



ugr | Universidad
de **Granada**

TRABAJO DE FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA

Observatorio Remoto: Un cliente INDI para Android

Autor

Jaime Torres Benavente

Tutor

Dr. Sergio Alonso Burgos



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

Granada, 11 de septiembre de 2015



Observatorio Remoto: Un cliente INDI para Android

Jaime Torres Benavente

Yo, **Jaime Torres Benavente**, alumno de la titulación **Grado en Ingeniería Informática** de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación** de la **Universidad de Granada**, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado (*Observatorio Remoto: Un cliente INDI para Android*) en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Además, este mismo trabajo es realizado bajo licencia **Creative Commons Attribution-ShareAlike 4.0** (<https://creativecommons.org/licenses/by-sa/4.0/>), dando permiso para copiarlo y redistribuirlo en cualquier medio o formato, también de adaptarlo de la forma que se quiera, pero todo esto siempre y cuando se reconozca la autoría y se distribuya con la misma licencia que el trabajo original. El documento en formato **LaTeX** así como el código del proyecto se puede encontrar en el siguiente repositorio de **GitHub**: <https://github.com/torresj/indi-android-ui>.

El estilo de este documento está extraído a partir de la plantilla **LaTeX** publicada por la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación, y la plantilla **LaTeX** de Germán Martínez Maldonado, publicada bajo licencia **Creative Commons Attribution-ShareAlike 4.0** (<https://creativecommons.org/licenses/by-sa/4.0/>) que se encuentra en <https://github.com/germaaan/TFG>



Fdo: Jaime Torres Benavente

Granada, a 11 de septiembre de 2015

Dr. Sergio Alonso Burgos, profesor del Departamento de Lenguajes y Sistemas Informáticos de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Observatorio Remoto: Un cliente INDI para Android*, ha sido realizado bajo su supervisión por **Jaimé Torres Benavente**, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 11 de septiembre de 2015.

El tutor:

A handwritten signature in blue ink that reads "Sergio A". The "S" is large and stylized, with a small "o" inside it. The "e" has a long horizontal stroke extending to the right. The "r" is a simple vertical line with a small loop at the top. The "g" is a large, open loop. The "i" is a short vertical line with a dot above it. The "o" is a small circle. The "n" is a short vertical line with a small loop at the top. The "t" is a short vertical line with a small loop at the top. The "o" is a small circle. The "r" is a short vertical line with a small loop at the top.

Dr. Sergio Alonso Burgos

Agradecimientos

A mi familia porque gracias a ellos soy como soy.

A mis amigos porque sin ellos no habría podido llegar hasta aquí.

A mi tutor Sergio, por aguantarme durante tantos meses con una paciencia infinita.

Índice general

1. Resumen	1
1.1. Breve resumen y palabras clave	1
1.2. Extended abstract and key words	3
2. Introducción	5
2.1. La Astronomía	5
2.2. Instrumental Astronómico	8
2.2.1. Telescopios	8
2.2.2. Cámaras CCD	9
2.2.3. Monturas	11
2.2.4. Rueda portafiltros	12
2.2.5. Enfocadores	12
2.2.6. Cúpulas	13
2.2.7. Ópticas activas/adaptativas	14
2.2.8. Estaciones meteorológicas	15
2.3. Control de dispositivos astronómicos	16
2.4. INDI	17
2.4.1. Breve introducción a INDI	18
2.4.2. Drivers, Servidores y Clientes INDI	19
2.4.3. Abstracción de los Dispositivos	20
2.4.4. Ejemplo de Abstracción de un Dispositivo	21
2.4.5. INDI for Java	22
2.5. Dispositivos Móviles	23
2.5.1. iOS	23
2.5.2. Android	24
3. Objetivos	27
3.1. Alcance de los objetivos	29
3.2. Interdependencia de los objetivos	29
4. Planificación	31
4.1. Fases	31
4.2. Estimación de tiempos	32

4.3. Recursos humanos	33
4.4. Presupuesto	34
4.5. SCRUM	34
4.6. Temporización	35
5. Análisis	39
5.1. Análisis de requisitos	39
5.1.1. Requisitos funcionales	39
5.1.2. Requisitos no funcionales	40
5.2. Casos de uso	40
5.2.1. Descripción de actores	41
5.2.2. Descripción casos de uso	41
5.3. Diagramas de casos de uso	66
5.4. Storyboard	67
6. Diseño e Implementación	69
6.1. Diseño de clases	69
6.1.1. Diseño de las actividades de Android	70
6.1.2. Diseño del cliente INDI	72
6.1.3. Diseño de las clases manejadoras de propiedades	74
6.1.4. Diseño de las clases manejadoras de dispositivos	76
6.2. Diseño de las interfaces de usuario	76
6.2.1. Interfaz principal de la aplicación	79
6.2.2. Interfaz para listar las propiedades	80
6.2.3. Interfaz de propiedad	80
6.2.4. Interfaz de diálogos	82
6.2.5. Interfaz de log	86
6.2.6. Interfaz de los ajustes	87
6.3. Mecanismo de adición de vistas	88
6.3.1. Propiedades	89
6.3.2. Dispositivos	94
6.4. Implementación	95
7. Pruebas	97
7.1. Pruebas Unitarias	97
7.1.1. Conexión INDI	98
7.1.2. Listas de propiedades	99
7.1.3. Manejadores de propiedades	99
7.1.4. Creación de ficheros de configuración	100
7.2. Pruebas de integración	101
7.2.1. Conexión con un servidor	101
7.2.2. Interacción con las propiedades	102
7.2.3. Gestión de las conexiones	103
7.2.4. Ajustes generales	103

7.3. Pruebas de sistema	105
7.4. Pruebas de aceptación	105
8. Difusión del proyecto	107
8.1. Página web del proyecto	107
8.2. Publicación en Google Play	110
8.3. Mantenimiento del repositorio en GitHub	110
8.4. Difusión en la página oficial de la biblioteca INDI	110
8.5. Difusión en el foro oficial de la biblioteca INDI	112
8.6. Algunas ideas futuras para la difusión	112
9. Conclusiones y trabajos futuros	115
A. User Manual	117
A.1. Preface: A user manual for a mobile app? Are you kidding me?	117
A.2. Starting the Application	117
A.3. Main Screen Description	117
A.4. Managing Connections	118
A.5. Adjusting Settings	119
A.6. Logging	121
A.7. Connecting and Disconnecting from INDI Servers	121
A.8. Device Views	123
A.9. Updating Properties	126
A.10. Hidding / Showing Properties	127
B. Manual del desarrollador	131
B.1. Mecanismo de adición de vistas	131
B.1.1. Propiedades	131
B.1.2. Dispositivos	145
Bibliografía	149

Índice de figuras

2.1. Telescopio	8
2.2. Cámara CCD	10
2.3. Montura	11
2.4. Rueda portafiltros	13
2.5. Enfocador	13
2.6. Cúpula	14
2.7. Óptica adaptativa	15
2.8. Estación meteorológica	16
2.9. ASCOM Standard	17
2.10. INDI Client	18
2.11. INDI Server	19
4.1. Proceso de desarrollo Scrum	36
4.2. Diagrama de Gantt inicial	37
4.3. Diagrama de Gantt final	37
4.4. Diagrama de Gantt de las iteraciones	38
4.5. Diagrama de Gantt de las iteraciones finales	38
5.1. Diagrama de casos de uso.	66
5.2. Storyboard de la aplicación.	68
6.1. Diagrama de clases de la actividad front	70
6.2. Diagrama de clases de la actividad principal	72
6.3. Diagrama de clases asociadas a INDI	74
6.4. Diagrama de clases asociadas a los manejadores de propiedades	75
6.5. Diagrama de clases asociadas a los manejadores de dispositivos	76
6.6. Bocetos de interfaces	77
6.7. Capturas de interfaces	78
6.8. Capturas de interfaces	79
6.9. Capturas de la interfaz principal	80
6.10. Capturas de la interfaz principal	81
6.11. Lista de propiedades de un dispositivo	82
6.12. Propiedades text, number y switch	83
6.13. Vista de un Blob	83

6.14. Dialogo para editar una propiedad text	84
6.15. Dialogo para editar una propiedad number	85
6.16. Dialogo para editar una propiedad switch	85
6.17. Dialogo para añadir una conexión	86
6.18. Dialogo para editar una conexión	86
6.19. Dialogo para borrar conexiones	87
6.20. Dialogo de aviso	87
6.21. Vista de los ajustes	88
6.22. Vista de los ajustes	88
6.23. Vista de las propiedades connection y abort	93
6.24. Android Studio	96
8.1. Captura de la Página Web en Firefox (PC, Linux)	108
8.2. Captura de la Página Web en Teléfono Móvil	109
8.3. Captura de Remote Observatory en Google Play	111
8.4. Captura de la página web oficial de la biblioteca INDI, donde se menciona al Observatorio Remoto	113
8.5. Captura del hilo del foro de la biblioteca INDI donde se presenta la aplicación	114
A.1. Init Screen	118
A.2. Main Screen	119
A.3. New Connection	120
A.4. Connection and Devices Panel	120
A.5. Main Menu	121
A.6. Settings	122
A.7. View Log	122
A.8. Edit Connection	123
A.9. Connect Button	124
A.10.Default Device View	124
A.11.Saving Blob Data	126
A.12.Read Only Property Alert	127
A.13.Updating a Text Property	128
A.14.Updating a Number Property	128
A.15.Updating a Switch Property	129
A.16.Global Visibility Button	129
B.1. Vista de las propiedades connection y abort	144
B.2. Vista de edición para la propiedad Connection	144
B.3. Vista de edición para la propiedad Abort	145

