecto, se utiliza para el almacenamiento del login del usuario y de su código de parque asociado:

```
final String MYPREFS = "MyPreferences_001";
SharedPreferences mySharedPreferences;
SharedPreferences.Editor myEditor;

public void onCreate(Bundle savedInstanceState) {
        mySharedPreferences = getSharedPreferences(MYPREFS, 0);
        myEditor = mySharedPreferences.edit();
        saveDetails(view)

}

private boolean saveDetails(View view) {
                myEditor.putString(“codigo_parque”, codigo_parque);
                myEditor.putString(“usuario”, usuario);
                myEditor.commit();
                return true;
}
```

De esta forma se guarda datos primitivos (Boolean, String, float...) y estos persisten aunque la aplicación muera.

Ficheros locales:

Acceso similar a Java estándar, se deben crear inputs y output streams, pero solo se soportan archivos que estén creados en la misma carpeta que la aplicación. Esta ruta es: /data/app (gratuitas) y /data/app-private (de pago):

```
String FILE_NAME = "tempfile.tmp";
FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE);
FileInputStream fis = openFileInput(FILE_NAME);
```

SQLite:

Es un base de datos Open Source que cumplimenta los estándares de Bds. Además no requiere excesivos recursos y las consultas se devuelven como objetos Cursor, apuntando a la información.
Cada base de datos es privada para la aplicación, pero pueden acceder todas las clases de ésta. Estas se almacenan en la carpeta /data/data/nombre_package/databases.

Servicio Web:

Otra forma de almacenamiento, pero en este caso externa, y que ya se ha comentado en capítulos anteriores es enviar la información a un servicio web para su almacenamiento a bases de datos externas y que generalmente sirve para cuando esta información tiene que ser gestionada de forma externa y además puede ser modificada por varios dispositivos móviles.

4.4.3 Tercera Iteración

Finalmente se completa el proyecto con una última iteración, donde se procederá con los detalles finales que se establecieron con el cliente de forma que se termine la aplicación y se puede pasar a la fase final de cierre o transición.

4.4.3.1 Selección de Parque

En este apartado se procede a la realización de un listado de parques con el adaptador detallado anteriormente “IconAdapter”. Este listado se encarga de mostrar el parque al que pertenece el personal autenticado, indicado en el servicio web, por otro lado, si el personal operativo esta asignado a central, el listado muestra todos los parques del Consorcio, por que los usuarios de central tienen permisos para hacer las revisiones de mantenimiento en cualquier parque del Consorcio.

4.4.3.1.1 Estado de Conexión

En la mayoría de ventanas de la aplicación Android se hace uso de conexión al servicio web y, por tanto, a la red del Consorcio. Por esto, es de especial interés gestionar debidamente el estadio de conexión con el servicio web y mostrar un dialogo de error en caso de que la comprobación sea errónea:

```
final AlertDialog dialog = new AlertDialog.Builder(this).create();
dialog.setTitle("Error");
dialog.setMessage("Imposible obtener la lista de Usuarios. \nRevise su conexión.");
dialog.setButton("OK", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int which) {
                // Tratamiento del dialogo
        }
});
// Obtenemos el personal
PersonalDAO PersonalDAO = new PersonalDAO(getApplicationContext());
ArrayList<String> items = PersonalDAO.descargarPersonal(usuario);
if (items == null) {
        // Si falla, mostramos la alerta de dialogo
        dialog.show();
        return false;
}
```

Una vez controlada la gestión de errores, se pasa a la comprobación de que el estado de la conexión sea valido, esto se realiza en el DAO y se comprueba si la conexión devuelve una excepción:

```
try {
        jsonResult = rest.getForObject(url, String.class, "usuarios");
} catch (RestClientException e2) {
        e2.printStackTrace();
        return null;
}
```

4.4.3.1.2 Diálogos

Dado que se ha hablado de diálogos en el apartado anterior se va a comentar y definir un poco sus utilidades en Android. Se trata de un mecanismo para mostrar o solicitar información puntual al usuario y que se podrán utilizar con distintos fines, en general:

- Mostrar un mensaje.
- Pedir una confirmación rápida.
- Solicitar una elección (simple o múltiple) entre varias alternativas.

4.4.3.2 Filtrado por Parque

En este apartado se describe como se ha implementado la lógica para filtrar tanto los activos como las ordenes de trabajo por el parque seleccionado previamente.

De forma básica y resumida, las operaciones que se realizan para tratar este filtrado consisten en la comprobación de que el campo del parque que tiene el activo almacenado corresponda al parque seleccionado por el personal operativo que esta autenticado en ese momento. Todo esto se realiza en el DAO de la aplicación Android en el proceso de descargar de datos del servicio web.

