UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Desarrollo de la interfaz de usuario para la gestión de un sistema multiagente en entornos paisajísticos

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Rives Estellés, Sergio

Tutor: Poza Luján, José Luis

Tutor: Carrascosa Casamayor, Carlos

2016-2017

Resumen

En los últimos años, las nuevas tecnologías se han introducido en todo tipo de ámbitos. Este proyecto surge para estudiar cómo introducir y aprovechar estos avances en el cuidado de plantas.

El objetivo del actual proyecto se centra en poder monitorizar e interactuar con una red de macetas inteligentes sensorizadas y dotadas de actuadores mediante una aplicación móvil para la plataforma Android.

La aplicación debe poder gestionar todos los elementos de la red y permitir a los usuarios de una forma cómoda y controlada realizar el cuidado de sus plantas desde sus dispositivos móviles.

Palabras clave: Agentes, Macetas, Aplicación, Jardín, Plantas, Chombo, Prototipo.

Abstract

In recent years, new technologies have been introduced in all kinds of areas. This project comes to light to study how to introduce and take advantage of these technological advances for plant caring.

The objective of the current project is to be able to monitor and interact with a network of smart pots sensitized and equipped with actuators through a mobile application for the Android platform.

The application must be able to manage all the elements of the network and allow users in a comfortable and controlled way to take care of their plants from their mobile devices.

Keywords: Agents, Pots, APP, Garden, Plants, Chombo, Prototype.

Resum

En els últims anys, les noves tecnologies s'han introduït en tot tipus d'àmbits. Aquest projecte sorgix per a estudiar com introduir i aprofitar estos avanços en l'atenció de plantes.

L'objectiu de l'actual projecte se centra a poder monitoritzar i interactuar amb una xarxa de tests intel·ligents sensorizadas i dotades d'actuadors per mitjà d'una aplicació mòbil per a la plataforma Android.

L'aplicació ha de poder gestionar tots els elements de la xarxa i permetre als usuaris d'una forma còmoda i controlada realitzar l'atenció de les seues plantes des dels seus dispositius mòbils.

Paraules clau: Agents, Tests, Aplicació, Jardí, Plantes, Chombo, Prototip.

Tabla de contenidos

1. Introducción

1.1. Entorno y motivación

Según la RAE el paisajismo es el "estudio o diseño del entorno natural, especialmente de parques y jardines". Siendo una actividad muy relacionada con el cultivo, aunque desde un punto de vista más estético. Las primeras presencias de esta actividad se remontan a la antigua Babilonia, con una de las grandes maravillas del mundo antiguo como son los jardines colgantes.

En la actualidad es una actividad que está presente en todo nuestro entorno cotidiano; desde los jardines privados, hasta parques públicos. Y aun siendo tan extendida, el progreso en este campo continúa estancado, manteniendo procesos arcaicos en el cuidado de las plantas.

Las plantas son seres vivos que dependen del entorno para su supervivencia, dado que generalmente no tienen habilidad motriz. Sería posible mejorar la capacidad de supervivencia de las plantas si se implementara un sistema autónomo que reaccionara de forma inmediata a sus necesidades, mediante acciones como pueden ser: dotarlas de movimiento, riego y fertilizante automáticos, entre otras.

Desde el ámbito de la ingeniería informática consideramos que existen herramientas que podrían dotar a las plantas de los mecanismos anteriores. De forma que se reducirá el trabajo de mantenimiento y la mortalidad de las plantas. Simplificando así la complejidad de la actividad paisajística.

Actualmente ya se usan sistemas para automatizar el cuidado de elementos de jardinería, como puede ser el control del riego. Sin embargo, todavía queda mucho territorio sin explorar en cuanto a la automatización de otras acciones que aprovechen nuevos sensores y actuadores introducidos en el mercado, así como en el ámbito de la inteligencia artificial y el aprendizaje mediante el uso de agentes.

El sistema sobre el cual vamos a desarrollar la aplicación de nuestro proyecto hace uso de estos nuevos sensores y actuadores para permitir al sistema de agentes un mayor control para poder alcanzar las condiciones óptimas para la planta. Este sistema requiere de cierta operatividad por parte del usuario experto a fin de parametrizar correctamente la red de Chombos y

asegurar una buena gestión de éstos; esta operatividad es la se obtendrá mediante la aplicación (a partir de ahora APP o ChomboApp).

1.2. Objetivos

La finalidad del presente proyecto es la creación de un sistema que permita la supervivencia de las plantas de forma desatendida en entornos diversos gracias al uso de nuevas tecnologías, realizando un registro de todos los datos relacionados con la vida de dichas plantas y permitiendo una monitorización de estos datos.

Para alcanzar el objetivo, y debido a la complejidad de este, el diseño e implementación del sistema ha sido dividido en distintos proyectos de final de grado, siendo el presente uno de ellos:

Nombre	Ramón Ferrer Mestre
Rol	Desarrollador del sistema de persistencia y comunicaciones
Objetivo	Implementar la persistencia

Nombre	Sergio Rives Estellés
Rol	Desarrollador de la interfaz de usuario (IU) y el render
Objetivo	Implementación en android de la IU

Nombre	Alberto Martín Taberner
Rol	Desarrollador del sistema multiagente y manager (simulador)
Objetivo	Implementar el sistema de toma de decisiones

Nombre	Salva Pons
Rol	Desarrollador hardware
Objetivo	Implementar funciones del Arduino y montar el hardware

Este proyecto está centrado en crear una aplicación capaz de permitir al usuario monitorizar e interactuar con el sistema de Chombos (a partir de ahora llamado jardín).

Las plantas que crecen en un Chombo necesitan ser observadas de forma remota para poder asegurar que el cuidado automatizado está siendo efectivo. Además, es necesario poder configurar ciertos parámetros del jardín para que este cuidado sea adecuado a las condiciones del entorno en el que se muevan los Chombos.

1.3. Estructura del documento

Este documento se estructura en 6 diferenciadas secciones, las cuales se resumen de la siguiente forma:

En la primera sección se ha explicado brevemente la motivación y objetivos del proyecto.

En la segunda sección se introduce el entorno del sistema en el que se realiza el proyecto, analizando otros sistemas similares que servirán como base para especificar las características a tener en cuenta para este proyecto.

En la tercera sección se explica en profundidad el diseño de la totalidad del sistema sobre el que se basa en proyecto, y posteriormente se entra en detalle sobre el papel de cada una de las partes (haciendo un mayor hincapié en el proyecto que nos ocupa).

En la cuarta sección se explica las implementaciones que se han llevado a cabo para construir el prototipo de aplicación de acuerdo al diseño detallado anteriormente, así como las pruebas realizadas.

En la quinta y última sección se extraen unas conclusiones sobre el proyecto, se enumeran las distintas dificultades que han surgido a lo largo de su desarrollo y la forma de afrontarlas, y se plantean una serie de posibles trabajos futuros a realizar.

Al final del documento se pueden encontrar dos secciones extra con las referencias y el glosario.

2. Estudio de mercado

A continuación vamos a estudiar una serie de productos/sistemas relacionados con el entorno paisajístico, de forma que podamos extraer de todos ellos los requisitos más importantes que debe cumplir nuestro propio sistema y podamos elegir con certeza las tecnologías a usar para lograr el objetivo.

2.1 Competidores potenciales/tecnologías similares

2.1.1 Parrot Pot

Maceta inteligente ya a la venta a un precio de 150 $, cuenta con 4 tipos de sensores: sensor de PH, de temperatura, de luz y de humedad. Si la planta tuviese alguna necesidad la maceta envía notificaciones vía bluetooth al móvil para poder atenderlas. Como función autónoma puede autorregularse si la planta necesita agua y está equipada con un tanque de agua a tal efecto. Funciona con 4 pilas AA que le dan una autonomía de 1 año. Utiliza la tecnología Flower Power.

Características:

- **SENSORS**
  - Capacitive soil moisture sensor (measurement range: 0 to 50%)
  - Fertiliser level sensor
  - Light sensor: 0 to 1000 µmol m-2. s-1
  - Air temperature sensor: 0°C to +55°C
  - Rain and water resistant (IPX5 rating)
- **SIZE**
  - Diameter x height: 20.5 cm x 31.2 cm
- **DETAILS**
  - Operating temperature: 0°C to 55°C
  - Material: ABS, PP, rubber
  - LED status indicator (red, green)
  - Built-in water tank: 2.2 litres
  - Soil volume: 2.4 litres
  - Batteries: 4 AA

- **CONNECTIVITY & APPS**
  - Connection via Bluetooth® Smart/Bluetooth V4.0 BLE
  - Free Parrot Flower Power app
  - Application programming interface (API)

http://www.odditycentral.com/technology/bad-with-plants-this-high-tech-flower-pot-can-keep-any-plant-alive.html

2.1.2 Flower Power

Tecnología usada por las macetas Parrot Pot, consistente en un dispositivo bluetooth adaptable a cualquier maceta de tamaño medio/grande que contiene 4 tipos de sensores: sensor de PH, de temperatura, de luz y de humedad. Gracias a una conexión con una base de datos de más de 8000 plantas, es capaz de enviar alertas al móvil por bluetooth si las condiciones de la planta no son óptimas.

Mediante su aplicación móvil es posible monitorizar el estado de una o más plantas, consultar un histórico con las condiciones de cada planta o planificar tareas a realizar para que las plantas estén en las mejores condiciones, siendo ésta planificación automática.

El dispositivo funciona con una pila AAA y su precio es de 50 $.

Ilustración 2.1 Flower Pot

https://www.amazon.com/Parrot-Flower-Power-Bluetooth-dedicated/dp/B00FOM2Y6W

2.1.3 RoseRunner (2012)

Consiste en un robot capaz de moverse entre las macetas, fertilizarlas y moverlas de un punto a otro.

- Interfaz de control
- Joystick
- PC interfaz

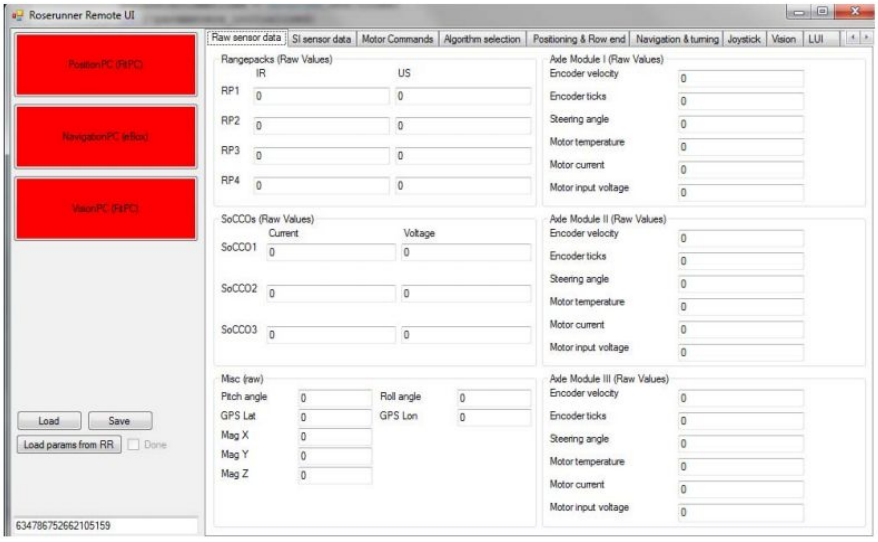


Ilustración 2.2. Rose Runner interfaz

Ilustración 2.3. Rose Runner
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.259.3801&rep=rep1&type=pdf

2.1.4 Jurema Action Plant

Máquina bio-interactiva capaz de transportar una planta sensitiva (Mimosa Pudica), intenta aumentar las funciones que puede realizar una planta habilitando que usen tecnologías similares a las que usan los humanos.

Usa unas señales eléctricas que circulan a través de las células de las plantas como si de nervios humanos se tratase, y permite que la planta se aleje de un obstáculo al tocarlo, usando la propia planta y sus señales "nerviosas" como sensor.

Adicionalmente dispone de un tanque de agua y un placa con los componentes electrónicos, lo cual interpreta las señales y necesidades de la planta para traducirlas en movimiento o dispensa de agua.

Ilustración 2.4. Jurema Action Plant

https://ivanhenriques.com/2011/06/02/jurema-action-plant/

2.1.5 Plantas nómadas

La Planta Nómada es un organismo vivo, constituido por un sistema robótico, una especie vegetal orgánica, un conjunto de celdas de combustible microbianas y fotovoltaicas.

Para sobrevivir, este organismo toma agua contaminada y la procesa en sus celdas de combustible mediante una colonia de bacterias autóctonas de estas aguas, que se alimentan transformando los nutrientes en electricidad, para ser almacenada por su sistema de cosecha de energía.

En este proceso de biodegradación mejora la calidad del agua y provee a la especie vegetal que también produce electricidad con su metabolismo. La liberación de oxígeno es el remanente de este ciclo energético. Por tanto no solo es una especie adaptada al entorno modificado, sino que también restituye la energía que dispone de la tierra.

Ilustración 2.5. Plantas nómadas

https://youtu.be/kQwYEWaEHTs

http://www.plantasnomadas.com/

2.1.6 Fliwer

Sistema de riego inteligente para huertos, jardines y macetas. Es una tecnología que valora las necesidades de las plantas en tiempo real para proporcionar un riego adecuado, así como proporcionar información al usuario mediante WiFi o 3G sobre el estado de las mismas.

Ilustración 2.6. Fliwer

http://www.fliwer.com

2.1.7 Click & Grow Smart flowerbed

Semillero inteligente que proporciona los cuidados suficiente para que las plantas broten en 1-2 semanas, cuenta con un pequeño depósito y se encarga de que las semillas tengan las cantidades adecuadas de agua, oxígeno y nutrientes.

Utiliza 4 pilas AA y el precio oscila entre 20-60€ según el modelo.

Ilustración 2.7. Click & Grow Smart flowerbed

https://www.amazon.com/Click-Grow-flowerbed-Cockscomb-Indoor/dp/B008K95K80

2.1.8 A Control Method for a Swarm of Plant Pot Robots that Uses Artificial Potential Fields for Effective Utilization of Sunlight (CMSPR)

Es un proyecto de la universidad de Tokio de agricultura y tecnología, desarrollado por Masato Yuasa y Ikuo Mizuuchi.Que tiene como objetivo maximizar el uso de la luz solar en el cultivo de plantas.

El sistema que implementan se basa en una serie de robots capaces de moverse y una serie de sensores instalados en la maceta, con el fin de monitorizar de la planta. El funcionamiento es realizado por medio de la transmisión de la información de estado de los sensores al ordenador de control, este calcula la orden y la envía a los robots para su ejecución.