Índice de tablas

5.1.	CU-1. Añadir nueva conexión.	42
5.2.	CU-2. Editar una conexión.	43
5.3.	CU-3. Borrar conexiones.	44
5.4.	CU-4. Editar los ajustes.	45
5.5.	CU-4. Conectarse a un servidor.	46
5.6.	Curso alterno de CU-5. Conectarse a un servidor.	46
5.7.	CU-6. Desconectarse de un servidor.	47
5.8.	CU-7. Salir de la aplicación.	47
5.9.	CU-8. Mostrar dispositivo.	48
5.10.	CU-9. Cambiar vista de dispositivo.	49
5.11.	CU-10. Editar una propiedad <i>text</i>	50
5.12.	Curso alterno de CU-10. Editar una propiedad <i>text</i>	50
5.13.	CU-11 Editar una propiedad <i>number</i>	51
5.14.	Curso alterno de CU-11. Editar una propiedad <i>number</i>	51
5.15.	CU-12 Editar una propiedad <i>switch</i>	52
5.16.	Curso alterno de CU-12. Editar una propiedad <i>switch</i>	52
5.17.	CU-13 Editar una propiedad <i>blob</i>	53
5.18.	Curso alterno de CU-13. Editar una propiedad <i>blob</i>	53
5.19.	CU-14 Editar una propiedad <i>connection</i>	54
5.20.	CU-15 Editar una propiedad <i>abort</i>	55
5.21.	CU-16 Guardar un <i>blob</i>	56
5.22.	Curso alterno de CU-16. Guardar un <i>blob</i>	56
5.23.	CU-17 Mostrar un <i>blob</i>	57
5.24.	Curso alterno de CU-17. Mostrar un <i>blob</i>	57
5.25.	CU-18. Ver el log.	58
5.26.	CU-19. Ocultar una propiedad.	59
5.27.	Curso alterno de CU-19. Ocultar una propiedad.	59
5.28.	CU-20. Mostrar una propiedad.	60
5.29.	CU-21. Activar la visibilidad de todas las propiedades ocultas.	61
5.30.	CU-21. Desactivar la visibilidad de todas las propiedades ocultas.	62
5.31.	CU-23. Expandir un grupo.	63
5.32.	CU-24. Contraer un grupo.	64

5.33. CU-25. Iniciar la aplicación.	65
---------------------------------------------	----

Índice de fragmentos de código

6.1. Interfaz UIPropertyManager	89
6.2. Vista XML en android	91
6.3. Lista de objetos manejadores de propiedades	93
6.4. Clase abstracta DeviceView	94
6.5. Lista de objetos manejadores de dispositivos	95
B.1. Interfaz UIPropertyManager	131
B.2. Clase manejadora de la propiedad connection	133
B.3. Clase manejadora de propiedades Abort	136
B.4. Vista item XML de una propiedad	139
B.5. Vista de edición XML de una propiedad	141
B.6. Lista de objetos manejadores de propiedades	143
B.7. Clase abstracta DeviceView	145
B.8. Clase DeviceMeteoView	146
B.9. Lista de objetos manejadores de dispositivos	147

Capítulo 1

Resumen

1.1. Breve resumen y palabras clave

Palabras clave: *móvil, Android, astronomía, hardware astronómico, control, INDI, software libre.*

El objetivo principal de este proyecto es crear una aplicación móvil para controlar y monitorizar diferentes dispositivos astronómicos.

Los avances tecnológicos de la última década han provocado la aparición de un hardware cada vez más sofisticado para usar en las actividades astronómicas. Todo esto unido al desarrollo de los dispositivos móviles y la conectividad a través de internet, han abierto un amplio abanico de posibilidades en un sector que busca la estandarización para permitir controlar cualquier dispositivo remotamente y que sea independiente del hardware concreto.

Actualmente podemos encontrar esfuerzos por implantar estándares para el control de dispositivos astronómicos. Uno de ellos es ASCOM [1], desarrollado bajo la plataforma *Windows* y diseñado para controlar los dispositivos conectándolos a un ordenador. Para los objetivos de este proyecto, ASCOM no es una opción óptima ya que no está pensado para acceder remotamente, recurriendo a soluciones como el “escritorio remoto” de *windows*.

Sin embargo existen soluciones de código abierto y multiplataforma como **INDI** [2].

INDI consiste a su nivel más básico en un protocolo que permite el control, automatización, obtención de datos e intercambio de los mismos entre distintos dispositivos hardware y programas cliente. La idea subyacente en el protocolo INDI es desacoplar aspectos específicos del hardware que se controla de tal manera que cambios en el hardware no impliquen necesariamente

cambios en el software (cosa que ocurre en sistemas más habituales donde el frontend software está fuertemente acoplado con el backend hardware).

Por otro lado, el gran auge de las tecnologías móviles, y en concreto de la plataforma **Android**, enmarcan el proyecto en un contexto ideal para poder conseguir el principal objetivo del proyecto, **crear una aplicación móvil para el control de instrumental astronómico**.

Además de crear una aplicación útil y fácilmente usable, basándonos en la filosofía del software libre y de mejora del software a través de terceros, se ha diseñado parte de la aplicación específicamente para facilitar la incorporación de nuevas vistas para dispositivos. Se han definido las vistas por defecto que son transparentes a los tipos de propiedades y a los tipos de dispositivos. No obstante, se permite que cualquier persona pueda añadir sus vistas personalizadas sin tener que conocer como está diseñada e implementada toda la aplicación.

Por último, al ser un proyecto con muchas posibilidades, se considera que tiene un gran recorrido por lo que es un desarrollo vivo, que seguirá más allá del objetivo del presente documento y cuya finalidad es adaptarse de la forma más fiable posible a las necesidades reales de los astrónomos, ya sean amateurs o profesionales.

Remote Observatory: A INDI client to Android

Jaime Torres Benavente

1.2. Extended abstract and key words

Key words: *mobile, Android, astronomy, astronomical hardware, control, INDI, free software.*

The main goal of this project is to create a mobile app to control and monitor different astronomical devices (hardware).

The advances in technology in the last decades have allowed astronomers all around the world (both professionals and amateurs) to use more sophisticated hardware in their usual astronomy activities. Typical astronomical devices are electronic mounts, that allow to track stars following the exact apparent movement of the sky, CCD cameras, that allow to capture images, electronic focusers and domes, weather stations and so on. All those improvements have additionally been enhanced with the development in network technologies and particular, with the popularization of Internet: not only is now possible to plug all your astronomical devices to your computer, but you can also control them remotely. This fact is increasing the interest of many amateur astronomers that with a relatively low budget can afford to install a remote observatory. It is worth mentioning that since light pollution is a huge problem for astronomy, those remote observatories are often located quite far from the astronomers headquarters (even thousand of kilometers away, in different continents).

To date, there have been efforts to establish standards for the control and monitoring of astronomical devices. One of the most used is ASCOM. However, this standard presents several disadvantages:

It's a Windows only based solution. Its design allows complete freedom for the driver writers to develop a particular user interface for their devices, in a quite deep relation with the operating system. This fact complicates the development of network solutions for the observatory, since those interfaces cannot usually be serialized to the client computer. Encourages closed software.

To overcome those problems, astronomers using ASCOM usually rely on remote desktop software to control their remote observatories.

However, there are other different approaches to control astronomical devices that allow to overcome some of those problems. One of those approaches is INDI, a library that is used to control astronomical devices following a client / server architecture. In this way, the astronomical devices are connected to a computer (INDI server) and the INDI clients that allow the user interaction can be in another different machine. Moreover, as the INDI drivers abstract the different devices as collections of properties (numerical, textual, switches, lights and BLOBs) it is relatively easy to construct generic clients to control any INDI device, even if the device didn't exist at the time of the writing of the client. Additionally, INDI specs are open and the core INDI libraries (and many utilities, including clients and servers) are free software.

It is also important to note that controlling hardware remotely can be quite challenging. Not only you do not have direct control over the hardware (you cannot "unplug" it), but the amount of information that you have from it can be rather limited. Moreover, as weather conditions may vary quite fast (and rain,snow, wind or even direct sunlight can be very dangerous for the observatory equipment), it is necessary to have as many control resources as possible in case of emergencies.

Therefore, having a mobile app to control the equipment may easy many tasks, specially when unexpected events occur (power outages, network failures, and so on). This is the main reason of developing this project, to be able to control and monitor all your astronomical devices anywhere, at any time.