Por otra parte para gestionar el filtrado en las ordenes de trabajo, el proceso es bastante similar, con la única diferencia, que tan solo un activo corresponde a una orden de trabajo, y es el activo el que pertenece al parque correspondiente, por tanto, se tendrá que comparar el activo asociado a esta orden de trabajo con el parque seleccionado por el personal operativo que esta autenticado en ese momento.

Por último, cabe decir que si el personal operativo ha seleccionado como parque Central, tendrá los listado de activos y ordenes de trabajo completos de todos los parques, excepto aquellas ordenes de trabajo que se hayan cerrado.

4.4.3.3 Ordenación por Fecha

En esta sección se detalla como se ha diseñado para realizar la ordenación de las tareas por fecha de inicio, siendo las mas cercanas a la fecha actual, las que se tendrá en el principio del listado de ordenes de trabajo.

En primer lugar, en el servicio web se realiza una consulta a la base de datos donde se almacenan estas ordenes de trabajo, la cual se ordena mediante el tiempo de inicio, de esta forma se obtiene un servicio web con ordenes de trabajo ya ordenadas debidamente.

A continuación, en la aplicación Android, tan solo se tendrá que realizar una lectura del servicio web, y este, le mandará los datos de forma ordenada por fecha de inicio de la orden de trabajo.

4.4.3.4 Cierre de Tarea

A continuación se describe la implementación del cierre de tareas en la aplicación Android, que se producirá una vez se haya hecho el mantenimiento en un activo. En primer lugar, se debe seleccionar la orden de trabajo correspondiente y realizar la edición de la tarea que se esta comprobando.
Tras esto, el personal operativo seleccionara del listado de tareas, la tarea que desea cerrar y la aplicación le indicara mediante un dialogo las opciones de las que dispone para el cierre de la tarea correspondiente:

- Resuelta: Tarea finalizada con éxito.
- Sin resolver: No ha podido resolverse, pero se cierra por distintas causas.
- Duplicada: Dos usuarios distintos han creado dicha tarea.
- No procede: La tarea no se corresponde con lo que se debe hacer.

Una vez seleccionada la opción, esta se guardará en la base de datos, marcando la orden de trabajo como cerrada y, por tanto, esta tarea ya no se visualizará en el apartado de listado de tareas.

4.4.3.5 Historial de Revisiones

En esta sección se comenta la lógica utilizada para el diseño del historial de revisiones, que será de gran utilidad para cumplir con todas las fases del mantenimiento en una empresa.

En primer lugar, cuando se pulsa sobre el icono del historial de revisiones, este descarga del servicio web todos los activos del parque seleccionado.

A continuación, el usuario seleccionará un activo que desee revisar y le aparecerá el listado de tareas, tanto activas como cerradas que se han realizado sobre ese activo en concreto.

4.4.3.6 Visualizar PDF

En esta sección se detalla la lógica aplicada para realizar la visualización de la documentación y guías operativas para que el personal operativo pueda consultar y realizar las tareas de forma óptima.

```
if (pdfFile.exists()) {
        Uri path = Uri.fromFile(pdfFile);
        Intent pdfIntent = new Intent(Intent.ACTION_VIEW);
        pdfIntent.setDataAndType(path, "application/pdf");
        pdfIntent.setFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
        startActivity(pdfIntent);
}
```

5 Fase de Cierre

El propósito de esta fase es asegurar que el software esté disponible para los usuarios finales, ajustar los errores y defectos encontrados en las pruebas de aceptación, capacitar a los usuarios y proveer el soporte técnico necesario. Se debe verificar que el producto cumpla con las especificaciones entregadas en el proyecto.

5.1 Manual de Instalación

A continuación se procede a la descripción de la generación de un instalable con la extensión .apk para la posterior instalación en el dispositivo móvil sin tener que subir la aplicación a Google Play.

Para la creación del instalador, el proyecto debe ser firmado digitalmente para la posterior instalación en un dispositivo. Se puede realizar sin la necesidad de un certificado digital, sino directamente al exportar el proyecto a un Android Application se muestra un formulario para rellenar los campos que el desarrollador debe modificar para poner sus propios datos:

- Location KeyStore: Ruta para guardar el keystore.
- KeyStore Password: Contraseña para la seguridad del keystore.
- Alias: Identificador con el que se refieren las claves que se están creando.
- Key Password: Contraseña para acceder a las claves que se están creando.
- Certificate Validity: Periodo de tiempo de validez.