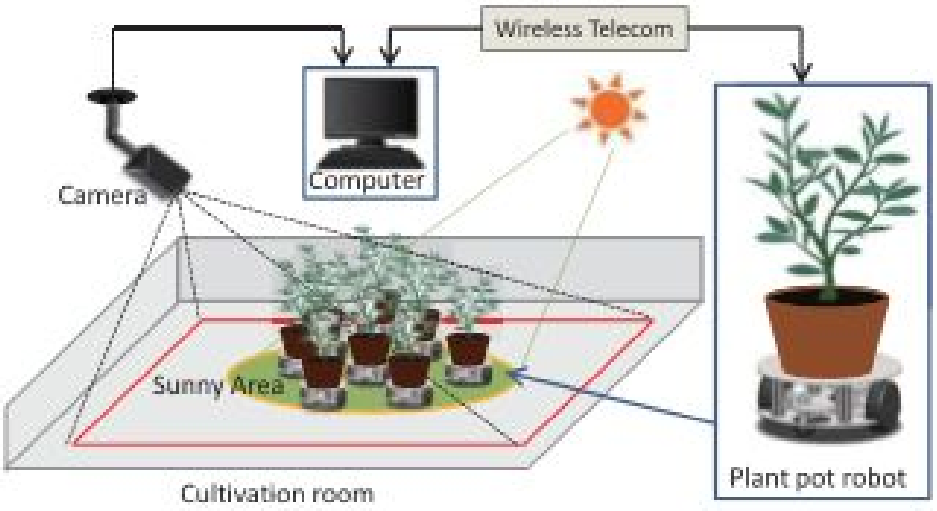


Ilustración 2.8. CMSPR

https://www.fujipress.jp/jrm/rb/robot002600040505/

2.1.9 PotPet

Este sistema tiene el objetivo de permitir a las plantas el moverse a hacia lugares soleados(sensor de luz) o detectar personas(sensor de movimiento), además puede advertir a las personas de la necesidad de riego y del punto de parada del riego mediante el sensor de humedad.

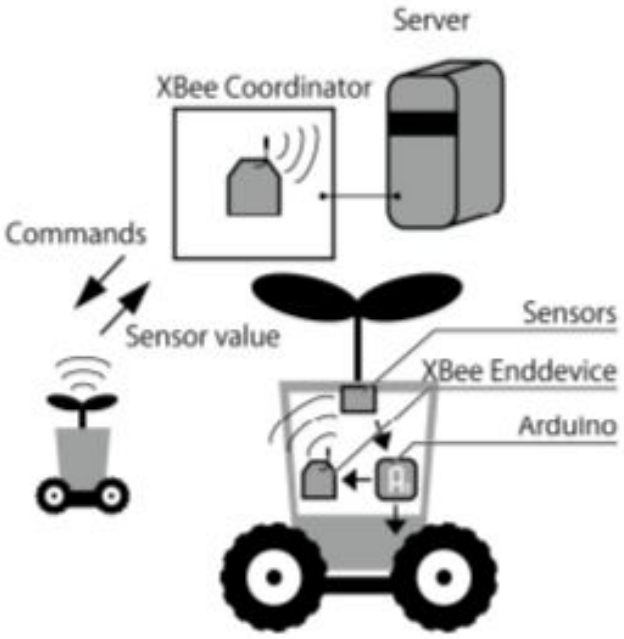


Ilustración 2.9. PotPet

http://sappari.org/pdf/potpet_tei2011.pdf

2.2 Tabla comparativa

	Parrot Pot	Flower Power	Rose Runner	Jurema Action Plant	Planta nómada	Fliwer	Click & Grow	Maceta intel.
Sensor humedad	✓	✓	✗	✓	✓	✓	✓	✓
Sensor fertilizante	✓	✓	✓	?	?	✓	✓	✓
Sensor luz solar	✓	✓	✗	?	✓	✓	✗	✓
Sensor temperatura	✓	✓	✗	?	✗	✓	✗	✓
Riego automático	✓	✗	✓	✓	✓	✓	✓	✓
Depósito agua	✓	✗	✓	✓	✗	✗	✓	✓
Movimiento	✗	✗	✓	✓	✓	✗	✗	✓
Alertas	✓	✓	✗	✗	✗	✓	✗	✓
Interior	✓	✓	✗	✓	✗	✓	✓	✓
Exterior	✓	✓	✓	✓	✓	✓	✗	✓
Simulación	✗	✗	✓	✗	✗	✓	✗	✓
Precio	150 $	50 $	?	?	?	1200 €	20-60 €	?
Disponible	✓	✓	✗	✗	✗	✓	✓	✗

Tabla 2.1. Comparativa mercado

	Parrot Pot	Flower Power	Rose Runner	Jurema	Planta nómada	Fliwer	PotPet	CMSPR	Click & Grow	Maceta intel.
Sensor humedad	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
Sensor fertilizante	✓	✓	✓	?	?	✓	✗	?	✓	✓
Sensor luz solar	✓	✓	✗	?	✓	✓	✓	✓	✗	✓
Sensor temperatura	✓	✓	✗	?	✗	✓	✗	✓	✗	✓
Sensor de movimiento	✗	✗	✗	✗	✗	✗	✓	?	✗	✗
Riego automático	✓	✗	✓	✓	✓	✓	✗	?	✓	✓
Depósito agua	✓	✗	✓	✓	✗	✗	✗	?	✓	✓
Movimiento	✗	✗	✓	✓	✓	✗	✓	✓	✗	✓
Alertas	✓	✓	✗	✗	✗	✓	✓	?	✗	✓
Interior	✓	✓	✗	✓	✗	✓	?	✓	✓	✓
Exterior	✓	✓	✓	✓	✓	✓	✓	?	✗	✓
Simulación	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓
Disponible	✓	✓	✗	✗	✗	✓	✗	?	✓	✗

Tabla 2.2. Comparativa mercado

2.3 Tecnologías

Para desarrollar un prototipo de aplicación que permita la monitorización del jardín estaban al alcance numerosos tipos de tecnología. Finalmente se decidió apostar por Android como plataforma, ya que la tecnología móvil es algo cada vez más cotidiano y al alcance de todo el mundo, por lo que se han utilizado las siguientes herramientas:

2.3.1. Android Studio

Android Studio es un entorno de desarrollo integrado oficial para Android creado por Google y basado en el software IntelliJ IDEA de JetBrains. Se decidió usar este IDE por ser el oficial para la creación de aplicaciones Android, por el gran soporte y documentación que recibe tanto de sus desarrolladores como de la comunidad, por ser publicado de forma gratuita a través de la Licencia Apache 2.0 y por la gran cantidad de librerías disponibles para este IDE. El poder realizar la implementación usando como lenguaje de programación Java también ha sido una ventaja, dada la experiencia adquirida previamente programando en este lenguaje.

El principal competidor de Android Studio es Eclipse, pues antes era el más usado en la programación para Android. Sin embargo, desde que apareció Android Studio, Eclipse recibe menos soporte para la programación en Android.

2.3.2. Androidplot

Librería que permite crear gráficas de dos dimensiones XY, lo cual es necesario en este proyecto para representar los datos históricos de los Chombos. En un principio esta librería estaba pensada para representar dos series de datos, pero se pudo comprobar que no tenía problemas en representar hasta 6 series de datos (que era la cantidad de datos que necesitábamos representar simultáneamente para cada Chombo).

Existen muchas librerías alternativas libres similares a Androidplot: MPAndroidChart, Holo Graph Library, aChartEngine, ChartView, aFreeChart, ChartDroid, Charts4j, GraphView, WilliamChart, HelloCharts... Se analizaron todas ellas junto a Androidplot para determinar cuál de ellas se adaptaba mejor a nuestras necesidades, y finalmente las demás se descartaron por encontrar alguna deficiencia en ellas: poder mostrar únicamente una serie de datos, tener un diseño que desentonaba demasiado con el resto de la aplicación, no ser adecuada para bajas resoluciones y/o poseer una documentación pobre.

2.3.3. Volley

Librería muy popular destinada al tratamiento de las comunicaciones y soportada oficialmente por Google, la cual ha sido necesaria para gestionar las peticiones REST a la BDD histórica.

La principal alternativa a esta librería es Retrofit, la cual tiene tiempos de respuesta más rápidos que Volley. Sin embargo, Retrofit no puede manejar la descarga de imagenes y requeriría de otra librería extra, por lo que finalmente nos decantamos por Volley al ser capaz de realizar la descarga de imágenes y la comunicación con una sola librería. Por otro lado, la diferencia de tiempos de respuesta en el orden de los milisegundos no es muy relevante para nuestra aplicación.

2.3.4. Google Maps

Google Maps es una aplicación muy conocida para obtener la localización de lugares alrededor del mundo. En el proyecto se ha aprovechado la API de Google Maps para construir el *render*, generando un mapa en el que representar la localización del jardín, su extensión y los elementos contenidos en él (Chombos y balizas) con un *overlay* (capa superpuesta donde se dibujan los elementos de interés).

A pesar de que se han encontrado alternativas para crear otro tipo de mapeados (Leaflet, Modest Maps, Polymaps), éstas no están tan extendidas ni cuenta con el amplio soporte que hay detrás de la API de Google.

3. Diseño

En el presente capítulo vamos a explicar todas las decisiones de diseño tomadas para construir el sistema global y cada una de sus partes.

3.1. Diseño completo del sistema

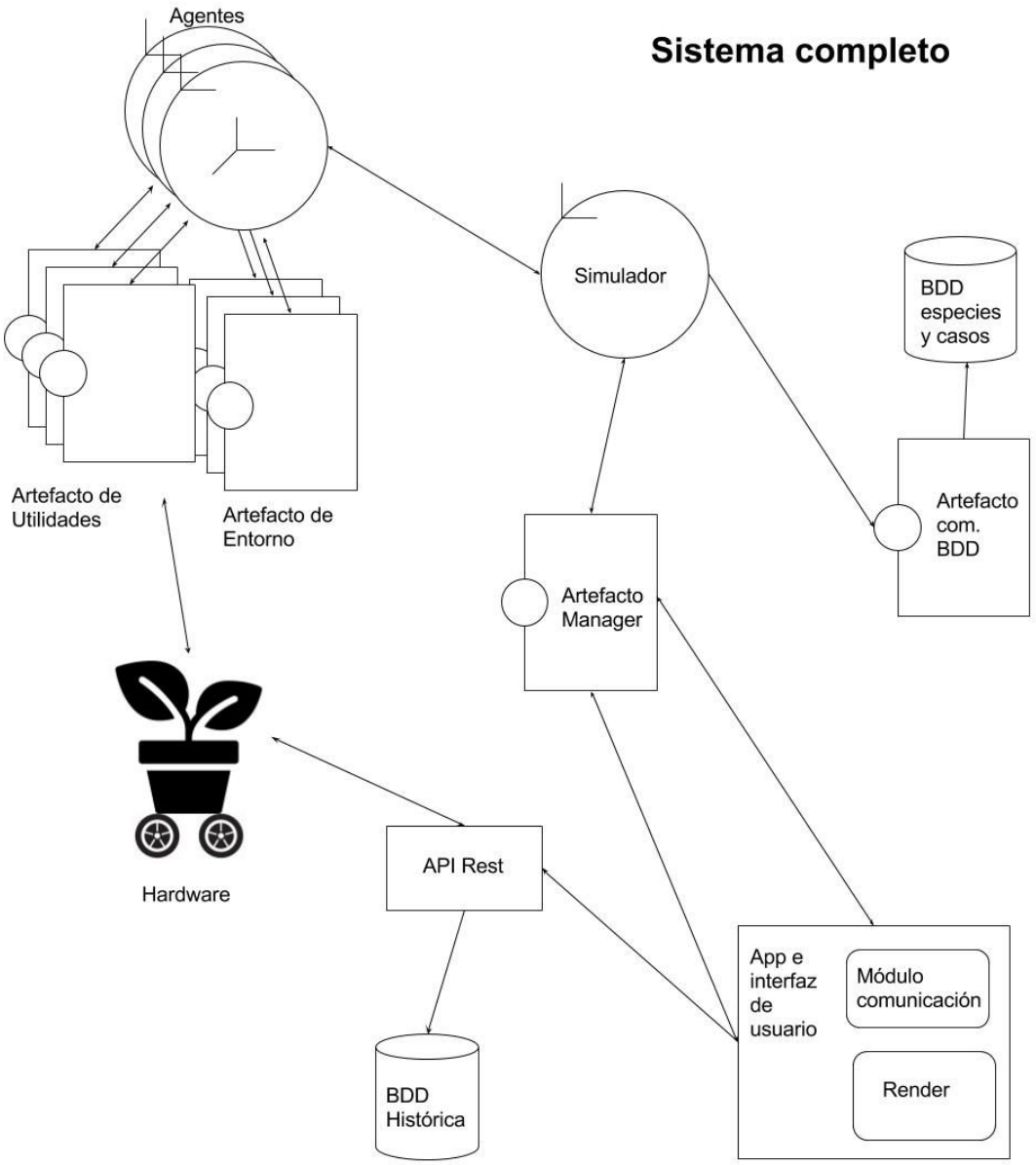


Ilustración 3.1. Diseño del sistema completo

En la *Ilustración 3.1* es posible contemplar el sistema en su totalidad, junto a todas las relaciones que existen entre los distintos proyectos dentro de este. Estos proyectos serían los cuatro mencionados anteriormente: la parte hardware o física del Chombo, la parte inteligente gestionada por el sistema de agentes, la parte de persistencia que da soporte al resto de los proyectos y por último la parte de monitorización mediante una interfaz de usuario (proyecto el cual nos ocupa).

3.2. Hardware

El primero de los módulos mencionados, correspondiente a la parte hardware, consiste en la creación de la maceta o Chombo, su sensorización y la adición de actuadores, de forma que con ayuda del sistema de agentes pueda desenvolverse de manera autónoma para que la planta que contiene crezca en las condiciones más idóneas posibles. Además de conectar con el proyecto encargado de implementar el sistema de agentes también tiene una conexión con el proyecto encargado de la persistencia de datos, pues la base de datos histórica se nutre de los distintos estados en los que se encuentran los Chombos a lo largo del tiempo para futuros estudios y evolución del aprendizaje. Sin embargo, este primer proyecto deberá integrarse con los otros tres en futuros trabajos.