Capítulo 2

Introducción

El presente proyecto constituye un esfuerzo por unir las tecnologías móviles con la astronomía y los instrumentos astronómicos. En los siguientes apartados describiremos las distintas tecnologías o campos de estudio que se verán implicados en el desarrollo posterior:

- La astronomía.
- Instrumental astronómico.
- Control de dispositivos astronómicos.
- INDI.
- Dispositivos móviles.

2.1. La Astronomía

Desde el principio de los tiempos, el ser humano ha mirado al cielo con incertidumbre, viéndolo como una fuente inagotable de interrogantes sin resolver. En casi todas las religiones antiguas existía la “**cosmogonía**” que intentaba explicar el origen del universo, ligando este a los elementos mitológicos, dando paso esta a la “**astronomía**”:

“Ciencia que se ocupa del estudio de los cuerpos celestes del universo, incluidos los planetas y sus satélites, los cometas y meteoritos, las estrellas y la materia interestelar, los sistemas de materia oscura, estrellas, gas y polvo llamados galaxias y los cúmulos de galaxias; por lo que estudia sus movimientos y fenómenos ligados a ellos.”

(<https://es.wikipedia.org/wiki/Astronomia>)

La Astronomía es probablemente la más antigua de las ciencias naturales originándose en la antigüedad en casi todas las culturas humanas. Sus orígenes se pierden en prácticas religiosas de la prehistoria cuyos vestigios se encuentran en numerosos sitios arqueológicos (como Stonehenge) e incorporados todavía en la astrología una disciplina entrelazada con la astronomía y no separada de ella completamente hasta el siglo XVIII en el mundo occidental. La astronomía antigua constituyó las bases del calendario y la medida de períodos temporales como la semana el mes o el año. Los astrónomos antiguos eran capaces de distinguir entre estrellas y planetas dado que las primeras permanecen fijas en sus posiciones relativas mientras que los planetas se mueven una cantidad apreciable de espacio a lo largo de períodos relativamente cortos (Saturno el más lento de los planetas conocidos en la antigüedad describe un periodo orbital en 29 años). La Astronomía antigua culmina con el desarrollo ordenado del modelo heliocéntrico expuesto en las obras de *Ptolomeo*. Previamente *Aristarco de Samos* había medido las distancias de la Tierra a la Luna y al Sol afirmando como consecuencia de éstas que el Sol era el centro del Universo alrededor del cual giraban los demás planetas incluyendo la Tierra. Otros logros destacados de la época clásica de la astronomía fueron los conseguidos por *Hiparco* quien realizó el primer catálogo estelar y propuso un sistema de clasificación estelar en 6 magnitudes basado en la luminosidad aparente de las diferentes estrellas. La Astronomía en la Europa medieval se produce un oscurantismo en todos los campos del conocimiento incluyendo la astronomía. Ésta permanece preservada en escasas copias de tratados antiguos de la astronomía griega y romana. La astronomía observacional tan sólo se conserva en el mundo árabe [3].

Tycho Brahe (1546-1601) introdujo la idea de la precisión de la medida en astronomía e inventó y produjo una gran cantidad de instrumental astronómico previo al telescopio. *Galileo Galilei (1564-1642)* construyó su propio telescopio a partir de un invento holandés y lo utilizó inmediatamente en el estudio astronómico descubriendo los cráteres de la Luna, las lunas de Júpiter y las manchas solares. Sus observaciones tan sólo eran compatibles con el modelo **copernicano**. Paralelamente *Johannes Kepler* expuso sus famosas **leyes de Kepler** para el movimiento de los planetas basando su trabajo en las detalladas observaciones de *Tycho Brahe*.

Una generación más tarde Isaac Newton fue el primer científico que unió la Física con la Astronomía proponiendo que las mismas fuerzas que hacían caer los cuerpos sobre la Tierra causaban el movimiento de los planetas y la Luna. Utilizando su Ley de la gravedad las leyes de Kepler resultan inmediatamente explicadas. Newton también descubrió que la Luz blanca del Sol está descompuesta en diferentes colores, un hecho importantísimo para el futuro desarrollo de la astronomía.

La **astronomía** es una de las pocas ciencias en las que los aficionados aún pueden desempeñar un papel activo, especialmente en el descubrimientos y seguimiento de fenómenos. Es por ello que existe una gran variedad de **herramientas e instrumental astronómico** que permiten a cualquier persona observar el universo.

2.2. Instrumental Astronómico

Existe una gran variedad de **instrumental astronómico** en la actualidad. A continuación se describen las familias más importantes de instrumentos.

2.2.1. Telescopios

“El telescopio es un instrumento óptico que permite ver objetos lejanos con mucho más detalle que a simple vista al captar radiación electromagnética, tal como la luz.”
 (<https://es.wikipedia.org/wiki/Telescopio>)



Figura 2.1: Telescopio (<https://josevicentediaz.files.wordpress.com>).

Hace cuatro siglos nació un invento que habría de redefinir nuestro lugar en el universo. Tachado en su momento como el instrumento más diabólico de la historia, el telescopio sacudió la sociedad hasta las raíces. Al alzar los ojos al cielo nos convencimos de que éramos el centro de la creación, y había razones para ello: desde nuestra perspectiva, todo parece girar en torno a la Tierra [5].

Los fabricantes de vidrio sabían desde la antigüedad que una esfera de vidrio podía aumentar imágenes, pero tuvieron que pasar siglos antes de que alguien ensamblara dos lentes en un tubo y mirara a través de ellas. Señalar la fecha, lugar y autor exactos de su invención es controvertido. Los holandeses se inclinan por el 2 de octubre de 1608, el día en que *Hans Lippershey* patentó un instrumento llamado **kijker**, que significa mirador. Un moledor

de vidrio holandés aseguraba haber inventado un aparato similar, pero el primero en patentarlo fue *Lippershey*. Como era alemán, vivía en Holanda y registró la patente en Bélgica, más de un país ha pugnado por el honor de su autoría. Sin embargo, como dijo *Darwin*:

“en la ciencia el crédito es del que convence al mundo y no del primero en tener la idea”
(Charles Darwin)

Por eso la gloria se la llevó Italia, ya que fueron las mejoras que introdujo *Galileo* las que permitieron usar el aparato como instrumento astronómico. El diseño de *Galileo* consistía en una lente convexa para el objetivo y otra cóncava en el ocular. En 1611 el alemán *Johannes Kepler* fue el primero en usar dos lentes convexas que enfocaban los rayos en un mismo punto. La configuración de *Kepler* aún se usa en binoculares y cámaras fotográficas modernas y es la base del telescopio refractor.

Tras la muerte de *Galileo*, fue *Isaac Newton* quien nos dio una nueva imagen del universo que sobrevivió 250 años hasta la llegada de *Albert Einstein*.

“Si he logrado ver más lejos ha sido porque me he subido a hombres de gigantes”
(Isaac Newton)

Y así, sobre la herencia de *Galileo*, *Newton* inventó el **telescopio reflector**, que es la base de los actuales. La innovación consistía en usar espejos en lugar de lentes para enfocar la luz y formar imágenes. Entonces el universo se nos abrió en todo su esplendor.

2.2.2. Cámaras CCD

Un dispositivo de carga acoplada (en inglés **Charge-Coupled Device**, conocido también como **CCD**), es un circuito integrado que contiene un número determinado de condensadores enlazados o acoplados. Bajo el control de un circuito interno, cada condensador puede transferir su carga eléctrica a uno o a varios de los condensadores que estén a su lado en el circuito impreso [6].

El **CCD** se inventó a finales de los 60 por investigadores de **Bell Laboratories**. Originalmente se concibió como un nuevo tipo de memoria de ordenador pero pronto se observó que tenía muchas más aplicaciones potenciales tales como el proceso de señales y sobretodo la captación de imagen, esto último debido a la sensibilidad a la luz que presenta el silicio.



Figura 2.2: Cámara CCD (<http://www.lunatico.es/>).

El sensor **CCD** de una cámara digital es como el motor de un coche, es la pieza principal. En su forma más elemental, el **CCD** es como un ojo electrónico que recoge la luz y la convierte en una señal eléctrica. Tienen dos diferencias básicas con los foto-multiplicadores:

Los sensores **CCD** son de menor tamaño y están construidos de semiconductores lo que permite la integración de millones de dispositivos sensibles en un solo chip. La eficiencia cuántica de los **CCD** (sensibilidad) es mayor para los rojos. Los foto-multiplicadores son más sensibles a los azules.

Físicamente, un **CCD** es una malla muy empaquetada de electrodos de polisilicio colocados sobre la superficie de un chip. Al impactar los fotones sobre el silicio se generan electrones generados que pueden guardarse temporalmente. Periódicamente se lee el contenido de cada píxel haciendo que los electrones se desplacen físicamente desde la posición donde se originaron (en la superficie del chip), hacia el amplificador de señal con lo que se genera una corriente eléctrica que será proporcional al número de fotones que llegaron al píxel. Para coordinar los períodos de almacenamiento (tiempo de exposición) y vaciado del píxel (lectura del píxel) debe existir una fuente eléctrica externa que marque el ritmo de almacenamiento-lectura: el reloj del sistema. La forma y amplitud de reloj son críticas en la operación de lectura del contenido de los píxeles.