Tras rellenar estos campos, se solicita una serie de datos personales que el desarrollador opcionalmente puede rellenar para asociar con el alias que se ha creado.

Una vez se ha hecho esto, finalmente Eclipse nos indica donde queremos situar nuestro .apk y con esto se obtiene la aplicación lista para su instalación.

5.1.1 Instalación de Aplicaciones

Primeramente se tiene que obtener el .apk que queremos instalar en el dispositivo móvil, explicado en el apartado anterior. A continuación se copia este archivo por cable al dispositivo y en el menú ajustes vamos pulsando tal y como se ve en las siguientes imágenes:

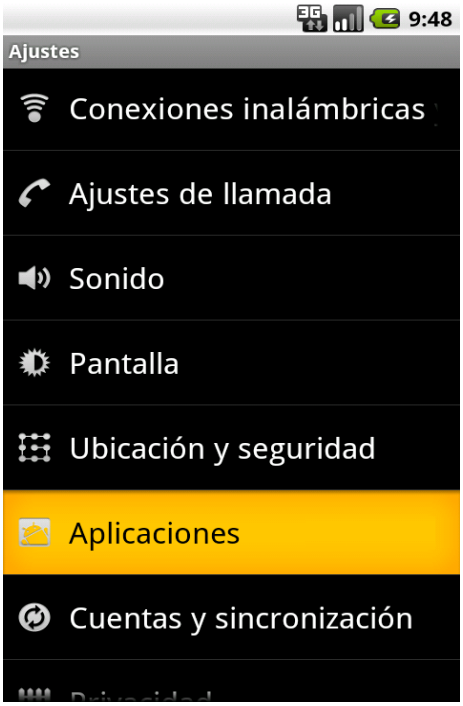

Figura 32: Configuración de aplicaciones

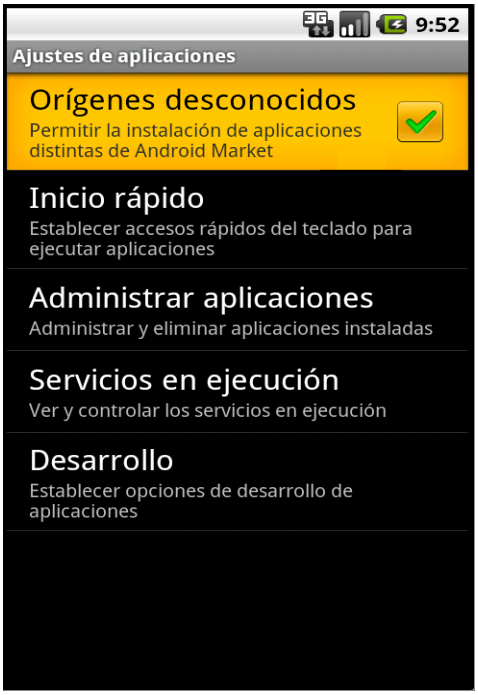

Figura 33: Activar Orígenes desconocidos

Finalmente sólo tenemos que ir con un navegador de archivos donde hubiéremos copiado el apk correspondiente y lo ejecutamos.

5.2 Manual de Usuario

Tras la generación del .apk y la configuración de dispositivos móviles para la instalación de aplicaciones que no se hayan descargado de Google Play, el personal operativo tan solo necesita una directrices para utilizar el sistema de mantenimiento de forma fluida.

5.2.1 Selección de la Aplicación

En primer lugar, se accede al listado de todas las aplicaciones que dispone nuestro dispositivo móvil, seguidamente, se selecciona la aplicación de mantenimiento del Consorcio Provincial de Bomberos de Valencia, aquella que dispone del logo del Consorcio como icono de la aplicación.

Figura 34: Selección de la aplicación

5.2.2 Autenticación de la Aplicación

A continuación, la aplicación nos muestra una interfaz de autenticación donde se debe introducir el usuario y contraseña para poder acceder al menú principal.