En principio se han tenido en cuenta que el Chombo contará con una serie de sensores y actuadores que variarán según el modelo de Chombo:

Sensores:

- Nivel de humedad
- Nivel de temperatura
- Nivel de fertilidad o PH
- Nivel de luminosidad
- Carga de batería
- Nivel de agua en tanque
- Sensor de posicionamiento GPS

Actuadores:

- Movimiento lineal
- Movimiento rotativo
- Bomba de riego

3.3. Sistema de agentes y simulador

El segundo módulo consiste en el diseño e implementación del sistema de agentes y de un simulador. Este simulador (el cual se encuentra en un servidor) permite reproducir distintas condiciones y entornos de forma virtual para el estudio del comportamiento y la toma de decisiones de los Chombos, mientras que el sistema de agentes es el encargado de realizar esta toma de decisiones mediante casos que se almacenan en una base de datos.

En cada uno de los Chombos o macetas se implementa un agente, el cual mediante los artefactos correspondientes establece una comunicación con sus sensores/actuadores y con el simulador. El simulador a su vez hace uso de dos artefactos, “Artefacto Com. BDD” y “Artefacto Manager” (el cual a partir de ahora se le llamará Manager) para establecer una comunicación con las partes del sistema encargadas de la persistencia y la interfaz de usuario respectivamente.

Para realizar la toma de decisiones los agentes primero obtienen los datos de sus sensores y con esta información consultan con la base de casos para tratar de obtener una situación similar a la actual y la acción que se realizó. La base de casos le devolverá el caso con un índice de confort más alto, y éste será tenido en cuenta por el agente en el momento de tomar una nueva decisión. Durante este proceso también se proporcionan los datos actuales al Manager para que puedan ser recogidos por la interfaz de usuario.

Dentro de un jardín, además de Chombos existen “balizas”. Las balizas son elementos que el simulador interpreta y que condicionan el comportamiento de los Chombos dentro de un jardín. Los distintos tipos de baliza son:

- **Baliza punto de encuentro**: demarca un punto del jardín donde es posible dirigir a los Chombos con una orden. Es un elemento único.
- **Baliza barrera**: usada para delimitar una zona dentro del jardín por donde los Chombos no deben circular (puede usarse para representar un accidente en el terreno, de forma que los Chombos no queden atascados).
- **Baliza punto de recarga**: indica un lugar donde el Chombo puede acudir a recargar su batería.
- **Baliza suministro de agua**: señala el lugar donde un Chombo puede cargar su depósito de agua.

3.4. Persistencia

El tercer módulo proporciona al resto la persistencia necesaria para que todo el sistema esté sincronizado y permanezca en el tiempo. Hay dos partes diferenciadas dentro de la persistencia del sistema:

BDD de especies y casos:

En esta base de datos se almacenan todas las especies de plantas con una colección de casos que se actualiza de forma constante.

Los casos que se almacenan contienen una serie de valores de la situación actual del Chombo en un momento determinado tras la realización de una acción, y se les asigna un índice de confort que determina lo adecuada que ha sido dicha acción para el bienestar de la planta.

Estos casos serán consultados por los agentes para realizar una toma de decisiones cuando aparezca una situación similar a la del caso, y a través de la creación de nuevos casos o la actualización de casos similares el Chombo tenderá a mejorar el cuidado de la planta.

BDD histórica:

La base de datos histórica, a diferencia de la base de casos, no actualiza sus entradas y simplemente agrega nuevas. La finalidad inicial de esta base de datos era registrar todos los estados en los que se ha encontrado un Chombo en un momento determinado para su posterior estudio y mejora del aprendizaje de los Chombos. Sin embargo, se ha aprovechado esta información guardada para poder monitorizar el estado de los Chombos a lo largo del tiempo, de forma que desde la APP de monitorización podamos ver una progresión.

3.5. Interfaz de usuario y render (APP)

El cuarto y último módulo que forma el sistema global es el que nos ocupa: la creación de una aplicación móvil capaz de monitorizar e interactuar con el jardín de Chombos.

Esta aplicación podría construirse para cualquier tipo de plataforma pero en este proyecto se decidió crear el prototipo en Android por las razones mencionadas en el punto 2.3 de este documento.

Tras analizar todos los requerimientos que debería tener el prototipo en base al estudio de mercado realizado, primeramente se extrajeron los casos de uso que debería abordar la aplicación, los cuales pueden verse en la *Ilustración 3.2*.

Lo más importante era poder consultar en tiempo real todos los datos obtenidos de la sensorización de cada Chombo, así como representar la localización de unos Chombos respecto a otros.

En un segundo plano existen los casos de uso que los permiten modificar el jardín, tanto su parametrización (localización del jardín, radio de superficie) como la creación y edición de nuevos elementos dentro de este (tanto Chombos como balizas).

Por último están los casos de uso avanzados que están más orientados a usuarios expertos, ya que permiten realizar un estudio a fondo de la progresión de los Chombos y parametrizar los comportamientos modificando datos de las especies de plantas. La edición de pesos de la especie (modificar cuan importante es un un valor respecto de otro para la especie, ej: temperatura vs humedad) y la consulta de los datos histórico de un Chombo son los casos incluidos en esta categoría.

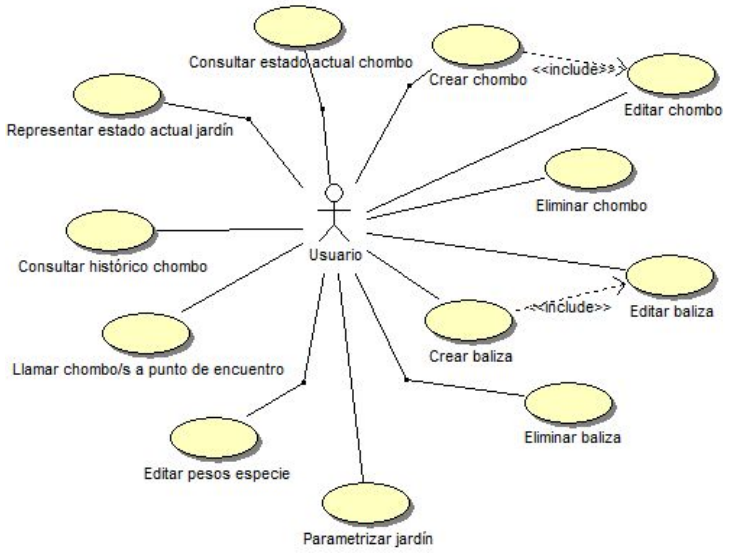
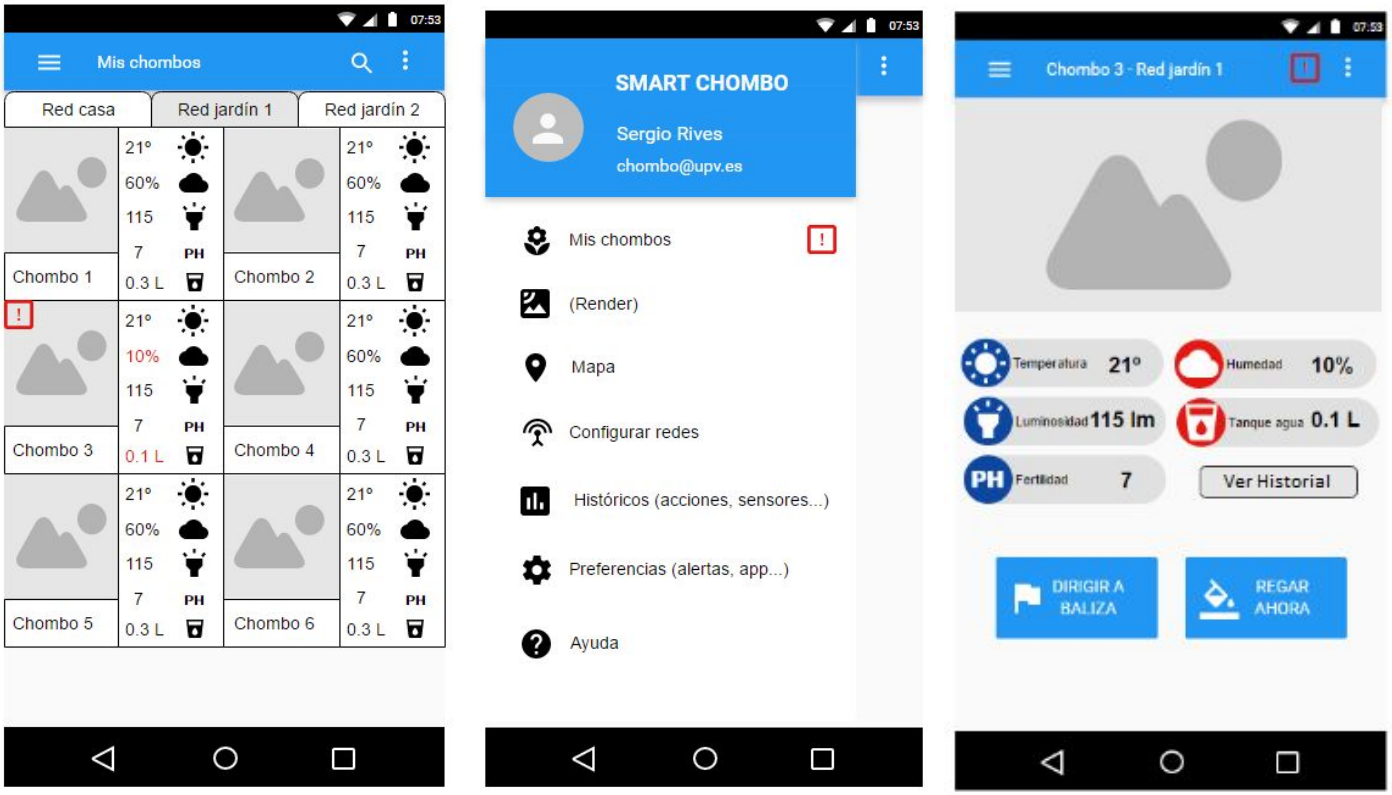


Ilustración 3.2. Casos de uso APP

Teniendo claras las funcionalidades que debía cumplir la aplicación, se diseñaron unos *mockups* poder disponer de una aproximación de cómo luciría la aplicación sin ser necesariamente el diseño final (*Ilustraciones 3.3, 3.4 y 3.5*).



Ilustraciones 3.3, 3.4 y 3.5. Mockups de ChomboAPP

Es importante destacar que las conexiones entre los distintos proyectos se tuvieron que consensuar, pues debían usarse tecnologías que fuesen compatibles entre sí. Una vez tomada esta decisión se crearon diagramas de flujo para representar estas conexiones. A continuación se muestran los diagramas que representan las conexiones de la APP con los proyectos de persistencia y el simulador/sistema de agentes.

3.5.1. Caso de uso: creación de Chombo

En el caso de la creación de un nuevo Chombo (*Ilustración 3.6*) es posible ver cómo ésta se produce desde el FragmentCreaChombo (la clase que controla la vista donde el usuario crea un nuevo Chombo) llama a la clase Factory para crear un objeto Chombo. Este objeto es entonces usado por la clase ManagerComm para conectar con el Manager que gestiona el simulador por protocolo UDP.

Tras realizar la petición de creación de un nuevo Chombo, pueden recibirse dos respuestas posibles: una de ellas nos dirá que el Chombo se ha creado correctamente (y una notificación en la APP nos lo hará saber), con lo

que la aplicación ya mostrará en el FragmentChombos (vista principal donde aparecen todos los Chombos del jardín) el nuevo Chombo creado; si por el contrario se recibe un error como respuesta (bien porque devuelve un error el Manager o por un *timeout*) la aplicación mostrará un mensaje informando de este error.

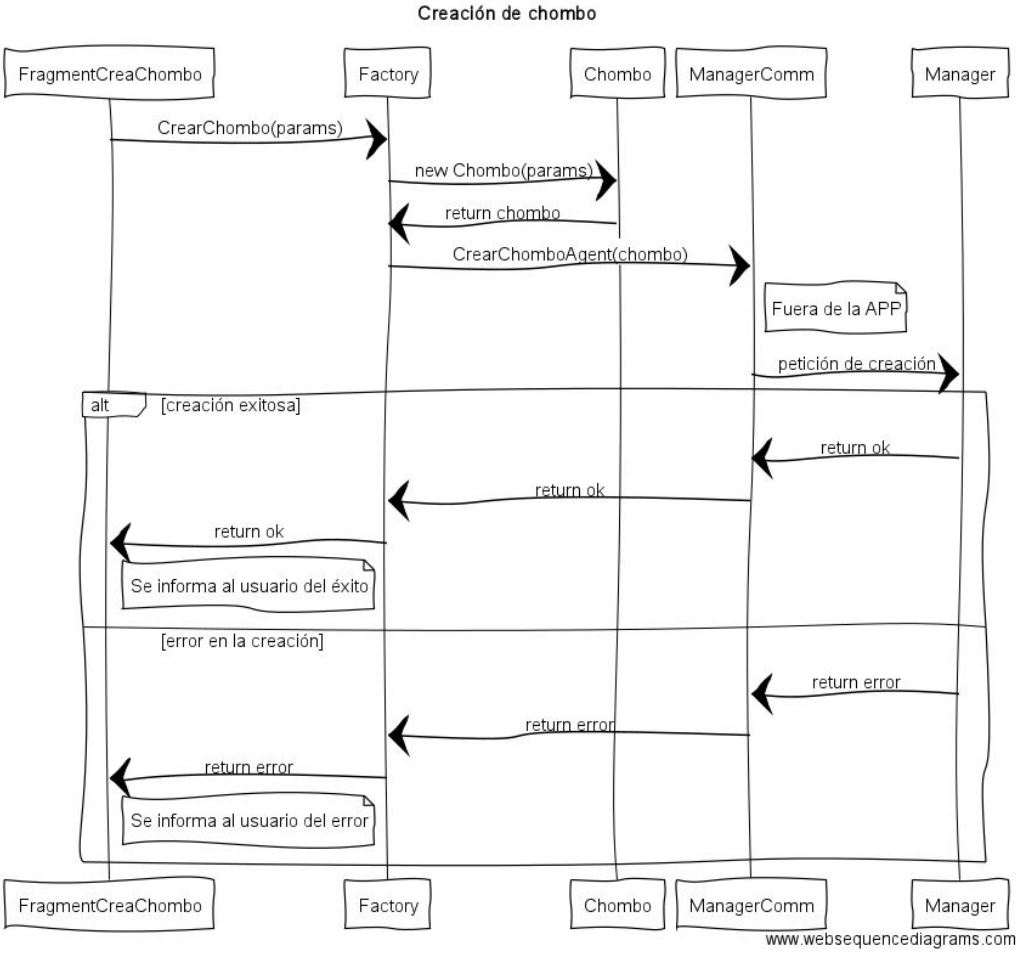


Ilustración 3.6. Creación de Chombo

3.5.2. Caso de uso: modificar Chombo

Los casos de eliminar Chombos o modificar los parámetros de un Chombo (*Ilustraciones 3.7 y 3.8*) son más simples, pues únicamente se enviará una petición al Manager con el identificador del Chombo (y los parámetros a modificar en el caso de editar un Chombo).