Al tratarse el **CCD** de un dispositivo semiconductor, técnicamente es posible implementar en él todas las funciones electrónicas de un sistema de

captación de imagen, pero esto no es rentable económicamente y por tanto se implementa en otros chips externos al **CCD**: la mayoría de **CCD** de cámaras tienen varios chips (de tres a ocho).

La necesidad de usar chips distintos implica dos desventajas importantes; la necesidad de voltajes múltiples de abastecimiento de los chips y un gran consumo de potencia de todo el sistema electrónico.

2.2.3. Monturas

La montura de un telescopio es la parte mecánica que une el trípode o base al instrumento óptico. Existen varios tipos de monturas, algunas muy simples, otras mas complejas, incluso con correctores electrónicos y dispositivos de localización y seguimiento muy sofisticados (sistemas **GOTO**)



Figura 2.3: Montura (<http://astrofacil.com/>).

La montura tiene como objetivo proveer de movimiento controlado al telescopio. Es muy importante la firmeza y suavidad de los movimientos, para que la observación sea confortable y las astro-fotografías perfectas. Las monturas se clasifican en dos grandes grupos, según los planos de referencia que utilicen (coordenadas).

La más simple es la montura altacimital, que realiza movimientos horizontales y verticales (acimut y altura, respectivamente). Este tipo de diseño

lo traen incorporados los telescopios pequeños, por lo general telescopios refractores de uso terrestre, dado que su uso es simple, y también varios modelos de equipos automatizados (sistemas **GOTO**)

Le sigue la montura ecuatorial, que utiliza como plano fundamental el ecuador celeste (proyección del ecuador terrestre). Este diseño usa las coordenadas ecuatoriales, ascensión recta (A.R. o R.A.) y declinación (Dec.), que son proyecciones de las coordenadas terrestres longitud y latitud, respectivamente, sobre la esfera celeste.

Existen varios tipos de monturas basados en los dos diseños fundamentales anteriores. La montura **Dobson** por ejemplo (suelen llamarse telescopios *dobsonianos* a los que la poseen), es un modelo basado en la altacimutal, sin trípode y un telescopio de diseño newtoniano como instrumento de observación. Es muy utilizado por los que desean una gran apertura en reflectores, por ejemplo los que se construyen su propio espejo y no quieran tener grandes gastos en monturas sofisticadas.

2.2.4. Rueda portafiltros

La rueda porta-filtros consiste en un cuerpo, generalmente de aluminio, que en su interior puede alojar varios filtros, normalmente de 1,25" de diámetro. Lo aconsejable es que tenga, al menos, 4 huecos para filtros si queremos hacer astro-fotografía con cámaras CCD blanco y negro, puesto que vamos a necesitar el azul, rojo y verde (RGB) y, posiblemente, un filtro para infrarrojos.

2.2.5. Enfocadores

El **enfocador** es una pieza fundamental del telescopio. Nos permitirá ver las imágenes formadas tras la reflexión de la luz en el espejo primario y su desviación por el espejo secundario. Para verlas necesitaremos un juego de oculares. La longitud focal de los oculares combinada con la longitud focal de nuestro telescopio nos dará el número de aumentos total del sistema. Dichos oculares están montados en el **enfocador**, un dispositivo móvil que permitirá mover la posición vertical del ocular para enfocar adecuadamente la imagen.

Un ejemplo de enfocador son los de tipo **Crayford** y los **rack and pinion**.



Figura 2.4: Rueda portafiltros (<http://www.lunatico.es/>).



Figura 2.5: Enfocador (<http://nimax-img.de/>).

2.2.6. Cúpulas

Las **cúpulas** son recintos cerrados mas o menos grandes que nos permiten albergar y proteger el instrumental astronómico. De esta forma, las **cúpulas** pueden ser abiertas o cerradas para exponer los instrumentos en el momento de las observaciones.



Figura 2.6: Cúpula (<http://cesar-programme.cab.inta-csic.es/>).

2.2.7. Ópticas activas/adaptativas

La **óptica adaptativa** es una técnica que permite corregir las perturbaciones más importantes que sufren las imágenes astronómicas debido a la atmósfera terrestre. Con este sistema es posible obtener imágenes más nítidas o de mejor resolución espacial. La diferencia que introduce esta técnica es comparable a la que existe entre mirar un objeto situado en el fondo de una piscina con agua o sin agua.

Las posibilidades que la óptica adaptativa ofrece a la astronomía son espectaculares. Eliminar las perturbaciones producidas por la atmósfera equivale esencialmente a observar desde el espacio. Las perturbaciones atmosféricas causan una pérdida en nitidez o resolución espacial. Esta pérdida se traduce, por un lado, en una disminuida capacidad para resolver objetos, es decir, para realizar estudios detallados de su morfología. Por otro lado, influye también en la capacidad de detectar objetos débiles, dado que la imagen se dispersa en puntos de luz mayores [7].

La mejora que introduce la óptica adaptativa se puede cuantificar utilizando la relación entre el tamaño del telescopio y el tamaño de la mejor imagen que puede obtener. El poder de detección de un telescopio aumenta con el diámetro de su espejo primario y disminuye con el tamaño de la imagen que forma de un objeto puntual (de aquí la importancia de la calidad de imagen en un telescopio). Por tanto, la diferencia con un mismo espejo de 10 metros, entre conseguir enfocar imágenes de 0.4 segundos de arco (lo posible en una noche de visibilidad excelente) y una imagen de 0.04 segundos de arco, que debe ser posible con un sistema de óptica adaptativa, equivaldría a tener un espejo primario de 100 metros.



Figura 2.7: Óptica adaptativa (<https://www.valkanik.com>).

2.2.8. Estaciones meteorológicas

Las **estaciones meteorológicas** son sistemas compuestos por un “*data logger*” y un conjunto de sensores que nos proporcionan datos de las distintas magnitudes meteorológicas, tales como la temperatura, humedad, presión barométrica, etc... permitiéndonos generar modelos a partir de los cuales conocer la situación climática y su posible evolución.

Gracias a los datos aportados por las **estaciones meteorológicas**, podemos conocer la climatología en el momento de realizar observaciones astronómicas. De esta forma podemos decidir si las condiciones son óptimas, o incluso decidir si debemos cerrar la cúpula para evitar daños en los instrumentos por lluvias o similar.



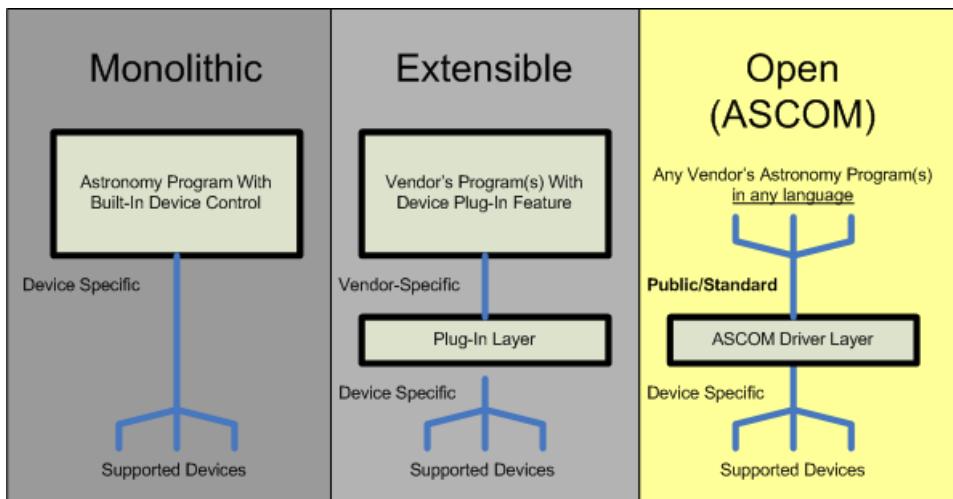
Figura 2.8: Estación meteorológica (<http://www.depositohidrografico.com/>).

2.3. Control de dispositivos astronómicos

Actualmente existen diversas formas de controlar los dispositivos astronómicos pero la mayoría presenta los mismos inconvenientes:

- Normalmente se controlan los dispositivos directamente.
- Se conecta el dispositivo a un PC y se trabaja desde él.
- Se utilizan herramientas para el control remoto como el “escritorio remoto”.

Por otro lado, existen estándares como el de **ASCOM** para instrumental astronómico. Con él, se intenta crear una capa entre los programas para controlar dispositivos astronómicos y los propios dispositivos. **ASCOM** solo puede utilizarse en sistemas *Microsoft Windows*. Su diseño tiene una relación bastante profunda con el sistema operativo lo cual dificulta el desarrollo basado en red.