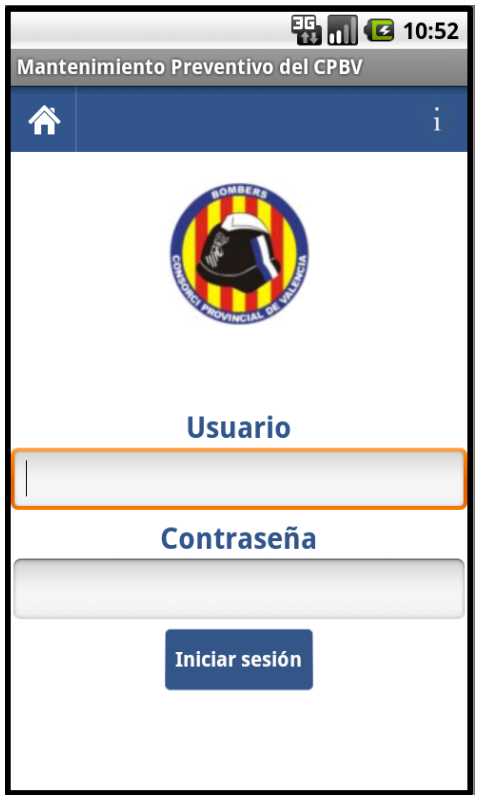

Figura 35: Autenticación en la aplicación

Cuando se haya introducido estos datos y al pulsar sobre el botón de "Iniciar Sesión" pueden ocurrir tres estado:

- Mostrar un mensaje de "Escribir usuario y/o contraseña", cuando el usuario no ha introducido ningún dato en los apartados correspondientes.
- Mostrar un mensaje de "El usuario o contraseña son incorrectos", cuando el usuario no ha introducido los datos correctamente o no tiene permisos para acceder a la aplicación.
- Acceso con éxito cuando el usuario se ha autenticado con éxito mediante el servidor de correo IMAP y se procede a la descarga de datos mediante el siguiente dialogo de progreso mostrado en la figura siguiente.

Finalmente, podemos pulsar sobre el icono de información que se encuentra arriba a la derecha, que muestra los pasos que se deben seguir en cada una de las pantallas de la aplicación.

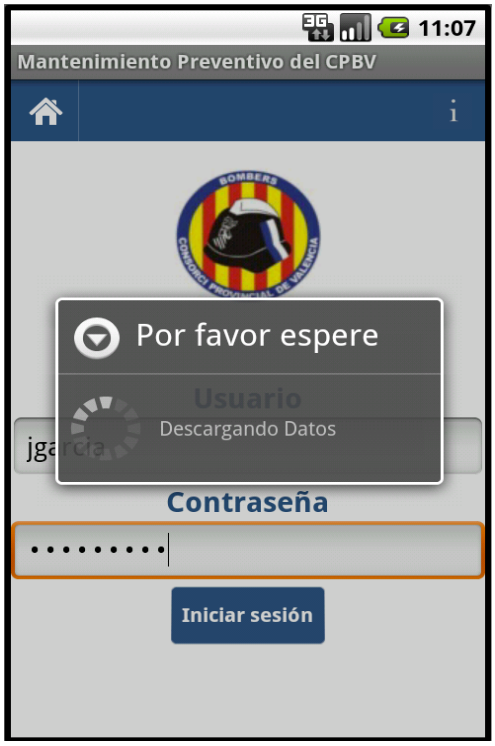

Figura 36: Dialogo descarga de datos

A continuación la aplicación muestra una pantalla donde se muestra un listado de los parques del Consorcio. Si el personal operativo esta asociado a central, se muestra el listado de todos los parques, en cambio, puede estar asociado a un parque en concreto, por lo que, por permisos en el Consorcio, solo se visualiza dicho parque:

Figura 37: Selección parque

5.2.3 Pantalla Principal

Este menú se divide en cuatro iconos principales para la gestión del mantenimiento, además contiene un icono de información de lo que realiza cada icono y una cruceta para realizar un logout en el programa:

Figura 38: Pantalla Principal

El primer icono a comentar es la que esta situada arriba a la derecha, esta pantalla nos permite cambiar de parque respecto al que habíamos seleccionado al principio, si el usuario es personal operativo asignado a central, sino tan solo saldrá el parque al que el usuario esta asignado. Esta pantalla es similar a la pantalla que se ha comentado anteriormente y que corresponde con la selección del parque.

5.2.4 Gestión de Activos

A continuación, se procede a la descripción de la gestión de activos, icono situado abajo a la izquierda y que tras su pulsación aparece un listado de todos los activos situados en el parque correspondiente, como se puede observar en la figura siguiente:

Figura 39: Listado de Activos

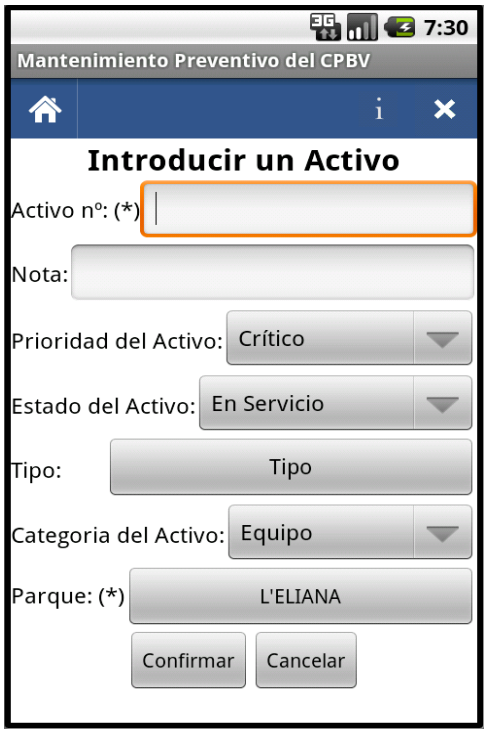

Figura 40: Añadir Activos

Tras obtener el listado de activos, se puede realizar la gestión completa del activo: añadir uno nuevo, editar y borrarlo como se puede observar en las figuras.
Tan solo detallar que en en el apartado del borrado de activos, este proceso mostrará un dialogo de error, si este activo contiene una orden de trabajo.
Por otro lado al editar un activo, aparecen dos flechas en la parte superior para poder moverse en el listado de una forma visual e interactiva.

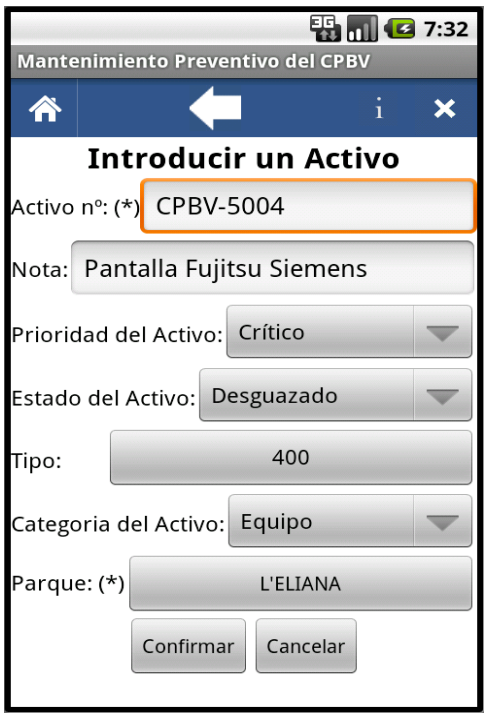

Figura 41: Edición Activos

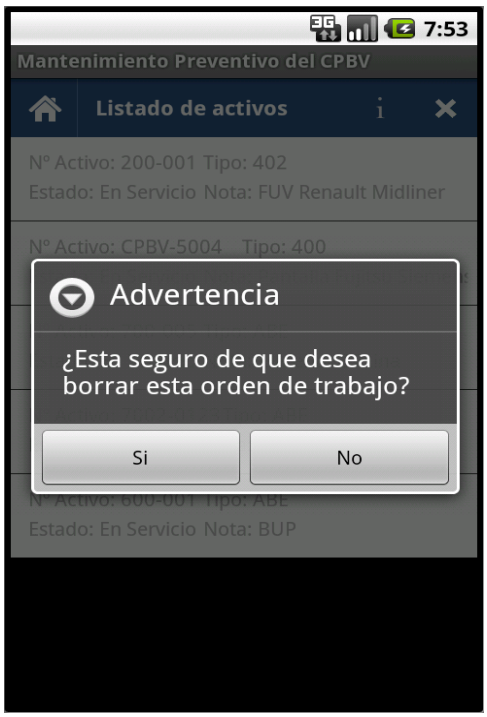

Figura 42: Borrar Activos

5.2.5 Gestión de Ordenes de Trabajo

Seguidamente, desde el menú principal se puede acceder a la gestión de ordenes de trabajo pulsando el botón situado en la parte superior izquierda, y que tiene una arquitectura muy similar a la de la gestión de activos, la única diferencia básica es que una orden de trabajo se puede cerrar, eliminándose del listado de tareas pero no del historial de revisiones.