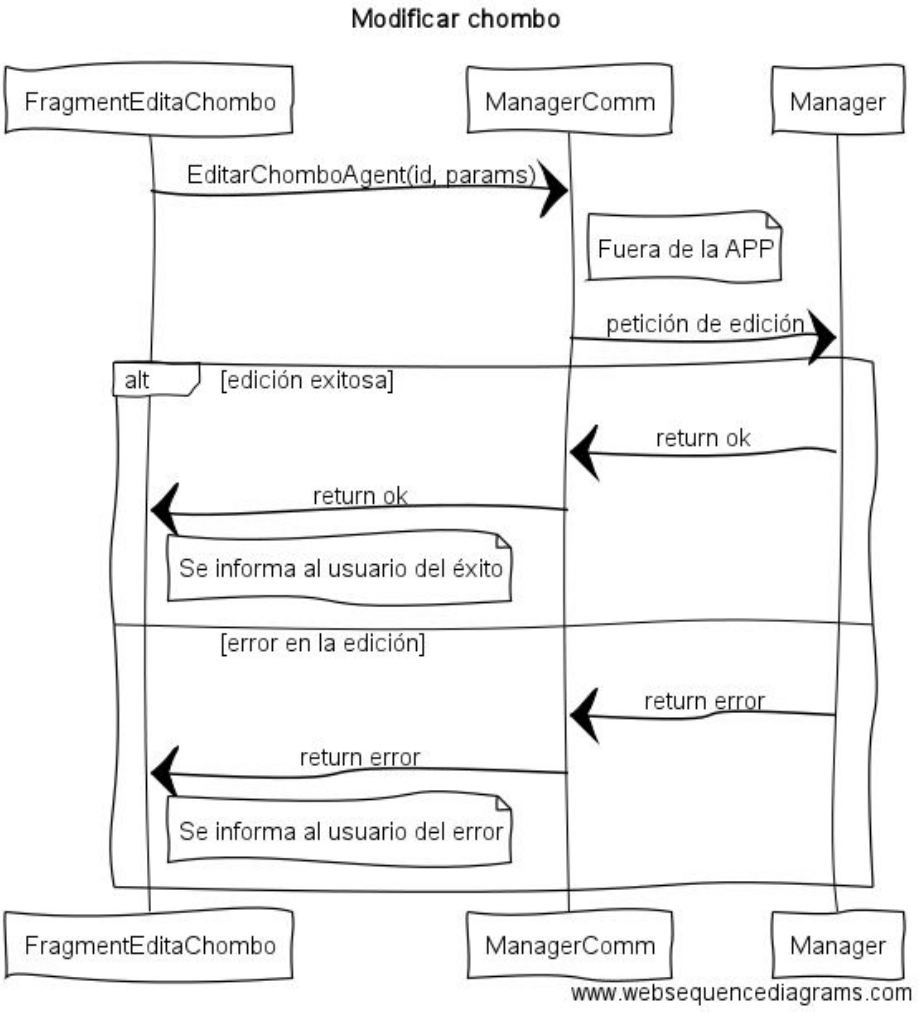


Ilustración 3.7. Modificación de Chombo

La modificación de algunos parámetros de un Chombo no supondrá un cambio directo en el comportamiento del Chombo. Sin embargo, modificar el tipo de planta que contiene sí tendrá unas mayores consecuencias, pues los casos almacenados en la base de casos para este Chombo quedarían inservibles y será necesario generar nuevos casos desde cero (o a partir de los casos base genéricos introducidos en la base de casos).

3.5.3. Caso de uso: eliminar Chombo

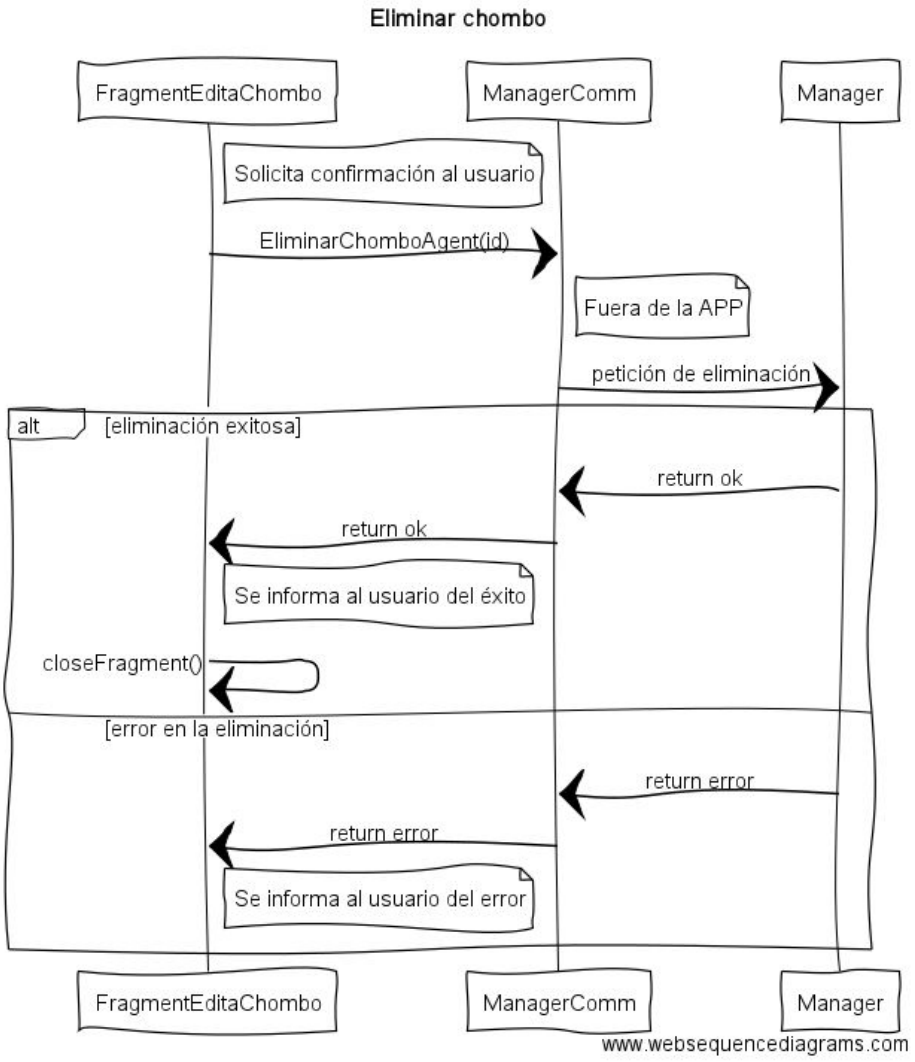


Ilustración 3.8. Eliminación de Chombo

Eliminar un Chombo de la red supone desconectarlo totalmente del jardín y que el servidor/simulador deje de tenerlo en cuenta. En el caso de un Chombo virtual esto implica que desaparece por completo; evidentemente no es así en el caso de un Chombo físico, pues únicamente perdería la conexión y el agente podría seguir tomando decisiones con la información proporcionada por sus sensores (sin tener acceso a la base de casos para la toma de decisiones, por lo que no "aprendería").

El proceso para la creación, modificación y eliminación de balizas sigue prácticamente el mismo flujo que el de los Chombos, con la diferencia de realizarse desde la vista del *render*. Por ser tan similar, no se realizó un nuevo diagrama pues sería redundante.

3.5.4. Caso de uso: parametrizar jardín

La parametrización del jardín (Ilustración 3.9) se realiza de forma similar a la edición de Chombos/balizas, pero en el prototipo actual de sistema únicamente existe un jardín. Por tanto, únicamente es necesario realizar una petición al Manager para alterar los parámetros del jardín enviando éstos en la llamada.

En un diseño inicial el sistema iba a disponer de una gestión de usuarios y jardines, donde cada usuario podía gestionar distintos jardines cada uno con sus elementos. Finalmente ésto se dejó fuera del alcance, pues únicamente hacía más compleja la implementación de las distintas partes del sistema y no era imprescindible a la hora de crear un prototipo funcional. La gestión de usuarios y jardines ha quedado como trabajo futuro, de cara a actualizar el sistema a una versión más comercial.

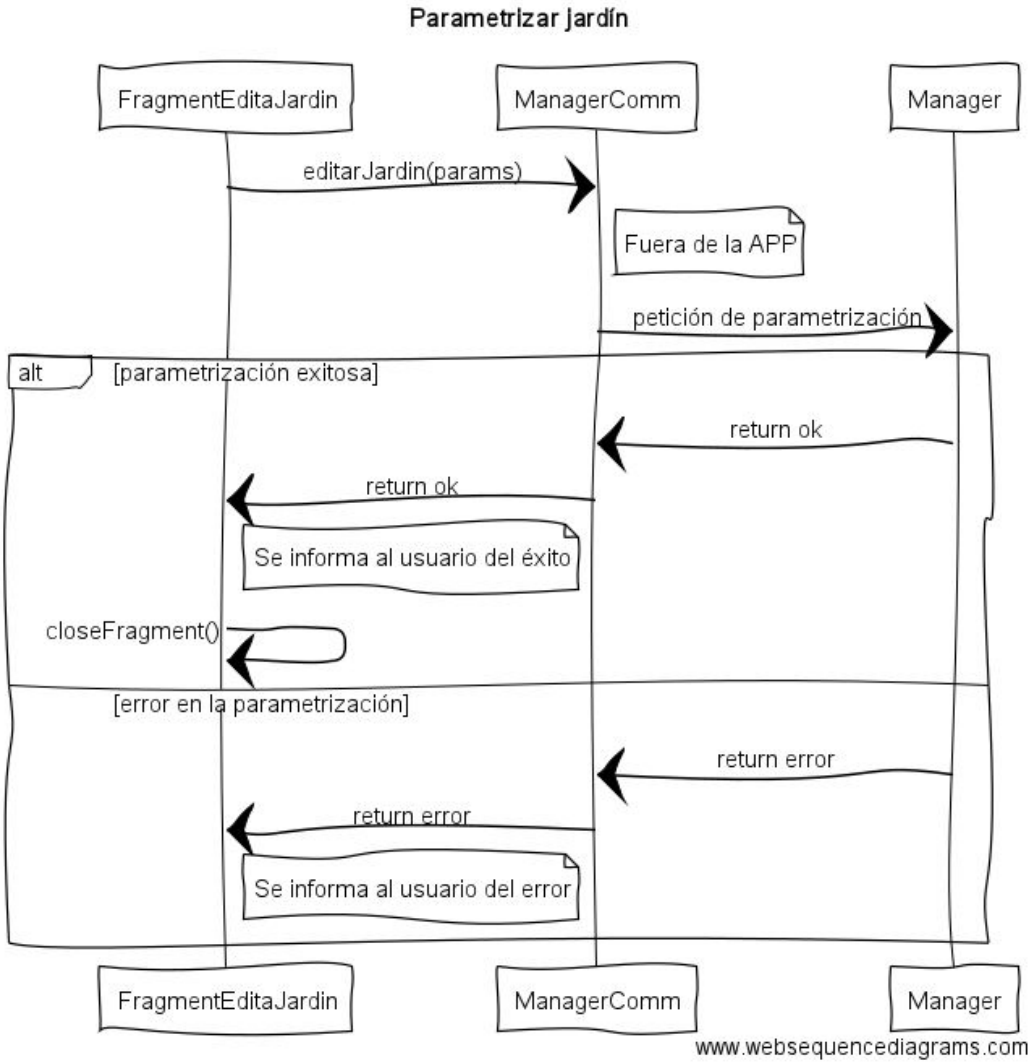


Ilustración 3.9. Parametrización del jardín

3.5.5. Caso de uso: edición de pesos de especie

El caso de la edición de pesos de pesos se produce prácticamente del mismo modo que la modificación de un Chombo o una baliza: se envía una petición al Manager con el identificador de la especie y los valores de parámetro tal y como requiera modificarse.

En un principio iba a ser algo más complejo, pues iba a poderse editar cualquier parámetro de la planta (incluso subir nuevas imágenes de la especie), pero finalmente se decidió dejar esto para trabajos futuros ya que era algo complementario al propósito del proyecto. Con el actual diseño únicamente se pretende editar los pesos de una especie; esto es, la importancia que tienen para un determinado tipo de planta parámetros como la temperatura, humedad o luminosidad, de forma que a la hora de tomar decisiones el Chombo sea capaz de hacerlo de acuerdo a una prioridad correcta preestablecida por el usuario experto.