- | | | |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> ✓ Each program must have its own code for device control ✓ Vendors must write all device control code ✓ New devices require new releases of programs ✓ Bugs in device control require new releases of programs | <ul style="list-style-type: none"> ✓ Several of vendor's programs can share device control code ✓ Anyone can write device control plug-ins ✓ New devices require only new plug-ins ✓ Bugs in device control require only new plug-ins | <ul style="list-style-type: none"> ✓ <u>Vendor-independent</u>: any program can use a driver ✓ Anyone can write and support drivers ✓ New devices require only drivers ✓ Bugs in device control require only new drivers |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Figura 2.9: ASCOM Standard (<http://ascom-standards.org/>)

2.4. INDI

“The Instrument Neutral Distributed Interface (INDI) Library is a cross-platform software designed for automation control of astronomical instruments. It supports a wide variety of telescopes, CCDs, focusers , filter wheels..etc, and it has the capability to support virtually any device. INDI is small, flexible, easy to parse, and scalable. It supports common DCS functions such as remote control, data acquisition, monitoring, and a lot more. With INDI, you have a total transparent control over your instruments so you can get more science with less time.”

(<http://indilib.org/about.html>)

El protocolo **INDI** es una plataforma software diseñada para el control de instrumental astronómico, aunque podría usarse con cualquier dispositivo, incluidos “virtuales”. La biblioteca **INDI** permite controlar cualquier dispositivo con un driver **INDI** mediante el uso de archivos XML. Sus principales ventajas frente a otras soluciones para el control de dispositivos son:

- Es una biblioteca ligera, flexible y escalable.
- Es de código abierto por lo que cualquiera puede ver su código y mejorararlo o crear drivers para cualquier dispositivo.
- El intercambio de información entre clientes, servidores y drivers es mínimo.
- Es multiplataforma.
- Separa clara y totalmente el cliente del servidor.
- Los fabricantes comienzan a desarrollar drivers para sus dispositivos o liberan las especificaciones para que la comunidad pueda desarrollarlos.
- Existen numerosos clientes INDI como kstars, Cartes Du Ciel, Xephem y Stellarium (en fase de desarrollo).

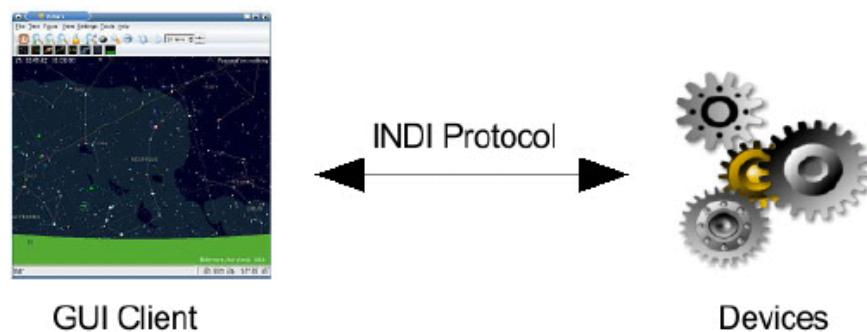


Figura 2.10: INDI Client (<http://www.indilib.org/>)

2.4.1. Breve introducción a INDI

INDI consiste a su nivel más básico en un protocolo que permite el control, automatización, obtención de datos e intercambio de los mismos entre distintos dispositivos hardware y programas cliente. La idea subyacente en el protocolo INDI es desacoplar aspectos específicos del hardware que se controla de tal manera que cambios en el hardware no impliquen necesariamente cambios en el software (cosa que ocurre en sistemas más habituales donde el frontend software está fuertemente acoplado con el backend hardware).

Para conseguir un desacople efectivo entre los clientes y el hardware se define un protocolo basado en XML que permite abstraer los dispositivos

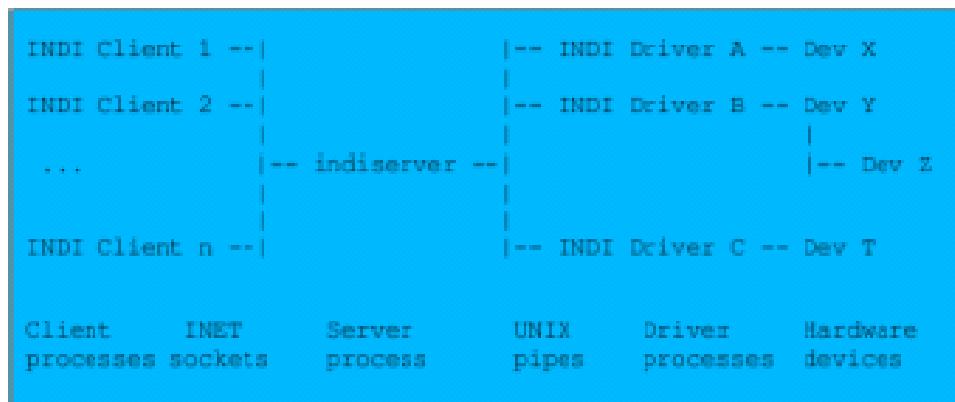


Figura 2.11: INDI Server (<http://www.indilib.org/>)

hardware como conjuntos de propiedades que pueden ser leídas, y modificadas por los clientes (siempre estableciendo las restricciones oportunas).

2.4.2. Drivers, Servidores y Clientes INDI

Pese a que nivel más básico INDI es “simplemente” una especificación de un protocolo basado en XML, a un nivel superior se distinguen tres entidades diferentes que interaccionan entre sí para tener un sistema de control plenamente funcional:

- **Drivers:** Son programas que se ejecutarán en la máquina en la que están conectados los dispositivos hardware. Son los encargados de la comunicación directa con los dispositivos y su abstracción a propiedades INDI.
- **Servidor:** Es un programa cuya función principal es ejecutar los drivers y permitir la conexión a los mismos por parte de los clientes (funciona de un modo similar a un proxy). Normalmente reside en la máquina donde están conectados los dispositivos, aunque en principio se pueden crear estructuras de red tipo árbol de servidores. El intercambio de información entre el servidor y los drivers se realiza utilizando el protocolo INDI.
- **Cliente:** Es un programa que permite conectar con uno o más servidores y su función principal es hacer de interfaz con el usuario. Para ello conecta (usualmente a través de la red) con el servidor e intercambia información sobre los dispositivos utilizando el protocolo INDI (FIGURA 2.11). Es interesante recalcar que los clientes pueden ser de cualquier estilo: desde programas con interfaz de usuario avanzadas,

hasta programas simples en línea de comandos scripts completamente automáticos que controlen o monitoricen los dispositivos.

2.4.3. Abstracción de los Dispositivos

Para conseguir abstraer los dispositivos y que puedan ser controlados o monitorizados por los clientes el protocolo INDI define las llamadas *propiedades*. Las propiedades tienen ciertas características como por ejemplo:

- **Permiso:** Las propiedades tienen uno de 3 posibles permisos:
 - **Lectura y escritura (R/W):** La propiedad puede ser leída y modificada.
 - **Solo lectura (RO)**
 - **Solo escritura (WO)**
- **Estado:** Las propiedades tienen uno de los siguientes 4 posibles estados:
 - **Ok:** Estado correcto
 - **Idle:** Estado indefinido (normalmente la propiedad aún no ha sido usada)
 - **Busy:** Está ocupada (probablemente cambiando de valor)
 - **Alert:** Ha ocurrido algún problema con la propiedad.

Al margen de esas características, hay que mencionar que las propiedades son un conjunto de usualmente uno o más elementos distintos. Es decir, una única propiedad puede agregar varios valores distintos (normalmente relacionados).

Existen 5 tipos de propiedades distintas:

- **Textuales:** Permiten manejar información textual (cadenas de caracteres).
- **Numéricas:** Permiten manejar información numérica. Estas propiedades permiten especificar los rangos de valores válidos así como el formato de visualización del número (entero, flotante, flotante exponencial o sexagesimal).
- **Luces:** Permiten manejar “señales” o luces que tienen uno de los siguientes cuatro posibles valores:
 - **Ok**
 - **Idle**

- **Busy**
- **Alert**
- **Switch:** Permiten manejar valores entre una lista de posibles alternativas. Permiten especificar la regla de selección de cada una de las alternativas:
 - **Una de muchas:** De todas las alternativas, una y solo una debe estar elegida.
 - **Como mucho una:** De todas las alternativas se puede elegir una o ninguna.
 - **Cualesquiera de muchas:** De todas las alternativas se pueden seleccionar cualquier numero de ella (desde ninguna a todas).
- **BLOB:** Permite manejar valores binarios (como por ejemplo datos de imagen en una cámara).

2.4.4. Ejemplo de Abstracción de un Dispositivo

Para comprender mejor el mecanismo de abstracción que aplica INDI, vamos a realizar un ejemplo sencillo de como el programador de un driver abstrae un dispositivo. Supongamos que se trata de una cámara sencilla. La especificación de la cámara dice que su funcionamiento es muy simple: Solo hace falta comunicarle por el puerto serie un comando que incluye el tiempo de exposición de la misma. Una vez mandado ese comando la cámara tomará la fotografía y devolverá los datos de la fotografía en bruto, como un array de valores numéricos (a mayor valor, mayor intensidad luminosa del pixel).