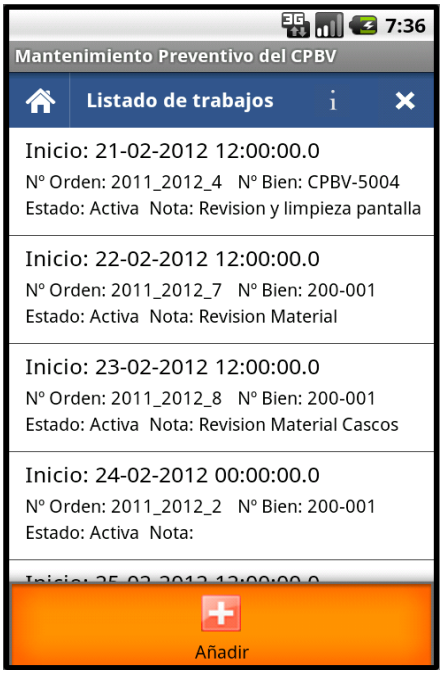

Figura 43: Listado de Tareas

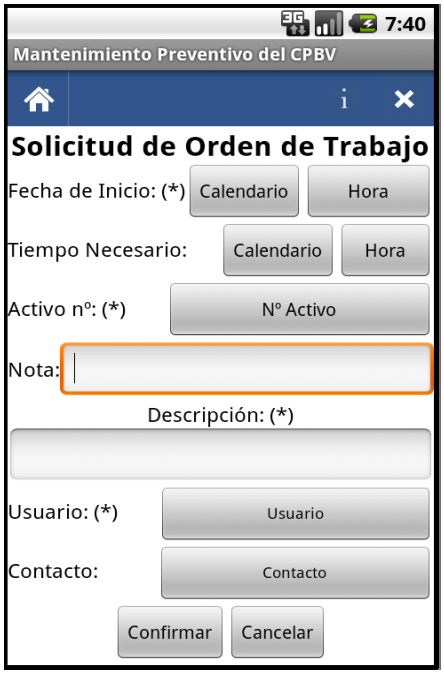

Figura 44: Añadir Tareas

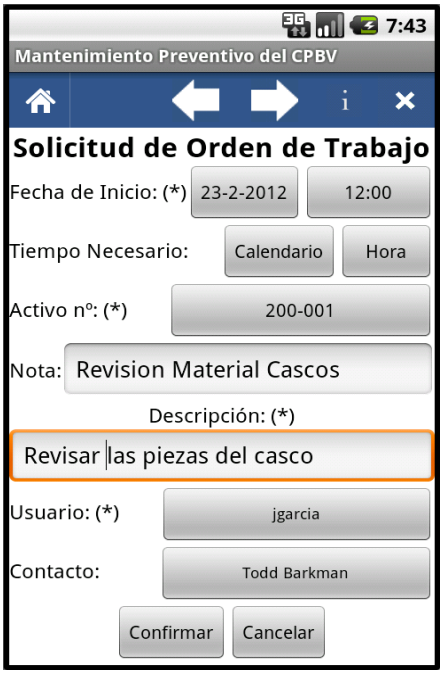

Figura 45: Edición de Tareas

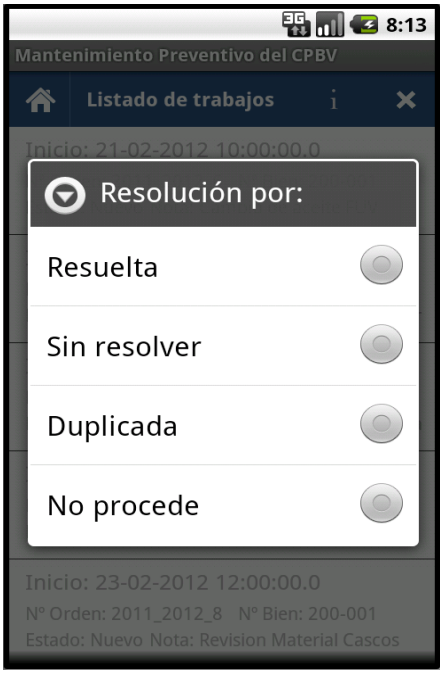

Figura 46: Cerrar Tareas

5.2.6 Historial de Revisiones

Respecto al historial, al que se puede acceder desde el menú principal pulsando sobre el botón situado en la parte inferior derecha, tenemos en primer lugar una pantalla de todos los activos que se encuentran en el parque seleccionado.

A continuación se selecciona el activo deseado y se pulsa sobre este, se puede obtener el listado de todas las ordenes de trabajo que se han realizado sobre este activo y visualizar las tareas de forma informativa.

Se ha seguido una política de ahorro y eficiencia en lo que corresponde en la reutilización de pantallas, por tanto no es necesario incluir las pantallas de listado de activos, dado que es el mismo que en la figura anterior.

Seguidamente y tras pulsar sobre un activo para que muestre su listado de tareas, se muestra la misma pantalla de listado de tareas con la única diferencia de que aparecerán aquellas tareas que ya han sido cerradas como se muestra en la figura.

Finalmente, tras pulsar sobre una tarea, aparece un QuickAction que nos despliega un menú para poder visualizar la tarea como se muestra en la figura, con la diferencia de que no se puede editar la información.