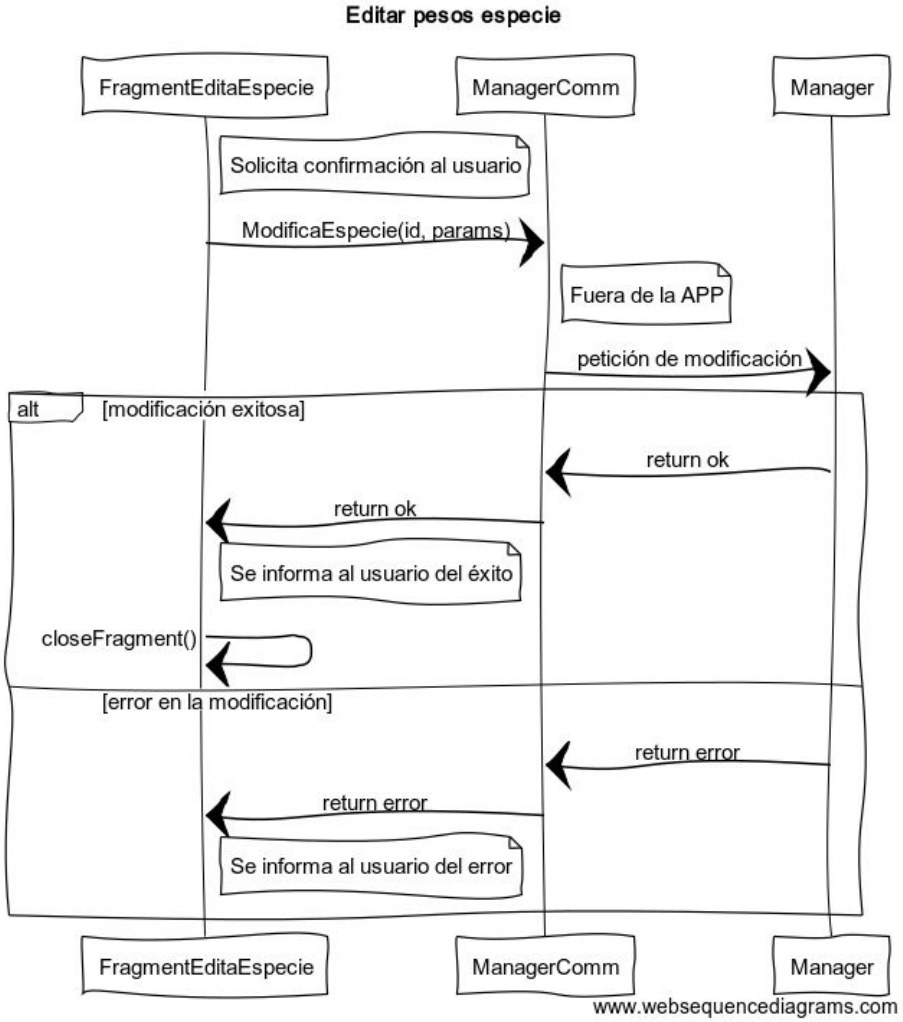


Ilustración 3.10. Editar pesos especie

3.5.6. Caso de uso: llamar Chombo a punto de encuentro

A diferencia del resto de conexiones basadas en casos de uso, la llamada de un Chombo al punto de encuentro (Ilustración 3.10) es una orden directa en lugar de una modificación o consulta de datos en las bases de datos. Sin embargo, desde el punto de vista de la APP esta orden se realiza de forma similar al resto de llamadas, con una petición al Manager (en la cual se envía el identificador del Chombo en caso de que únicamente quiera llamarse un Chombo, o sin identificador en caso de querer llamar a todos los Chombos del jardín).

Al igual que en las otras llamadas, en caso de que la petición no pueda procesarse correctamente se devolverá un error al usuario informando de ello.

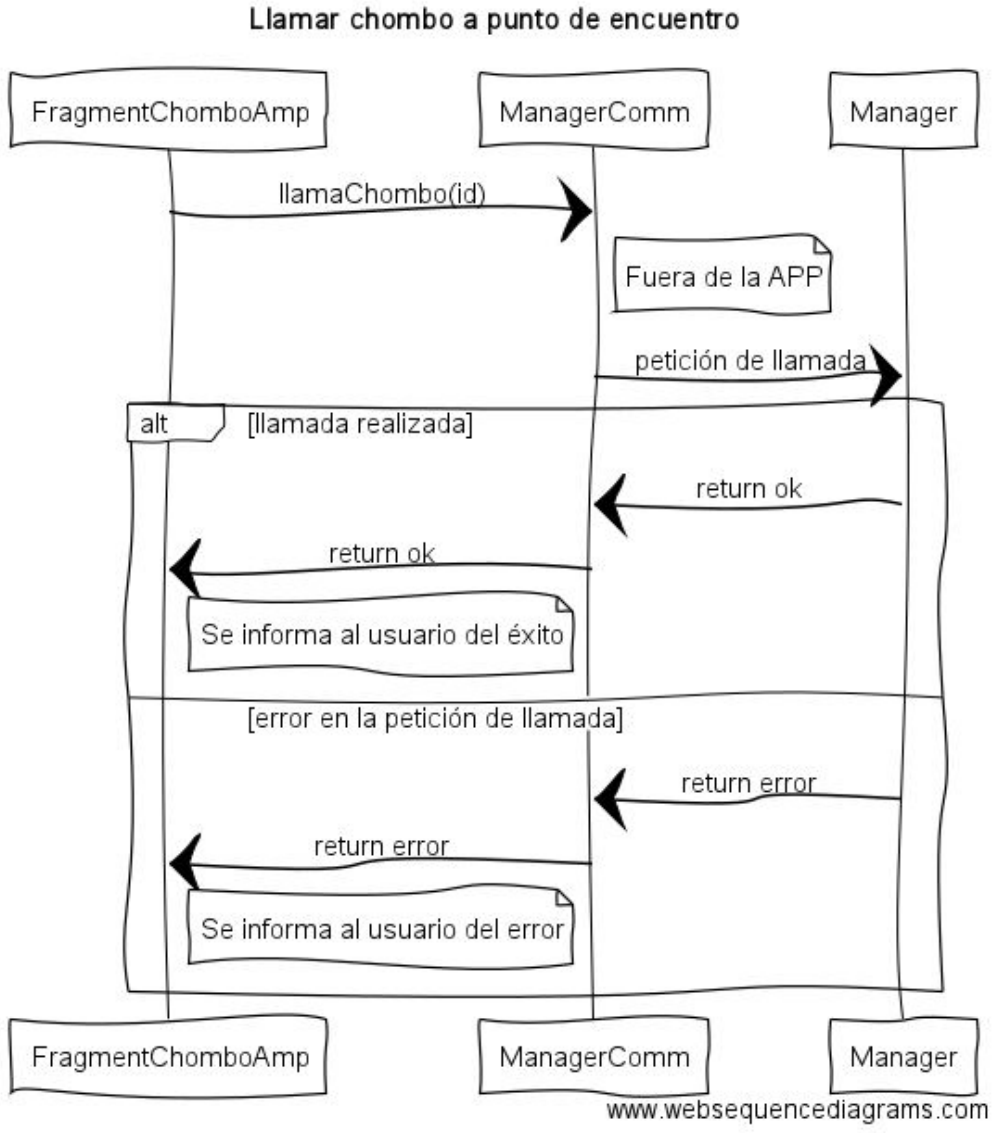


Ilustración 3.11. Llamar Chombo a punto de encuentro

3.5.7. Caso de uso: consultar estado actual de los Chombos

La consulta del estado actual de los Chombos (*Ilustración 3.12*), funciona de forma diferente a las anteriores llamadas, ya que no se dispara por una acción del usuario en la aplicación sino que al arrancar la APP se ejecuta una tarea asíncrona (*AsyncTask*) que realiza de forma contínua consultas al Manager para obtener los datos de los Chombos actualizados.

El fallo de esta petición es en principio transparente para el usuario. Sin embargo, si la petición falla *Y* veces seguidas (donde *Y* es un valor numérico entero que por el momento se ha fijado a 10) por problemas en el servidor o fallos de conexión se muestra un error para que el usuario tenga en cuenta que los datos que está visualizando podrían no estar ya sincronizados con la realidad.

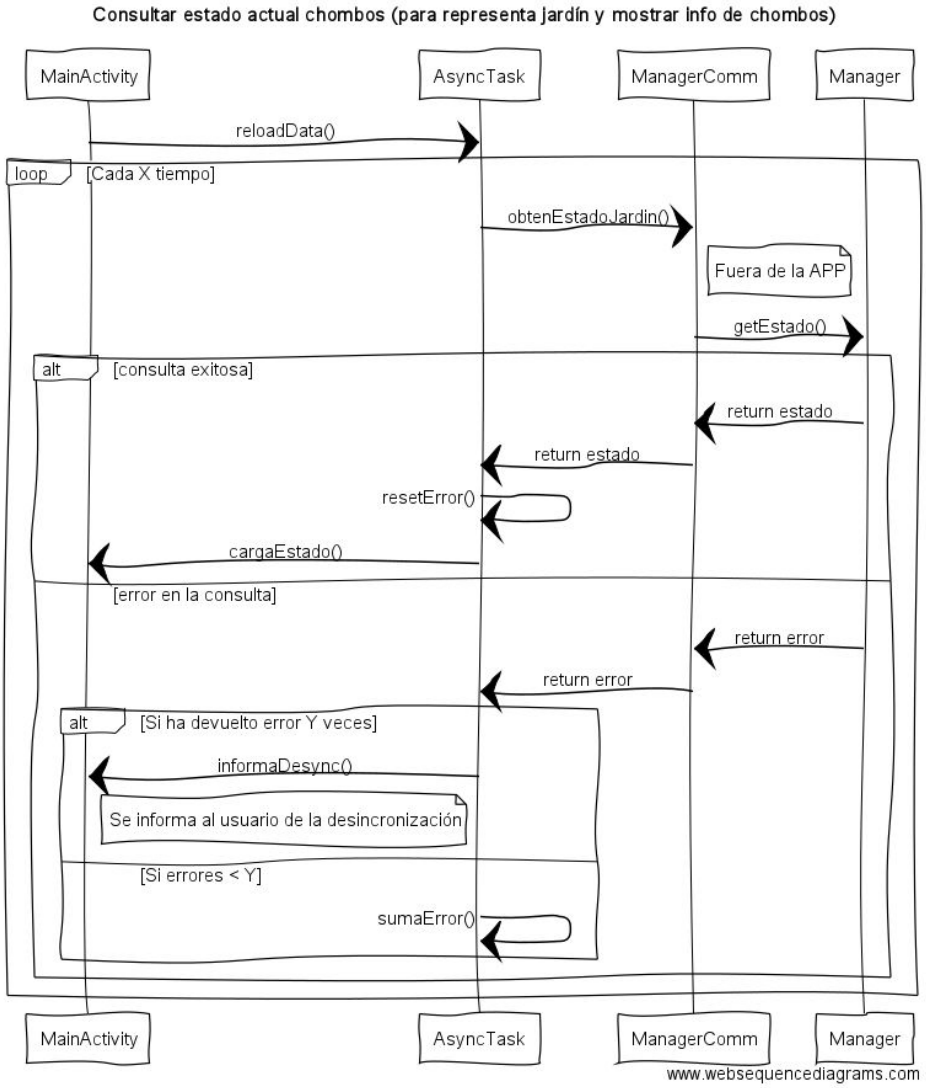


Ilustración 3.12. Consultar estado actual de los Chombos

3.5.8. Caso de uso: consultar histórico de un Chombo

Por último se tiene la consulta del histórico de un Chombo (*Ilustración 3.13*). A diferencia del resto de consultas, ésta no se realiza al Manager, sino que por medio de una API Rest se realiza una consulta directa a la BDD histórica con la que se obtienen últimos datos históricos de un Chombo. Por el momento se ha fijado el valor de entradas obtenidas a 10, de forma que cada vez que el usuario consulta desde la aplicación el histórico de un Chombo podrá visualizar en una gráfica los 10 últimos registros de cada uno de sus parámetros (temperatura, humedad, luminosidad…).

Al igual que con el resto de conexiones, si la consulta no puede realizarse de forma correcta la APP reporta el problema al usuario.

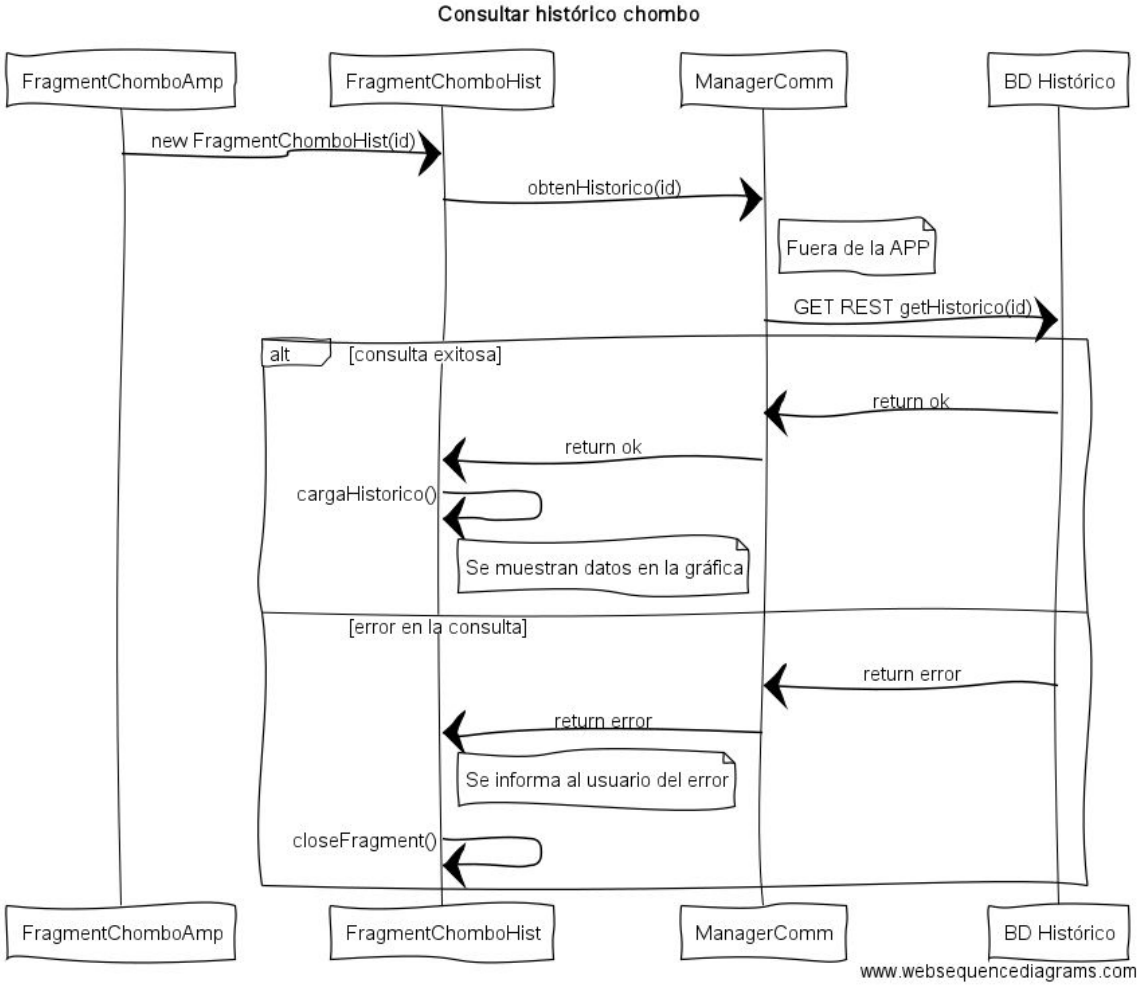


Ilustración 3.13. Consultar histórico de Chombo

4. Implementación

En este capítulo se va a detallar cómo se ha ido haciendo la implementación del prototipo de APP. Tal y como se ha mencionado en puntos anteriores, la aplicación se va a desarrollar para Android, usando el IDE Android Studio.