Por tanto, para ofrecer la funcionalidad al cliente INDI, el driver define las siguientes propiedades:

- **Nombre del driver:** Tipo texto, R0. Contendrá el nombre del driver y dispositivo. No cambiará nunca.
- **Tiempo de Exposición:** Tipo numérico, R/W, entre 0 y 3600 (segundos). Por defecto 0.
- **Imagen:** Tipo BLOB, R0. Contendrá la información binaria de la imagen. En este ejemplo, será una imagen tipo PNG.

El driver, además de definir esas propiedades tendrá el siguiente comportamiento general:

- Cuando conecte un cliente mandará la información de las tres propiedades.

- Quedará a la espera de que cambie el **tiempo de exposición**.
- Cuando el cliente mande un **tiempo de exposición** nuevo, mandará el comando apropiado a la cámara para que haga la captura de la **imagen** y esperará a recibir los datos binarios en bruto de la misma.
- Una vez recibidos dichos datos binarios los transformará a formato **PNG**.
- Mandará al cliente nuevos valores para las propiedades de **tiempo de exposición** (0, para indicar que ya ha acabado la exposición) e **imagen** (con los datos binarios, el **PNG**).

2.4.5. INDI for Java

La biblioteca **INDI** está escrita en lenguaje **C**, pero existe una implementación completa realizada en **Java** y que se encuentra en constante mejora. En la página oficial de INDI podemos encontrar toda la información sobre nuevas versiones y la documentación para poder utilizarla. La principal ventaja de poder usar **Java** es que podemos implementar drivers y clientes con la potencia de un lenguaje Orientado a Objetos y combinarlo con otras tecnologías como los dispositivos móviles basados en la plataforma **Android**

2.5. Dispositivos Móviles

Un **dispositivo móvil** es un tipo de computadora de tamaño pequeño, con capacidad de procesamiento, con conexión a internet, con memoria, que pueden llevar a cabo otras funciones más generales.

Los **dispositivos móviles** hoy en día están integrados en la mayoría de tareas cotidianas de una persona. La tendencia de la sociedad actual nos empuja hacia un mundo cada vez más móvil donde necesitamos estar conectados e interactuar con otros sistemas. Es por ello que la mayoría de soluciones tecnológicas, hayan sido pensadas o no para el sector de los dispositivos móviles, siempre acaba teniendo una versión para éstos.

Paralelamente a la expansión de los **dispositivos móviles**, se han creado un gran número de sistemas operativos para estos dispositivos entre los que se encuentra:

- Android.
- iOS.
- BlackBerry OS.
- Palm OS.
- Windows Mobile/Phone.
- Symbian.
- ...

Actualmente **Android** y **iOS** copan el 96.3 % del mercado [8] . Por lo que nos centraremos principalmente en estos sistemas operativos (S.O.) y los **dispositivos móviles** compatibles con ellos.

2.5.1. iOS

iOS es un S.O. móvil de la compañía *Apple Inc* originalmente desarrollado para el *iPhone*¹ y posteriormente introducido en otros **dispositivos móviles** de la compañía como el *iPod touch*² y el *iPad*³. **iOS** no puede ser instalado en hardware de terceros.

Actualmente tiene una cuota de mercado aproximadamente del 19.7 %, siendo el segundo S.O. más utilizado.

¹Smartphone de la compañía Apple Inc.

²Dispositivo móvil para reproducir multimedia de la compañía Apple Inc

³Tablet de la compañía Apple Inc.

iOS es un sistema muy estable, diseñado para un hardware muy concreto y por tanto, muy eficiente y depurado. Pero de cara a elegirlo como una opción a la hora de desarrollar una nueva aplicación para **dispositivos móviles** se debe tener en cuenta los siguientes aspectos:

- Hay que pagar una cuota anual de 99\$ para poder publicar aplicaciones en el *Apple Store*⁴. Además si esta licencia, no podremos desarrollar aplicación y cargarla en nuestros dispositivos Apple.
- Necesitamos un MAC⁵ ya que las herramientas para el desarrollador solo pueden utilizarse en sus equipos.
- Necesitaremos conocer el lenguaje de programación **Objective-C**
- **iOS** es un sistema de código cerrado que va en contra de la filosofía del **Software Libre** y el código abierto y reutilizable.

Aunque **iOS** es un sistema muy extendido y con un gran número de usuarios, creemos que no es la mejor opción para orientar una aplicación móvil basada en **Software Libre** además de la inversión anual requerida para poder publicar una aplicación que pretendemos sea gratuita, libre, accesible a cualquier usuario y sin publicidad.

2.5.2. Android

Android es un Sistema Operativo basado en un **núcleo Linux**⁶. Fue diseñado principalmente para dispositivos móviles con pantalla táctil, relojes inteligentes, televisiones inteligentes y automóviles. Inicialmente pertenecía a la compañía **Android Inc.** que posteriormente sería adquirida por **Google**. Actualmente posee la mayor cuota de mercado de aproximadamente el 76.6 %.

Los principales componentes del sistema operativo **Android** son:

- **Aplicaciones:** Todas las aplicaciones están escritas en lenguaje de programación **Java**.
- **Framework**⁷: Los desarrolladores tiene acceso completo a las mismas API's⁸ que utiliza el sistema. La arquitectura está diseñada para simplificar la reutilización de componentes.

⁴Tienda de aplicaciones de la compañía Apple Inc.

⁵Computadoras personales de la compañía Apple Inc.

⁶Sistema operativo basado en Unix

⁷Marco de trabajo para los desarrolladores

⁸Interfaz de programación de aplicaciones (Application Programming Interface)

- **Runtime de Android:** Android incluye un 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*⁹
- **Núcleo Linux:** Android depende de Linux para los servicios base del sistema como la seguridad, la gestión de memoria, la gestión de procesos, etc. El núcleo Linux también sirve como capa de abstracción entre el hardware y el software.

Android no tiene restricciones de uso por lo que puede utilizarse en número muy extenso de dispositivos móviles. Además es un sistema parcialmente de código abierto. Está basado en Linux y la mayoría del código es abierto aunque no todo el sistema lo es.

De cara al desarrollo de aplicaciones móviles, **Android** es una opción muy recomendable por las siguientes razones:

- La arquitectura del sistema (basada en Linux, lenguaje de programación Java,...)
- La mayoría de los dispositivos móviles del mundo tienen como sistema operativo a **Android** por lo que la difusión será mayor que con otros sistemas.
- Las herramientas para desarrollar en Android son multiplataforma y gratuitas. Para poder crear y probar una aplicación solo necesitas un ordenador con cualquier sistema operativo, un dispositivo móvil con Android y descargar las herramientas para desarrolladores.
- Para poder publicar aplicaciones en *Google Play*¹⁰ hay que únicamente pagar 25\$ sin tener renovarlo anualmente y sin ninguna limitación.

⁹Máquina virtual que utiliza la plataforma Android para ejecutar aplicaciones Java.

¹⁰Tienda de aplicaciones para dispositivos Android

Capítulo 3

Objetivos

El objetivo global de este proyecto es desarrollar una aplicación para la plataforma **Android** que implemente un cliente utilizando la biblioteca “**INDI for Java**” basado en el **Software Libre** y que sea fácilmente extensible.

Este objetivo se desglosa en los siguientes objetivos principales:

- **OBJ-1.** Conseguir un cliente funcional capaz de controlar cualquier dispositivo **INDI**.
- **OBJ-2.** Poder gestionar múltiples conexiones con múltiples dispositivos simultáneamente.
- **OBJ-3.** Fácilmente extensible, permitiendo añadir vistas para propiedades y dispositivos por parte de desarrolladores ajenos al proyecto.
- **OBJ-4.** Desarrollar la aplicación bajo una licencia de código abierto fomentando la filosofía del **Software Libre** y la publicación de todo el código y recursos.

Además de los objetivos principales, se persigue alcanzar los siguientes objetivos secundarios:

- **OBJ-S-1.** Desarrollar la aplicación siguiendo los estándares actuales y las recomendaciones para la plataforma **Android**.
- **OBJ-S-2.** Facilitar la usabilidad mediante un diseño adecuado de las interfaces, adaptándola a los distintos tamaños de pantalla y personalizándolas a las propiedades estándares de **INDI**.
- **OBJ-S-3.** Desarrollar para incluir un correcto funcionamiento en el mayor número posible de versiones de **Android**, maximizando el número de dispositivos compatibles.

- **OBJ-S-4.** Añadir una versión estable en *Google Play* y publicar el *APK*¹ para poder descargarlo a través de internet.
- **OBJ-S-5.** Internacionalizar la aplicación (como mínimo en inglés y castellano).
- **OBJ-S-6.** Difusión del software a través de varios canales: *Google Play*, página oficial de **INDI** y desarrollo de una web propia.