Figura 47: Historial de Tareas

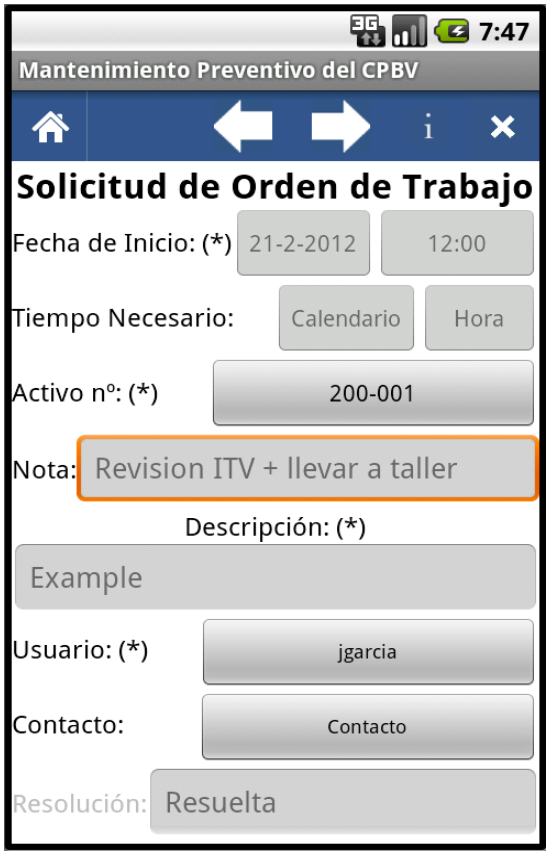

Figura 48: Visualizar Historial

5.2.7 Mensajes de la Aplicación

A continuación, se detalla brevemente un conjunto de figuras con el resto de posibles mensajes que se puede encontrar un usuario al utilizar la aplicación.

Figura 49: Fijar Fecha

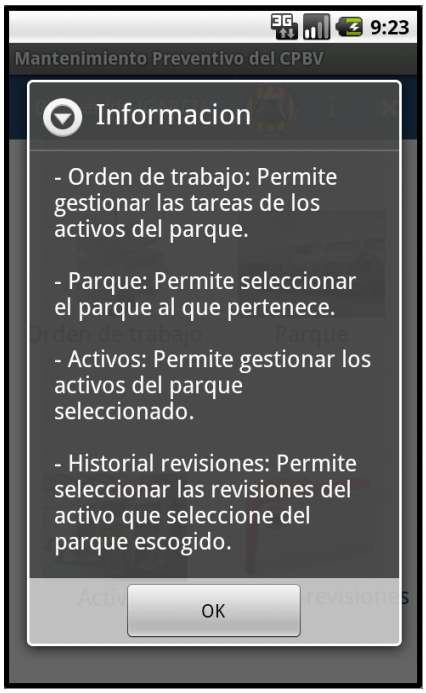

Figura 50: Información

Figura 51: Fijar hora

Figura 52: Selección

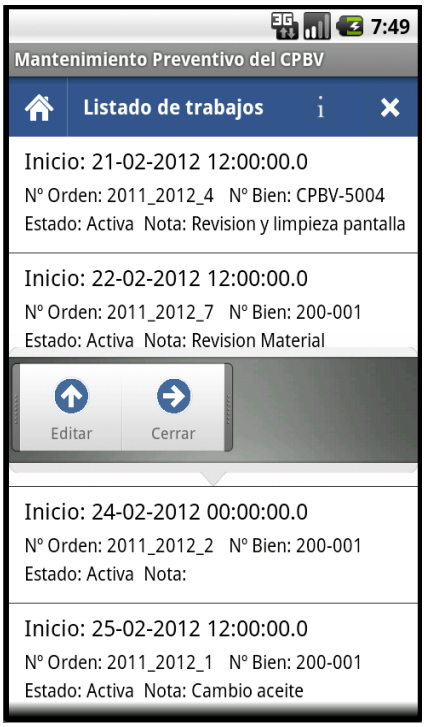

Figura 53: QuickActions

Figura 54: Error conexión

Finalmente en los apartados de edición de tareas o de edición de activos se puede consultar las guías informativas para realizar las tareas de mantenimiento o para conocer las partes y características respectivamente.

Figura 55: Guías Operativas

6 Ampliaciones

Tanto la aplicación como el servicio web REST están preparados y programados para facilitar el diseño e implementación de nuevas funcionalidades en un futuro para el Consorcio.

Este sistema de mantenimiento es el esqueleto de un sistema que se incrementara en el futuro según las necesidades del personal operativo, con lo que se pretende diseñar para informatizar toda esta gestión y que además sea de forma móvil.