4.1. Desarrollo del esqueleto de la aplicación

Tal como se ha explicado en los apartados introductorios, la aplicación se decidió desarrollar en Android. El entorno a usar para ello es Android Studio, pues es el IDE oficial. El lenguaje en el que se realiza la programación es Java, lo cual es ideal porque se contaba con un buen dominio de éste.

También había que decidir una arquitectura, y ya que la más común para aplicaciones Android es el Modelo-Vista-Controlador, se decidió usar este mismo modelo. Esta arquitectura consiste en tres capas diferencias que interactúan entre sí:

- **Modelo**: es la capa de datos donde se gestiona la lógica de negocio y las conexiones de red que se usan para conectar con las distintas partes del sistema.
- **Vista**: interfaz de usuario, la capa que muestra de forma gráfica los datos contenidos en el modelo.
- **Controlador**: capa lógica que se encarga de notificar a la aplicación del comportamiento de los usuarios y actualiza el modelo cuando es necesario.

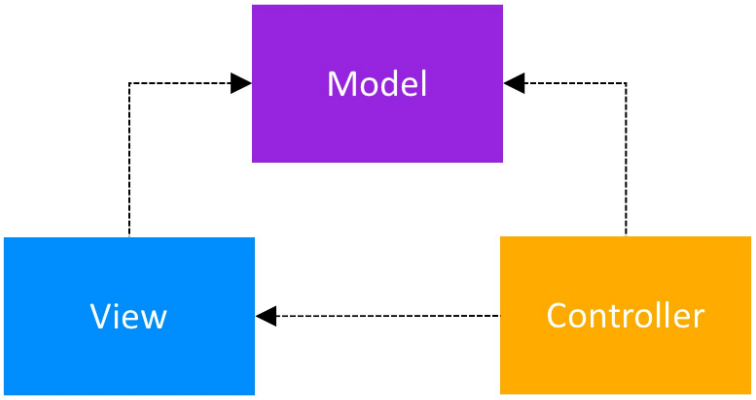


Ilustración 4.1. Arquitectura Modelo-Vista-Controlador

La aplicación de esta arquitectura resulta muy sencilla en Android porque la Activity junto con algunas clases de apoyo que llamamos Helpers constituyen de forma clara el Modelo. Por otra parte los Controladores son los Fragments, los cuales están fuertemente enlazados a cada una de las Vistas para procesar lo que en éstas ocurre.

Las vistas en Android se construyen a partir de ficheros XML como si de una plantilla se tratase, aunque pueden modificarse en tiempo de ejecución (algo que en esta APP se ha decidido hacer en los Fragments enlazados a las vistas). En cuanto a los Adapters, estarían mediando entre los Fragments y las vistas, por lo que podríamos considerarlos parte del Controlador.

Con la arquitectura ya clara, primero era necesario crear la Activity (nuestra clase principal) y la vista de la cajonera. Con la Activity y la cajonera ya enlazadas, es necesario crear también un primer Fragment que será el que se cargue en la Activity al iniciar la aplicación.

Se decidió que el Fragment que debería presentarse como principal sería el FragmentChombos (donde se mostraría la lista de Chombos añadidos ya y en funcionamiento en nuestro jardín). Una vez creado el FragmentChombos (todavía vacío y sin funcionalidad) y enlazado a la Activity ya era posible realizar un primer arranque de la APP y ver el diseño realizado (*Ilustración 4.2*).

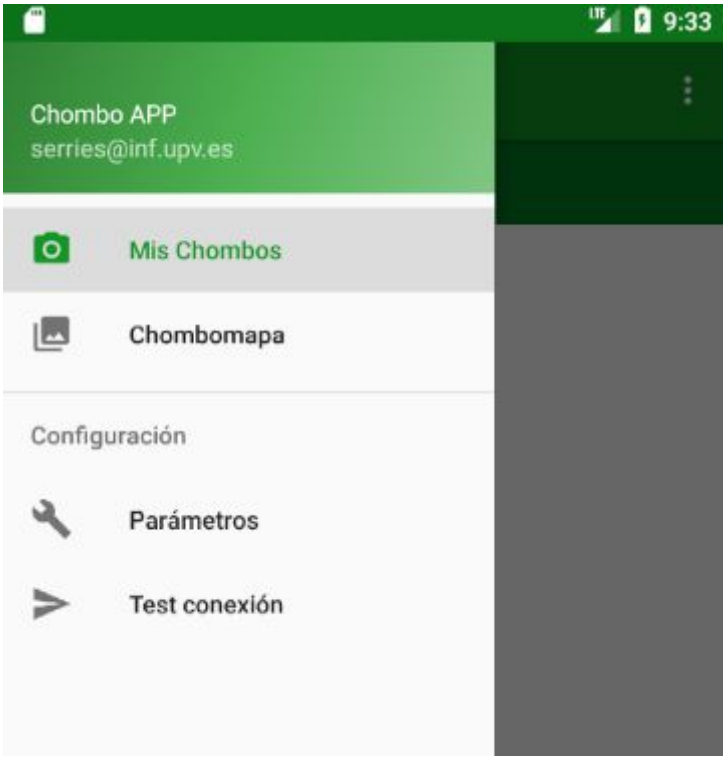


Ilustración 4.2. Cajonera con 4 ítems

4.2. Desarrollo de Helpers y Adapters

El primer Fragment a implementar es el FragmentChombos, donde se muestran todos los Chombos del jardín. Sin embargo, antes de poder realizar este desarrollo es necesario generar algunas clases Helpers y al menos un Adapter. La razón de esto es que vamos a llenar la vista de este Fragment con los distintos Chombos, pero todavía no se ha definido qué constituye un Chombo.

Creamos pues una clase **Chombo** donde se especifica cuáles van a ser los atributos de un Chombo y cómo se va a acceder a ellos. Los atributos son:

- **id**: Identificador del Chombo tal como estará registrado en las bases de datos.
- **name**: Nombre del Chombo, el cual lo decide el usuario en el momento de la **creación** del Chombo.
- **plant**: Identificador de la especie de planta que contiene el Chombo.
- **type**: Tipo de Chombo, ya que se ha pensado para una futura implementación tener distintos tipos de Chombo.
- **temp**: Temperatura actual del Chombo tal como la registra el sensor.
- **hum**: Humedad actual del Chombo tal como la registra el sensor.
- **lum**: Luminosidad actual detectada por el sensor lumínico del Chombo.
- **watMax**: Capacidad máxima del tanque de agua incorporado al Chombo, lo decide el usuario en el momento de su creación.
- **watLevel**: Nivel de agua actual contenido en el tanque.
- **battery**: Nivel de batería actual del Chombo, representada como un porcentaje.
- **img**: Imagen que se mostrará al usuario en el FragmentChombos para poder identificarlo de forma rápida.
- **posX**: Longitud en la que se encuentra el Chombo.
- **posY**: Latitud en la que se encuentra el Chombo.

Una vez creada la clase Chombo también se crean las clases **Beacon** (baliza) y Jardin, pues iban a ser necesarias también durante el desarrollo de los Fragments y vistas. La clase Beacon se crea con los siguientes atributos:

- **id**: Identificador de la baliza tal como estará registrada por el Manager.
- **name**: Nombre de la baliza que le asigna el usuario durante su creación y ayuda a identificarla, opcional.
- **type**: Tipo de baliza tal como la reconocerá el simulador (los distintos tipos de baliza se han detallado en el apartado 3.3 del presente documento).
- **posX**: Longitud en la que se encuentra la baliza.
- **posY**: Latitud en la que se encuentra la baliza.

A continuación se detallan los atributos de la clase **Jardin**. A destacar que en esta clase también ha sido necesario programar las funciones que permiten añadir o eliminar Chombos/balizas del jardín:

- **id**: Identificador del jardín. En la actual implementación este atributo no se usará, pues únicamente existe un jardín.
- **coorCenterX**: Longitud en la que se encuentra el centro del Jardín.
- **coorCenterY**: Latitud en la que se encuentra el centro del Jardín.
- **coorRadius**: Radio del jardín, usado para establecer qué superficie tiene el jardín.
- **chombos**: Lista de objetos Chombo que contiene el jardín.
- **beacons**: Lista de objetos Beacon que contiene el jardín.

Ya que es necesario crear instancias de Chombos y balizas para poder trabajar con ellos, en este punto también fue necesario crear la clase Factory. Esta clase permite crear todos los objetos que se vayan a requerir en la aplicación desde una única instancia de Factory mediante el patrón *singleton* (la cual se inicializa la primera vez que es llamada), permitiendo simplificar y mejorar la lectura del código ya que la inicialización de todos los parámetros de cada objeto se realizará dentro de ésta (lo cual es propio del patrón de diseño *factory*, como el nombre de la clase indica).

Es necesaria también una clase Adapter (llamada ChombosListAdapter) que se encargue de realizar el tratamiento de las lista de Chombos. Realmente la clase actúa como controlador de una vista donde se van a dibujar todos los

Chombos contenidos en la lista de nuestro jardín, y entre sus funciones se encuentran:

- Controlar cómo se obtienen los datos que se enlazan en la vista de la lista.
- Realizar el tratamiento de las imágenes a mostrar de las plantas.
- Obtener todos los Chombos de una lista de Chombos y notificar cuando haya cambios en esta lista.

Es necesario hacer lo propio para la lista de balizas. El funcionamiento será prácticamente idéntico, con la salvedad de que las balizas únicamente son representadas en el *render*.

4.3. Desarrollo de los Fragments, sus vistas y pruebas unitarias

Con el esqueleto de la aplicación ya montado y las clases auxiliares implementadas, ya es posible abordar el desarrollo de todas las distintas vistas y sus correspondientes Fragment, los cuales harán de controlador de éstas.

Durante el desarrollo de los Fragments también se han desarrollado clases de testeo para poder llenar con datos las listas y poder probar el módulo, pues sin ellas habría que esperar a realizar las pruebas de todos los módulos una vez estuviese terminada la parte de comunicaciones (lo cual será la última parte del desarrollo junto con las pruebas de integración).

4.3.1. FragmentChombos: Listado de Chombos del jardín

El FragmentChombos está enlazado a una vista la cual a su vez contiene otra vista. Esto es necesario, pues en la primera se realiza desde el Fragment la inserción de los datos de la lista de Chombos, mientras que en la segunda estos datos se separan en entradas individuales para mostrar los Chombos junto con sus datos haciendo uso del ChombosListAdapter que se ha creado previamente para colocarlos en la posición correcta.

Con el Fragment y sus vistas ya completado, se creó una clase auxiliar para insertar en la lista de Chombos algunos ejemplos (sin imágenes) y visualizar cómo se vería el resultado antes de poder introducir datos reales (*Ilustración 4.3*).

Ilustración 4.3. Vista del FragmentChombos

4.3.2. FragmentChomboAmp: Vista ampliada de datos del Chombo

El siguiente Fragment implementado fue el FragmentChomboAmp, cuyo papel es mostrar una vista ampliada del Chombo seleccionado en el FragmentChombos y permitir el acceso a otras secciones de la aplicación.

Además de diseñar la nueva vista y enlazarla al Fragment, era necesario crear el enlace entre los dos Fragments. En el FragmentChombos se tuvo que crear un proceso de tipo *Listener* el cual queda a la escucha de un cambio o evento para ejecutar una acción. En este caso, el evento a esperar es una pulsación por parte del usuario sobre uno de los Chombos de la lista, mientras que la acción a realizar es la apertura del nuevo FragmentChomboAmp.

Adicionalmente, es importante controlar el flujo de los Fragment es una aplicación Android, ya que el comportamiento adecuado al pulsar el botón de "Atrás/*Back*" en el móvil estando en la vista ampliada sería volver a la vista que contiene la lista de Chombos. Para ello, el FragmentChomboAmp se apila de forma que llamando a una función del FragmentManager (una clase de utilidades de los Fragments) y se desapila cuando pulsamos "atrás". También se

ha añadido un icono en la esquina superior derecha que cumple la misma función (*Ilustración 4.4*).

En este Fragment también se han incluído el acceso a la llamada de DirigirPuntoEncuentro (la cual se implementará en la parte de comunicaciones), el FragmentHistorico (implementado en el siguiente punto) y al FragmentEditaChombo, el cual tiene una vista réplica del FragmentChomboAmp a excepción de los dos botones inferiores, que cambian por un botón de "Confirmar edición" y un campo en el que introducir un nuevo nombre al Chombo. También estará en el menú desplegable la opción de eliminarlo.

Ilustración 4.4. Vista del FragmentChomboAmp

4.3.3. FragmentHistorico: Datos históricos de un Chombo

En la creación del FragmentHistorico y su vista asociada se empleó la librería Androidplot, de forma que al recibir los datos por REST (mediante la librería Retrofit, lo cual se implementaría más adelante) y organizados en listas se representen en la gráfica mediante distintas líneas cada uno de los atributos del Chombo visto a lo largo del tiempo.

Cada vez que se entra al FragmentHistorico se realiza una consulta REST para obtener los datos de histórico de dicha planta desde la BDD histórica, tal como se ha explicado en la sección de diseño de esta memoria. Los datos no se actualizarán en tiempo real mientras se permanezca en este Fragment y sólo se renovarán al entrar de nuevo a esta vista; esto es así por varios motivos:

1. La BDD histórica no está preparada para un tipo de servicio de consultas síncrono.
2. Las entradas de datos históricos en la BDD son introducidas cada 1-2 minutos, y no tiene mucho sentido mantener una comunicación síncrona con la BDD mientras se visualiza la gráfica de un Chombo por el gasto extra de batería que puede suponer en un dispositivo móvil.

En una primera implementación del FragmentHistorico se han tenido que introducir datos de muestra para probar que los datos de las listas se muestran correctamente (*Ilustración 4.5*), de forma que en el momento en que se implemente el Fragment ManagerComm únicamente haya que llenar la lista con datos obtenidos de la BDD histórica. Las unidades de medida usadas para mostrar los valores de cada una de las propiedades son las siguientes:

- **Temperatura**: Grados centígrados.
- **Humedad**: Porcentaje de humedad.
- **Luminosidad**: Lúmenes, aunque ha sido necesario realizar una conversión de forma que los valores de 0 a 5000 lúmenes se representen entre 0 y 100 para mejorar la legibilidad de la gráfica.
- **Fertilidad**: Usamos el PH de la planta.
- **Nivel de agua en tanque**: Litros de agua en el depósito.
- **Nivel de batería**: Porcentaje de batería restante.