Para la realización de los objetivos se pondrán en práctica los conocimientos alcanzados en:

- **Ingeniería del software** para el análisis y diseño del proyecto.
- **Programación orientada a objetos** para la estructura y la organización del código **Java**.
- **Programación concurrente y sistemas operativos** para la gestión de las distintas hebras y la comunicación entre ellas.
- **Programación de sistemas multimedia** para poder implementar las interfaces de usuario en **Android** y poder tratar y mostrar imágenes enviadas por los dispositivos.
- **Infraestructura virtual** para poder gestionar los sistemas para realización de test y simulaciones.
- **Transmisión de datos y redes de computadores** para comprender el comportamiento del protocolo **INDI** y configurar correctamente las redes para las pruebas.
- **Diseño de Aplicaciones para Internet** para añadir código **HTML** a las interfaces de **Android** y para el desarrollo de un portal web que difunda información sobre la aplicación.

Por otro lado, han sido necesarios alcanzar conocimientos en otras áreas:

- **Astronomía y equipos astronómicos** para entender a los usuarios potenciales y poder acomodar la aplicación a sus necesidades.
- **Android** para conocer las herramientas que ofrece la plataforma y usar las más adecuadas según las necesidades concretas.
- **Raspberry Pi**² para montar un servidor permanente de pruebas o acceso público para probar la aplicación

¹Paquete para el sistema operativo Android (Application Package File)

²Ordenador de placa reducida y única de bajo coste.

- **Latex³** para la realización del presente documento y la ampliación de conocimientos para futuros textos científicos.
- **Git** para la gestión de versiones y la publicación de código abierto que permita a otros desarrolladores participar.

3.1. Alcance de los objetivos

La aplicación móvil desarrollada debe cumplir los objetivos principales para cubrir una necesidad existente. Actualmente no existe ninguna aplicación móvil basada en **INDI** para controlar dispositivos astronómicos que cumplan los objetivos propuestos. Con la realización del proyecto se pretende cubrir dicha necesidad, obteniendo una aplicación estable y que será mantenida y mejorada más allá de la finalización del Proyecto Fin de Grado. Se trata, por tanto, de un proyecto vivo y extensible en el tiempo.

La consecución de alcanzar también los objetivos secundarios tendrá un efecto directo en la difusión de la aplicación y en la satisfacción directa de los usuarios de la misma. Por ello, se comprará una licencia de desarrollador para *Google Play* y se publicará y dará difusión en distintos canales de comunicación como la página oficial **INDI** y a través de foros y páginas web.

3.2. Interdependencia de los objetivos

El principal objetivo que debe cumplir la aplicación es el *OBJ-1*, aunque todos los objetivos son independientes excepto los objetivos secundarios *OBJ-S-1*, *OBJ-S-2* y *OBJ-S-3*. Seguir los estándares y recomendaciones de la plataforma **Android** derivará en una mayor compatibilidad con versiones antiguas del sistema operativo y un diseño de la interfaz de usuario más amigable y fácil de usar.

³Sistema de composición de textos.

Capítulo 4

Planificación

4.1. Fases

El proyecto se divide en una sucesión de fases previamente establecidas que nos ayudará a estructurar, temporizar y evaluar los costes tanto económicos como humanos. Dado que el propio planteamiento del proyecto implica el uso de una serie de tecnologías como Android e **INDI**, y la necesidad de conocer el campo de la **astronomía**, se propuso el proyecto con bastante antelación ya que se preveía tener que realizar una fase de familiarización con las tecnologías y campos implicados. Esta fase es bastante extensa ya que se parte de cero.

- **Fase 0:** Planteamiento del problema.
- **Fase 1:** Familiarización con las tecnologías implicadas.
- **Fase 2:** Especificaciones del proyecto.
- **Fase 3:** Análisis y diseño.
- **Fase 4:** Implementación.
- **Fase 5:** Pruebas.
- **Fase 6:** Documentación.

4.2. Estimación de tiempos

A continuación se muestran las fases con sus actividades principales y la estimación inicial de tiempos.

▪ **Planteamiento del problema:**

- Primera reunión con el cliente.
- Descripción de los objetivos que se persiguen.
- Planteamiento de las posibles tecnologías.
- Estimación: 4 horas.

▪ **Familiarización con las tecnologías implicadas:**

- Android: Generación de aplicaciones.
- Android: Generación de interfaces de usuario.
- Android: Posibles entornos para el desarrollador
- INDI: Comprensión del protocolo.
- INDI: Familiarización con la biblioteca “*INDI for Java*”
- Familiarización con el campo de la astronomía.
- Realización de pruebas simples para estudiar la viabilidad técnica del proyecto
- Estimación: 80 horas.

▪ **Especificación del proyecto:**

- Tecnologías elegidas. Entornos de trabajo.
- Recursos humanos.
- Presupuesto.
- Temporización.
- Estimación: 18 horas.

▪ **Análisis y diseño:**

- Análisis de requisitos.
- Diagramas.
- Metodología de desarrollo.
- Estimación: 36 horas.

▪ **Implementación:**

- Herramientas seleccionadas.

- Creación de una aplicación en Android para abrir y cerrar conexiones de red.
- Creación de una aplicación en Android para conectarse con un driver INDI.
- Creación de una aplicación en Android para poder listar todas las propiedades y dispositivos de una conexión INDI.
- Creación de una aplicación en Android para poder interactuar con las propiedades de los dispositivos de una conexión INDI.
- Creación de una aplicación en Android para poder gestionar varias conexiones INDI simultáneamente.
- Creación de interfaces de usuario específicas para propiedades concretas y dispositivos concretos.
- *Estimación:* 180 horas.

- **Pruebas:**

- Pruebas de la aplicación en entornos simulados.
- Pruebas de la aplicación en entornos reales.
- *Estimación:* 30 horas.

- **Documentación:**

- Documentación de la aplicación.
- Manual de usuario.
- Documentación del proyecto.
- Manual del desarrollador.
- *Estimación:* 30 horas.

4.3. Recursos humanos

Dado que el objetivo es demostrar las capacidades y competencias del alumno a la hora de afrontar un proyecto, el equipo de recursos humanos solo lo formará él, teniendo que afrontar todas las etapas del desarrollo del proyecto.

4.4. Presupuesto

Para el presente proyecto se tendrán en cuenta los siguientes costes:

- **Costes por hora de equipo humano:** En este caso son las horas dedicadas al proyecto por parte del alumno. Podemos ver que el total de horas estimadas son 298 horas. Si cuantificamos el precio de desarrollo por hora. Si estimamos el precio por hora en 25€ tenemos un coste en equipo humano de 7450€.
- **Costes asociados a licencias necesarias para publicar o desarrollar el software:** Dado que hemos elegido la plataforma **Android** y que basamos el proyecto en **Software Libre** no será necesario realizar ninguna inversión previa. Únicamente debemos tener en cuenta que para poder publicar la aplicación en *Google Play* debemos pagar, aproximadamente, 25€.
- **Costes asociados a los entornos de prueba simulados:** Para poder realizar las pruebas ha sido necesario comprar una *Raspberry Pi B+* que tiene un coste asociado de 35€. En ella se alojan los simuladores necesarios para testear las diferentes funciones del software.
- **Costes asociados a los entornos de prueba con equipos reales:** Los entornos de prueba simulados son limitados, por lo que para poder probar de forma completa el software estaría planificado adquirir instrumental astronómico. Por ello se recomienda una montura, un telescopio básico y una cámara básica por 400€, aunque para el proyecto este equipo ha sido prestado.
- **Costes asociados a la publicación y difusión a través de internet:** Para dar difusión y permitir descargar la aplicación sin tener que usar *Google Play* se ha desarrollado una página web cuyo coste anual de dominio y hosting asciende a 30€ al año.

Como puede observarse, el coste inicial del proyecto es de **7940€**

4.5. SCRUM

Hasta ahora hemos basado la planificación en una metodología de desarrollo clásica o *en cascada*. Esta metodología se basa en un conocimiento alto de los requisitos del sistema por parte del cliente y una estructura fija y previamente establecida.

Para el proyecto actual no podemos utilizar este tipo de metodología ya que el cliente solo sabe a grandes rasgos lo que quiere, dado que hay una

parte de investigación asociada a la consecución del proyecto, lo cual implica la revisión de los requerimientos a lo largo del proceso de desarrollo. Es por ello que se considera más idóneo el uso de una **metodología ágil** basada en iteraciones incrementales como **Scrum**.

En **Scrum** se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, **Scrum** está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales [9].

En **Scrum** un proyecto se ejecuta en bloques temporales cortos y fijos (iteraciones de un mes natural y hasta de dos semanas, si así se necesita). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

Debida a las características descritas, se puede aplicar al proyecto dado que en cada iteración el cliente tendrá una aplicación funcional que podrá probar y comprobar si cumple con los objetivos y que nuevos requerimientos son necesarios para, de esta forma, retroalimentar el proceso de desarrollo produciendo una nueva iteración.

En la figura 5.1 podemos ver el proceso de desarrollo por iteraciones incrementales en **Scrum**.