Una posible e inmediata ampliación podría ser el registro de horas del personal operativo que realiza dicho mantenimiento y a partir de aquí se pueden obtener estadísticas anuales de eficiencia, velocidad, eficacia...etc.

Por otro lado, otra ampliación con respecto al registro de horas, sería que el personal operativo registre el inicio del mantenimiento y que la aplicación automáticamente, mediante un cronometro, vaya contabilizando el tiempo, y que cuando el usuario finalice la gestión y el mantenimiento y lo registre en la aplicación, esta finalice el tiempo y automáticamente indique el intervalo de tiempo que ha consumido el usuario en realizar dicha tarea.

Finalmente, se puede realizar una ampliación que incremente la seguridad en los servicios web, de forma que cuando se solicite información a través de la aplicación se vuelva a hacer una autenticación IMAP en el servidor de correo de la compañía, aunque estuvo contemplado, finalmente se decidió por no diseñar este modulo por redundancia.

7 Conclusión

En conclusión, este proyecto realizado durante todo este tiempo en el Consorcio Provincial de Bomberos de Valencia ha servido de una introducción en el mercado laboral utilizando los conceptos aprendidos en la Universidad Politécnica de Valencia. Además ha sido muy gratificante poder ayudar al cuerpo de bomberos y devolverle una pizca de lo que ellos hacen por nosotros a diario.

En primer lugar, puntualizar que todos los objetivos que se han desarrollado en la fase de inicio han sido llevados a cabo con éxito, gracias a la motivación de que el sistema ha sido requerido por el mismo Gerente del Consorcio y sobretodo a que el proyecto era de cierta importancia debido a que se quiere aplicar de forma inmediata.

Por otro lado, este proyecto resulta innovador para el consorcio, aunque existen algunos software Open Source de tipo web en el mercado, se requería una aplicación para un dispositivo móvil dado que la gestión de mantenimiento requiere de mucha movilidad.

Además de esto la solución implementada ha ofrecido ciertas ventajas, como el peso reducido de los dispositivos móviles, la interfaz intuitiva diseñada para el proyecto, la integración con las bases de datos del consorcio y la entrada de datos “in-situ”.

Gracias a esto, se han eliminado los costes de impresión de los formularios de gestión de mantenimiento y/o la justificación de que una orden de trabajo se ha realizado pensando en el medio ambiente, además otra ventaja es la continua actualización de la información en las bases de datos y el registro de las revisiones para futuras consultas.

El proyecto también requiere de cierto personal, sobretodo personal operativo de central, que pueda gestionar el mantenimiento de los parques con un dispositivo móvil, por tanto, ha sido muy complaciente el poder incrementar los puestos de trabajo y todo ello a coste cero por lo que respecta el desarrollo y licencias del software.

Finalmente, desde el punto de vista personal, he podido aprender a programar en una plataforma móvil utilizando metodologías ágiles (RUP) y plasmándolo en la memoria del proyecto, además de estar en contacto constante con el cliente, realizando reuniones y obteniendo requisitos para la aplicación a través de ellas.

8 Bibliografía

Reto Meier, "Professional Android 2 Application Development", Ed: Wrox Programmer to Programmer, Marzo 2010.

Dan Pilone, Neil Pitman, UML 2.0 in a Nutshell. Ed: O'Reilly, June 2005

Hans-Erik Eriksson, UML 2.0 Toolkit. Ed: Wiley 2004.

Android developer Guide: http://developer.android.com/guide/index.html *accedido en Marzo 2012*

Mark L. Murphy, "Beginning Android 2", 2010.

Web oficial Android: www.android.com *accedido en Abril 2012*

Grupo de desarrolladores Android: http://groups.google.com/group/desarrolladores-android *accedido en Marzo 2012*

Desarrollo Iterativo: http://es.wikipedia.org/wiki/Desarrollo_iterativo_y_creciente *accedido en Febrero 2012*

Desarrollo CRUD: http://es.wikipedia.org/wiki/CRUD *accedido en Marzo 2012*

Diseño singleton: http://www.vogella.de/articles/DesignPatternSingleton/article.html *accedido en Marzo 2012*

Diseño REST: http://www.vogella.de/articles/REST/article.html, *Febrero 2012*

REST + Spring: http://www.ibm.com/developerworks/webservices/library/wa-restful *accedido en Febrero 2012*

Cliente Spring: http://www.springsource.org/ *accedido en Febrero 2012*

QuickActions: http://www.londatiga.net/it/how-to-create-quickaction-dialog-in-android, *Marzo* 2012