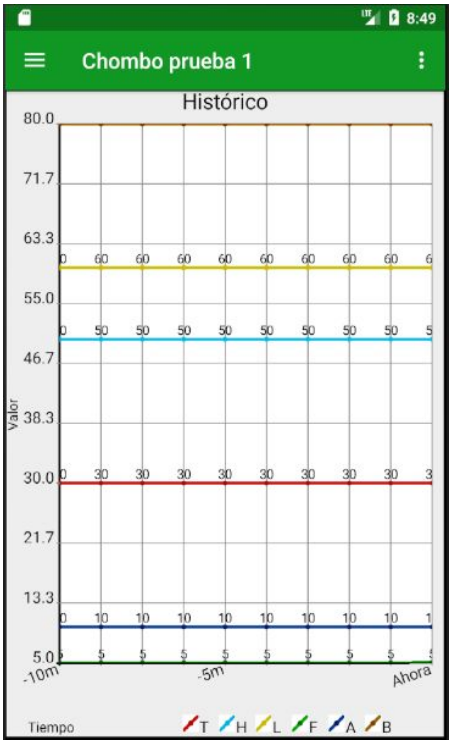


Ilustración 4.5. Vista del FragmentHistorico

4.3.4. FragmentCreaChombo: Creación de un Chombo nuevo

Para crear un nuevo Chombo se ha creado un Fragment desde el cual, después de seleccionar los parámetros deseados, se crea un nuevo Chombo siguiendo el flujo explicado en el punto 3.5.1 del documento. Los parámetros a configurar a la hora de crear un Chombo son:

- **Nombre**: El nombre que le de a un Chombo el usuario aparecerá en la lista de Chombos para identificarlo, así como en la cabecera de la aplicación cuando el usuario se encuentre dentro de la vista ampliada.
- **Tipo de planta**: El usuario seleccionará una de las especies de plantas contenidas en la base de casos, por lo que esta lista estará llena con los valores obtenidos del Manager.
- **Tipo de Chombo**: En este control el usuario puede seleccionar el tipo de Chombo a crear, aunque para este prototipo únicamente se contempla un tipo de Chombo con los sensores y actuadores mencionados anteriormente.

Esto módulo no ha podido ser probado hasta tener terminado la parte de comunicaciones, pues se requiere obtener la lista de tipos de planta a través del Manager y realizar la llamada de creación del Chombo una vez construído el nuevo objeto.

4.3.5. FragmentNuevaBaliza: Creación de una nueva baliza

La creación de la nueva baliza se realiza desde una vista simple similar a la del FragmentCreaChombo. Para la creación de la baliza únicamente es necesario especificar:

- **Nombre**: El nombre que el usuario le de a la baliza a fin de identificarla.
- **Tipo de baliza**: El tipo de baliza seleccionado en este menú determinará cómo será interpretada por el simulador.
- **Coordenada X**: Determinarán la posición que ocupará la baliza en el eje X (longitud). Se puede alterar arrastrando la baliza ya creada.

- **Coordenada Y**: Determinarán la posición que ocupará la baliza en el eje Y (latitud). Se puede alterar arrastrando la baliza ya creada.

Este módulo anterior tampoco puede ser probado por completo una vez implementado hasta no haber realizado la parte de comunicaciones, pues es necesario una vez creado el objeto Beacon realizar la llamada al Manager para incluirlo.

4.3.6. FragmentEditaJardin y FragmentEditaEspecies: Parametrización del jardín y modificación de pesos

Al igual que las dos anteriores vistas, la vista enlazada al FragmentSettings es un formulario con campos a completar que son usados posteriormente en una llamada al Manager. En el prototipo actual, únicamente hay tres parámetros en esta llamada:

- **Coordenada X**: Longitud en la que se encuentra el centro del Jardín.
- **Coordenada Y**: Latitud en la que se encuentra el centro del Jardín.
- **Radio**: Radio del jardín, usado para establecer qué superficie tiene el jardín.

Estos valores fueron establecidos con valores arbitrarios en un inicio, pues la entidad del jardín existe desde el momento en que se pone en marcha el sistema. No fue posible realizar pruebas exhaustivas de este Fragment hasta haber implementado las comunicaciones.

Desde la misma vista se puede acceder a una segunda vista formulario desde la cual se realiza la llamada para modificar los pesos de una especie tras seleccionar la especie de planta (del mismo modo que se realiza en la creación de un nuevo Chombo) e introduciendo los pesos deseados.

4.3.7. FragmentRender: Mapa del jardín y sus Chombos

En el FragmentRender se muestra un mapa generado con la API de Google Maps. Para generar el mapa ha sido necesario incluir las dependencias de Google Services y Google Maps en la aplicación y registrar una cuenta de Google para solicitar una clave de API.

Una vez configurada la clave de API en la aplicación fue posible construir el FragmentRender y su vista (la cual simplemente contiene el mapa). Desde el Fragment se introducen los datos a modo de *overlay*: el centro del jardín (sobre el cual se focaliza la cámara al entrar a la vista), las balizas y los Chombos. Todos ellos se obtendrán desde la conexión con el Manager una vez lista, pero para probar el módulo se han introducido marcadores fijos a modo de prueba (*Ilustración 4.6*).

Los iconos usados para representar cada uno de los elementos del jardín en el *render* son los siguientes:

- Centro del jardín
- Baliza de punto de encuentro
- Baliza barrera
- Baliza punto de recarga
- Baliza suministro de agua
- Chombo

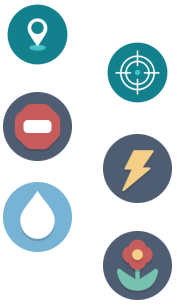


Ilustración 4.6. Vista del FragmentRender

4.4. Desarrollo del módulo de comunicaciones y pruebas de integración

La última fase en la implementación del prototipo es la construcción de las comunicaciones. Se ha dejado esta implementación en último lugar debido a lo complicado que es coordinar las pruebas, pues tanto el sistema de agentes como el módulo de persistencia debía estar en funcionamiento en el momento de estas.

Para las llamadas REST se ha usado la librería Volley. Ésta es usada para obtener el histórico de un Chombo desde el FragmentHistorico (la única comunicación que se produce fuera de la clase ManagerComm) y para descargar las imágenes de plantas con la URL obtenida desde el Manager.

El resto de las comunicaciones se realizan desde la clase ManagerComm, la cual sigue el patrón de diseño *singleton*. El protocolo de comunicación usado es UDP y para ello se han utilizado *sockets* haciendo uso de las clases de comunicación estándar de Java. Junto a la clase ManagerComm se ha diseñado un FragmentTestConexion y una vista en forma de registro con los que se han probado todos los casos de uso sobre datos previamente introducidos (*Ilustración 4.7*). También ha sido necesario implementar *AsyncTask* para realizarllamadas desde ManagerComm al Manager de una forma asíncrona, al mismo tiempo que el usuario interactúa con la aplicación.

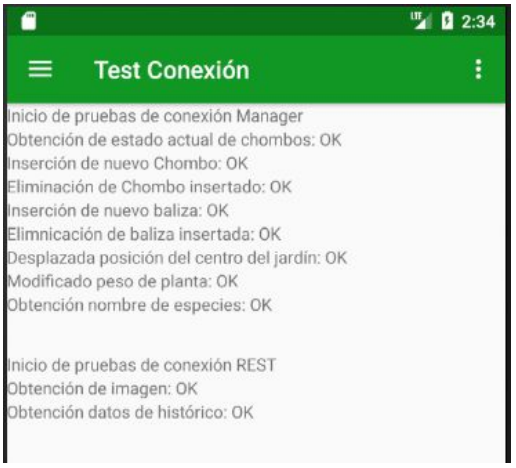
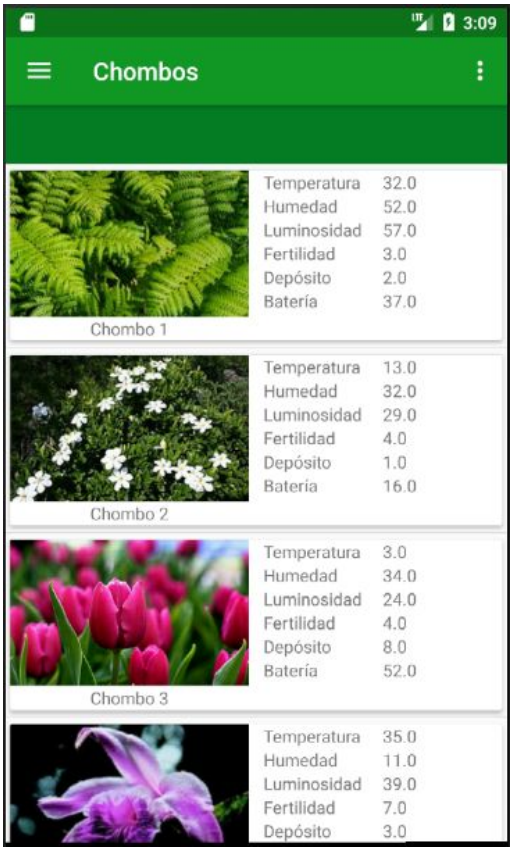
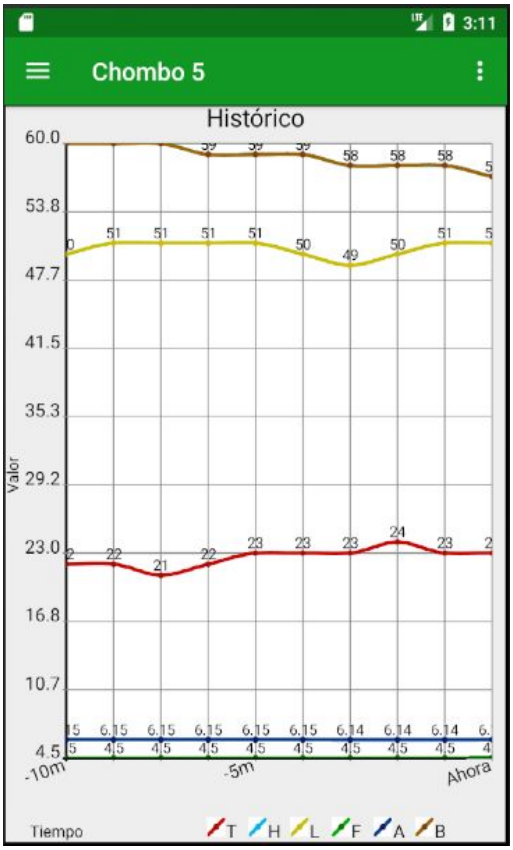
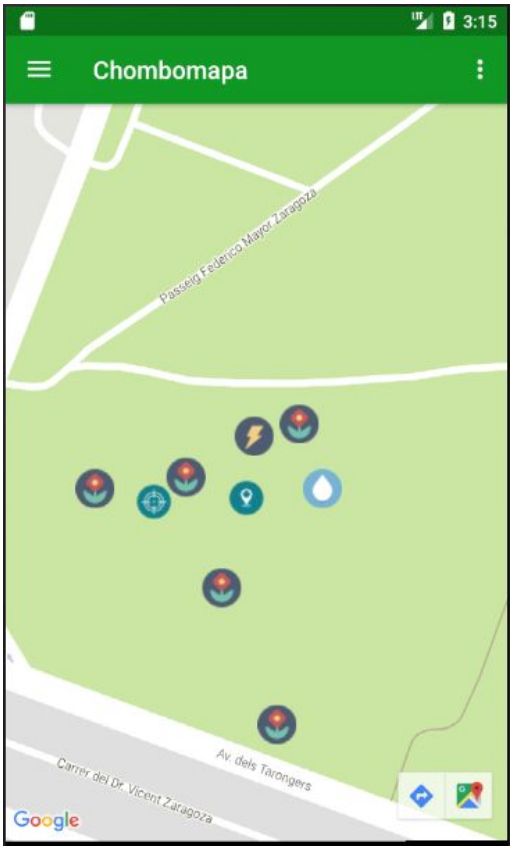


Ilustración 4.7. Vista del FragmentTestConexion

Una vez aseguradas las conexiones se han realizado pruebas funcionales con el prototipo completo. En las *Ilustraciones 4.8, 4.9, 4.10 y 4.11* se pueden apreciar las vistas con datos obtenidos desde el Manager y la BDD histórica.







Ilustraciones 4.8, 4.9, 4.10 y 4.11. Vistas con datos obtenidos del Manager y la BDD histórica

5. Conclusiones

A lo largo de este documento se han presentado las soluciones consideradas en base a las decisiones que se han tomado durante el diseño e implementación del proyecto. A continuación presentamos los objetivos que se han alcanzado y algunas propuestas para futuras ampliaciones y mejoras del sistema a fin de pasar de un prototipo a un producto más comercial.

5.1. Objetivo individual

Se ha creado un prototipo de aplicación móvil con la cual monitorizar e interactuar con el funcionamiento del sistema encargado en el mantenimiento y cuidado de plantas con recipientes robotizados (Chombos).

La aplicación es capaz de tomar información en tiempo real a través del Manager y representarla en la interfaz de usuario. Esta información incluye los valores obtenidos por la sensorización del Chombo: iluminación, temperatura, humedad, fertilidad, nivel de agua en depósito y nivel de batería. Además es capaz también de consultar una base de datos histórica para obtener los datos correspondientes a los últimos estados en los que se ha encontrado un Chombo y representarlos en una gráfica.

En la aplicación también se incluye un *render*, donde se representa la posición de unos Chombos respecto a otros dentro de su entorno en un mapa. En este mapa también se representa la posición de las diferentes balizas que hayan sido introducidas en el entorno.

El prototipo, además de ser capaz de monitorizar el estado del sistema, permite al usuario la interacción con este. Desde la aplicación es posible crear o modificar elementos (Chombos y/o balizas), incluso eliminarlos. También es posible llamar a los Chombos a un punto de encuentro establecido dentro del jardín, para que podamos fácilmente localizar el Chombo en caso de que tengamos que realizar algún tipo de mantenimiento.

5.2. Trabajo futuro

Durante el diseño e implementación del proyecto han surgido ideas que no han entrado dentro del alcance pero sería interesante tenerlas en cuenta para una futura versión, ya que lo mejorarían de forma sustancial de cara a tener un producto más comercial y completo.

5.2.1. Trabajo futuro de la interfaz de usuario

A pesar de haberse desarrollado el actual prototipo para la plataforma Android, la estructura modular del sistema global permitiría conectar una nueva aplicación desarrollada para cualquier tipo de plataforma. Uno de los trabajos futuros posibles pues sería la recreación de la aplicación para otras plataformas, ya que contar con una aplicación multiplataforma permitiría que los usuarios pudieran controlar la aplicación desde cualquiera de sus dispositivos.

Una forma de obtener aún mayor control sobre la monitorización de las plantas sería tener un sistema de alertas. Esto permitiría no tener que abrir la aplicación en el dispositivo para comprobar si algo va mal con uno de los Chombos, pues recibiríamos una notificación que nos mostraría el problema (el Chombo ha volcado, ha quedado sin batería y no ha podido cargarla, los Chombos no pueden acceder al suministro de agua, etc). De esta forma el sistema podría funcionar de forma más desatendida incluso.

El prototipo también puede recibir algunos cambios *QOL* en su interfaz de usuario: cambiar algunas etiquetas por iconos personalizados similares a los de los mockup iniciales ayudaría a identificar de forma más rápida los elementos, inclusión de gestos para una navegación más rápida, introducir filtros de búsqueda, dotarla de capacidad para subir fotos propias al servidor, permitir el visionado en streaming de la planta a través de su cámara cenital, disponer de conexión con redes sociales, etc.

Así mismo, dotar a la aplicación de multilenguaje y adaptarla a cualquier tipo de resolución ayudaría a que pudiera ser acogida por un mayor público. De realizarse pruebas de usabilidad con un grupo de usuarios sería posible descubrir más mejoras que los usuarios ven necesarias en la aplicación.

Podría ser interesante disponer de un render en tres dimensiones que pudiera representar la diferencia de altura entre los Chombos y rotar el punto

de vista para obtener una mejor perspectiva, aunque habría que tener en cuenta la carga extra de procesamiento que esto exigiría a los dispositivos.

Por último, antes de alcanzar una versión comercial sería necesario implementar una gestión de usuarios, lo cual permitiría gestionar un mismo jardín desde cualquier dispositivo siempre que se dispusiera de las credenciales correctas. Sería interesante incluso que un mismo usuario pudiese gestionar distintos jardines desde su cuenta, ya que la versión actual está limitada a un jardín, y que existieran distintos tipos de usuario para que los usuarios expertos pudieran realizar una parametrización más al detalle del sistema. Para lograr este objetivo deberían hacerse cambios a lo largo del todo el sistema (persistencia, sistema de agentes e interfaz de usuario), pero es algo imprescindible para la creación de un producto comercial.

5.2.2. Trabajo futuro del sistema global

Para un mejor análisis del entorno sería interesante disponer de un sistema de aprendizaje automático a fin de mejorar la toma de decisiones de los Chombos. Este es el principal motivos por el que se diseñó la BDD que almacena los datos históricos, ya que a posteriori será posible realizar el análisis de los datos para lograr alcanzar este objetivo.

Otro punto interesante podría ser el análisis automático de los logs relacionados con el razonamiento basado en casos, de forma que se pueda extraer el conocimiento del cuidado de plantas como un flujo de procesos mediante minería de procesos.

La mayor limitación del sistema actual es el disponer de la entidad servidor, pues es un único punto de fallo para el sistema (aunque los agentes físicos sigan trabajando, se corta toda la comunicación con los usuarios y la base de casos). Lo ideal sería establecerse de forma distribuida, ya sea usando replicación o completa distribución entre los agentes.

Con un sistema distribuido, la comunicación podría realizarse desde los propios agentes, lo cual mejoraría su coordinación al informar directamente de sus propias condiciones al resto del sistema, sin necesidad de pasar por el simulador (el cual podría desaparecer llegado a este punto).

Siguiendo con la propuesta de distribuir el sistema, sería interesante también disponer una base de datos distribuida y escalable horizontalmente. Además como mejora para el acceso por parte de los agentes se podría realizar

una división de la información más acercada a los interesados que van a utilizarla.

Una forma de facilitar el despliegue del sistema completo podría ser el uso de la plataforma Docker, dado que nos permite automatizar la inyección de dependencias del sistema y poner en marcha todos los componentes mediante Docker-compose.

Una vez generado este primer prototipo es necesaria una evaluación de los costes de comunicación en entornos reales, dependiendo del resultado podría ser interesante el uso de middlewares que optimicen la velocidad, el ancho de banda utilizado o la pérdida de datos.

6. Referencias

1. Delgado Sanchis, A. (2016). Comunicación entre módulos para la construcción de jardines inteligentes. Universidad Politécnica de Valencia, ETSINF.

2. Guzmán Godia, A. (2016). Informatización de un módulo para la construcción de jardines inteligentes. Universidad Politécnica de Valencia, ETSINF.

3. Gil Rodríguez, C. (2016). Desarrollo de aplicación web para gestionar módulos inteligentes para la construcción de jardines. Universidad Politécnica de Valencia, ETSINF.

4. Ferrando Ferragut, L. (2016). Monitorización y control de los módulos de un jardín inteligente. Universidad Politécnica de Valencia, ETSINF.

5. Martín Taberner, A. (2017). Desarrollo de un simulador basado en agentes para la gestión inteligente de un entorno paisajístico. Universidad Politécnica de Valencia, ETSINF.

6. Ferrer Mestre, R. (2017). Desarrollo del gestor de datos y casos para el soporte a un sistema multiagente para un entorno paisajístico. Universidad Politécnica de Valencia, ETSINF.

7. https://developer.android.com/develop/index.html server support. [online] Available at: https://developer.android.com/ [Accessed 15 Jul 2017].

8. https://bitbucket.org/androidplot/androidplot/ server support. [online] Available at: https://bitbucket.org/androidplot/androidplot/src/a3c4ca3bb01e0095a7358473b3f02ed731debe3b/docs/quickstart.md?fileviewer=file-view-default [Accessed 5 Jul 2017].

9. http://square.github.io/retrofit/ server support. [online] Available at: http://square.github.io/retrofit/ [Accessed 20 May 2017].

10. Barella, A. Ricci, O. Boissier, and C. Carrascosa. (2012) MAM5: Multi-Agent Model For Intelligent Virtual Environments. In 10th European Workshop on Multi-Agent Systems. EUMAS. p.16-30.

11. J.A. Rincon, C. Carrascosa and Emilia Garcia. (2014) Developing Intelligent Virtual Environments using MAM5 Meta-Model International Conference on Practical Applications of Agents and Multi-Agent Systems pp. In Press.

12. J.A. Rincon, Emilia Garcia, V. Inglada and C. Carrascosa. (2014) Developing Adaptive Agents Situated in Intelligent Virtual Environments International Conference on Hybrid Artificial Intelligence System pp. In Press.

13. https://docs.oracle.com/javase/8/docs/api/. (2016). server support. [online] Available at: https://docs.oracle.com/javase/7/docs/api/java/awt/event/ActionListener.html [Accessed 15 Jun 2017].

14. http://www.emse.fr/~boissier/enseignement/maop11/doc/cartago-main-api/index.html. server support. [online] Available at: http://www.emse.fr/~boissier/enseignement/maop12/doc/c4jason-api/cartago/invoke_obj.html [Accessed 25 Jun 2017].

15. http://www.telegraph.co.uk. [online] Available at: http://www.telegraph.co.uk/gardening/tools-and-accessories/the-best-apps-to-identify-unknown-plants-and-flowers/ [Accessed 1 Dec 2016].

16. http://blog.parrot.com. (2015) [online] Available at: http://blog.parrot.com/2015/01/05/ces-2015-flower-power-pot-the-smart-pot-that-grows-healthy-plant/ [Accessed 1 Dec 2016].

17. https://www.alibaba.com [online] Available at: https://www.alibaba.com/showroom/smart-flower-pot.html [Accessed 1 Dec 2016].

18. http://es.aliexpress.com [online] Available at: http://es.aliexpress.com/popular/smart-flower-pot.html [Accessed 1 Dec 2016].

19. https://www.google.com [online] Available at: https://www.google.com/patents/US3961444 [Accessed 2 Dec 2016].

20. https://www.google.com [online] Available at: https://www.google.com/patents/US4403443 [Accessed 2 Dec 2016].

21. https://www.google.com [online] Available at: https://www.google.com/patents/US6070359 [Accessed 2 Dec 2016].

22. https://www.google.com [online] Available at: https://www.google.com/patents/US4108350 [Accessed 2 Dec 2016].

23. https://www.google.com [online] Available at: https://www.google.com/patents/US3069807 [Accessed 2 Dec 2016].

24. https://www.google.com [online] Available at: https://www.google.com/patents/US20060272210 [Accessed 2 Dec 2016].

25. https://www.google.com [online] Available at: https://www.google.com/patents/US6243986 [Accessed 2 Dec 2016].

26. http://www.actahort.org [online] Available at: http://www.actahort.org/members/showpdf?booknrarnr=417_4 [Accessed 4 Dec 2016].

27. http://www.fliwer.com [online] Available at: http://www.fliwer.com/# [Accessed 4 Dec 2016].

28. http://delivery.acm.org [online] Available at: http://delivery.acm.org/10.1145/1360000/1357335/p1797-consolvo.pdf?ip=158.42.174.22&id=1357335&acc=ACTIVE%20SERVICE&key=DD1EC5BCF38B3699%2E016407C0B79CBB66%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35&CFID=677700695&CFTOKEN=86003516&__acm__=1475767424_edd7e7a77bbde675bf3f0ef7fc1ec83d [Accessed 4 Dec 2016]

29. Ministerio de cultura y pesca. (2013) Proyecto sigAGROasesor. Available at: http://agroasesor.es/es/ [Accessed 7 Dec 2016]

30. Emmi Pizzella, L. A. (2011). Entorno de simulación para flota de robots orientado a la gestión de malas hierbas en cultivos. Universidad Complutense de Madrid, Máster en Investigación en Informática, Facultad de Informática, Departamento de Ingeniería del Software e Inteligencia Artificial.

31. https://myflowerpower.parrot.com. (2016) [online] Available at: https://myflowerpower.parrot.com/#plantdb [Accessed 10 Dec 2016].

32. https://developer.android.com [online] Available at: https://developer.android.com/training/volley/index.html [Accesed 20 Jul 2017]

33. https://www.iconfinder.com/ [online] Available at: https://www.iconfinder.com/ [Accesed 10 Aug 2017]

34. https://www.websequencediagrams.com/ Available at https://www.websequencediagrams.com/ [Accesed 12 Jun 2017]

7. Anexo

7.1. Glosario

- **API**: Application Programming Interface, es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.
- **APP**: Acrónimo de la palabra aplicación, al que se recurre para referirse a una aplicación móvil.
- **Activity**: Tipo de clase Java usada en la programación en Android que extiende de la clase AppCompatActivity. Una instancia de esta clase representa una ventana de la APP.
- **Actuador**: Dispositivo capaz de transformar energía eléctrica en la activación de un proceso con la finalidad de generar un efecto sobre un proceso automatizado.
- **Adapter**: Tipo de clase Java que extiende la clase Adapter y se encarga de integrar un tipo de datos empaquetados en un objeto en una vista determinada.
- **Agente**: Aplicación informática con capacidad para decidir cómo actuar para alcanzar sus objetivos.
- **BDD**: Acrónimo del término base de datos.
- **Cajonera**: Elemento muy usado en las aplicaciones Android, es un menú lateral que permite moverse entre distintas secciones de la aplicación desde la mayoría de puntos de esta.
- **Chombo**: Contenedor de una o varias plantas, con sensores, actuadores y capacidad de movimiento, que es controlado por un agente.
- **ChomboAPP**: Aplicación móvil que se encargará de la monitorización e interacción del sistema.
- **Clase**: Término usado cuando se habla de Java para referirse a un molde a partir del cual pueden crearse objetos.
- **Factory**: Es un patrón de diseño consiste en utilizar una clase constructora abstracta con unos cuantos métodos definidos y otros

abstractos: el dedicado a la construcción de objetos de un subtipo de un tipo determinado.

- **Fragment**: Tipo de clase Java que extiende la clase Fragment. Este tipo de instancias se suelen inyectar dentro de un Activity para añadir elementos de interfaz con nuevas funcionalidades. Su nombre viene de "fragmento", pues no es más que un fragmento de una Activity.
- **Helper**: Clase Java en la que se apoyan otras clases, similar a una clase Utility. La diferencia principal es que una clase Helper puede ser instanciada.
- **Jardín**: Entorno donde una red/agrupación de Chombos que comparten un espacio de actividad.
- **Java**: Lenguaje de programación multiplataforma orientado a objetos e imperativo.
- **Librería**: También llamada Bibloteca, es un conjunto de implementaciones funcionales, codificadas en un lenguaje de programación, que ofrece una interfaz bien definida para la funcionalidad que se invoca. Su fin es ser utilizada por otros programas.
- **Listener**: Conocidos como oyentes o escuchadores en español, se encargan de controlar los eventos, esperan a que el evento se produzca y realiza una serie de acciones.
- **Manager**: Middleware encargado de conectar entre sí los módulos de persistencia de la base de casos, el sistema multiagente y
- **Objeto**: Paquete de datos que tiene una estructura y sentido.
- **Overlay**: Capa de datos superpuestos a un elemento.
- **Protocolo**: Sistema de reglas que permiten que dos o más entidades de un sistema de comunicación se comuniquen entre sí para transmitir información.
- **QOL changes**: Acrónimo de Quality of life, los cambios de este tipo están principalmente destinados a mejorar la usabilidad.
- **REST**: Estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web, en el que cada mensaje HTTP contiene toda la información necesaria para comprender la petición.

- **Render**: Parte de la aplicación donde se representa de forma gráfica las posiciones de unos elementos respecto de otros dentro del entorno paisajístico o jardín.
- **Sensor**: Dispositivo capaz de detectar magnitudes físicas y transformarlas en valores binarios.
- **Simulador**: Reproducción del sistema en un entorno con unas características específicas.
- **Singleton**: También llamado "patrón de instancia única", es un patrón de diseño que permite restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella, ahorrando memoria al impedir que se cree innecesariamente más de una instancia.
- **Sistema multiagente**: Modelo de programación en el cual entidades virtuales (llamadas agentes) toman el papel de los usuarios, tomando control de las acciones necesarias que se deben llevar a cabo para cumplir los objetivos marcados. Pudiendo crear nuevos objetivos y variaciones para cada caso particular, en función del entorno.
- **Sistema**: Se usa este término para referirse globalmente a todas las partes que componen el proyecto, desde la APP y el sistema de agentes hasta los sensores y actuadores, pasando por las conexiones y el sistema de persistencia.
- **Timeout**: Error que se produce cuando una comunicación queda interrumpida por exceder el tiempo de espera de respuesta.
- **Vista**: Representación de datos tal como va a verlos el usuario en la interfaz.