Esta metodología redefine las fases en tanto en cuanto ahora debemos repetir varias veces las fases de análisis, diseño, implementación, pruebas y documentación. Las horas estimadas son las mismas ya que se repartirían entre las iteraciones. De esta forma el coste del proyecto no se ve incrementado, solo la organización de las tareas y fases de cara al desarrollo del software.

4.6. Temporización

En la figura 4.2 se muestra un diagrama de **Gantt**¹ para ilustrar la temporización de las tareas basándonos en la planificación inicial. Como puede observarse, tenemos una fase inicial de planteamiento del problema. Tras ella, se realiza un parón pactados para poder completar el resto de docencia del curso. Después encontramos la fase de familiarización a la que se dedica 123 días para documentarse sobre los objetivos del proyecto y los campos que van a intervenir. A partir de aquí podemos encontrar la fase **Scrum** con 6 iteraciones mensuales con una reunión al inicio de cada una

¹Herramienta gráfica para mostrar la temporización de una serie de tareas

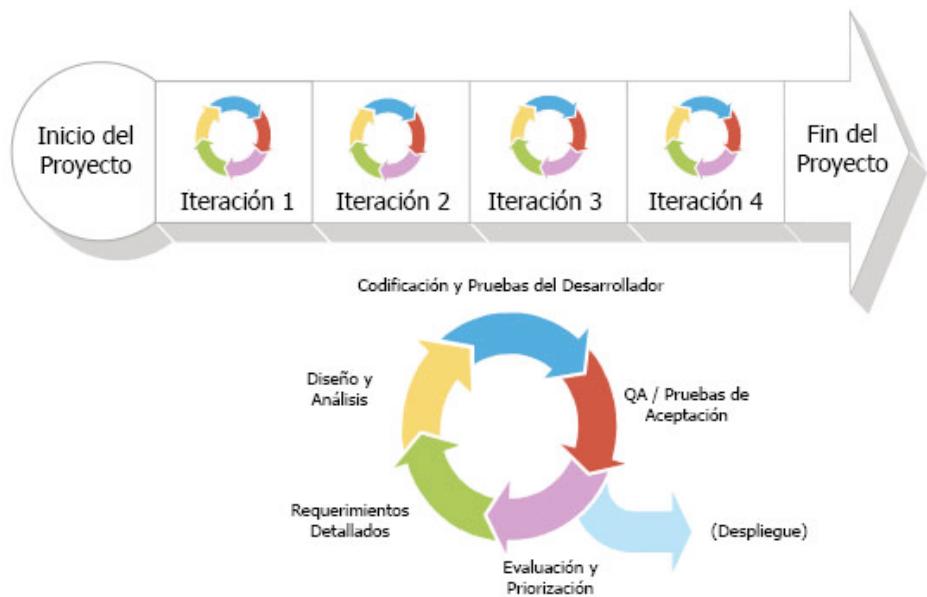


Figura 4.1: Proceso de desarrollo Scrum (<http://www.qasoluciones.es/metodologia/agile>)

para el análisis de requerimientos y una final para la entrega de la versión implementada con el fin de comprobar si se han cumplido. Cuando se considera que el proyecto cumple todos los objetivos se finalizan las iteraciones y se pasa a la fase de cierre del proyecto, en paralelo con la difusión del software a través de una serie de canales de comunicación que se tratarán más a delante.

Aunque inicialmente se plantea una temporización basada en un número fijo de iteraciones con un periodo de un mes cada una, el proceso de desarrollo hace necesario modificar los planteamientos iniciales de horas, costes y temporización. Por ello, en la figura 4.3 puede observarse una diagrama de **Gantt** con la temporización real en la que podemos ver como la fase de familiarización fue más extensa de lo previsto, debido en parte a la carga de asignaturas que impidieron dedicar más tiempo al objetivo, así como la dificultad de iniciarse en tecnologías como **Android** o **INDI**. El retraso en esta fase provocó el desplazamiento de las iteraciones. Es por ello que las dos últimas iteraciones se marcaron con una duración de 15 días para acelerar el proceso y poder entregar el proyecto en la convocatoria de septiembre. El hecho de tener que modificar una temporización planteada inicialmente es una excelente lección sobre los cálculos de tiempo y la necesidad de tener en cuenta imprevistos no contemplados inicialmente.

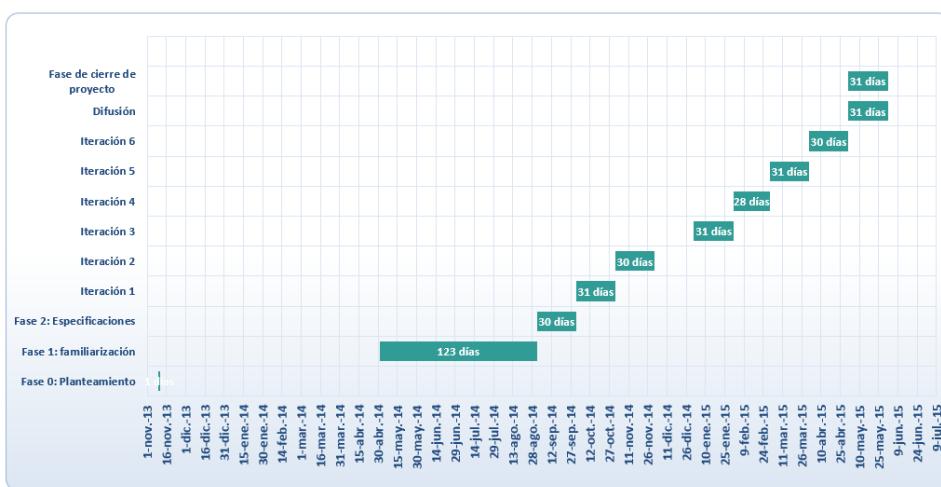


Figura 4.2: Diagrama de Gantt inicial

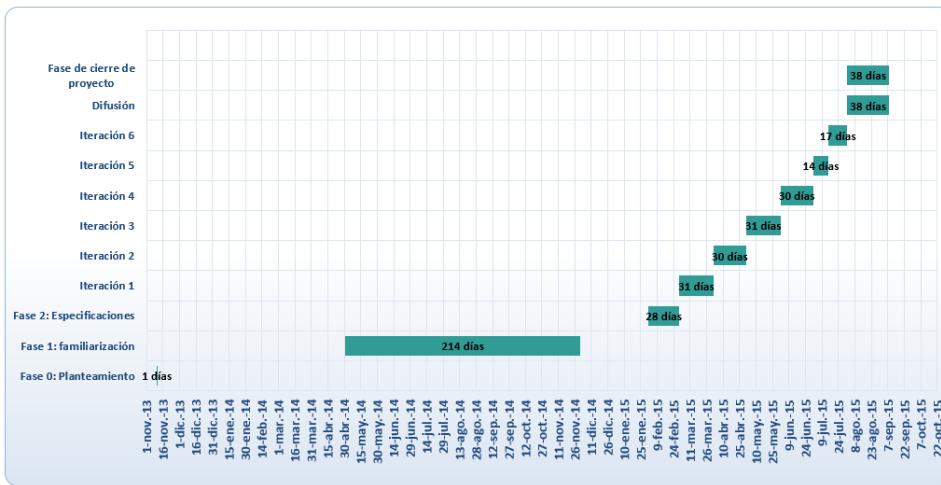


Figura 4.3: Diagrama de Gantt final

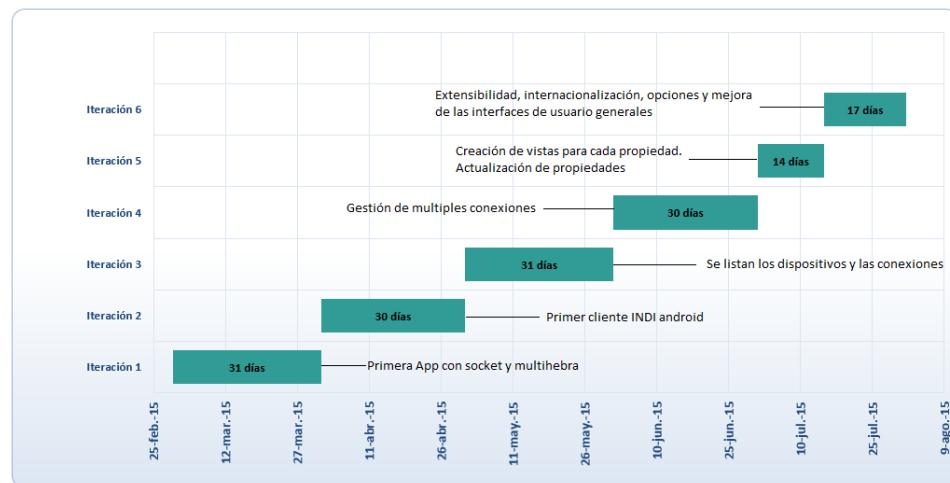


Figura 4.4: Diagrama de Gantt de las iteraciones

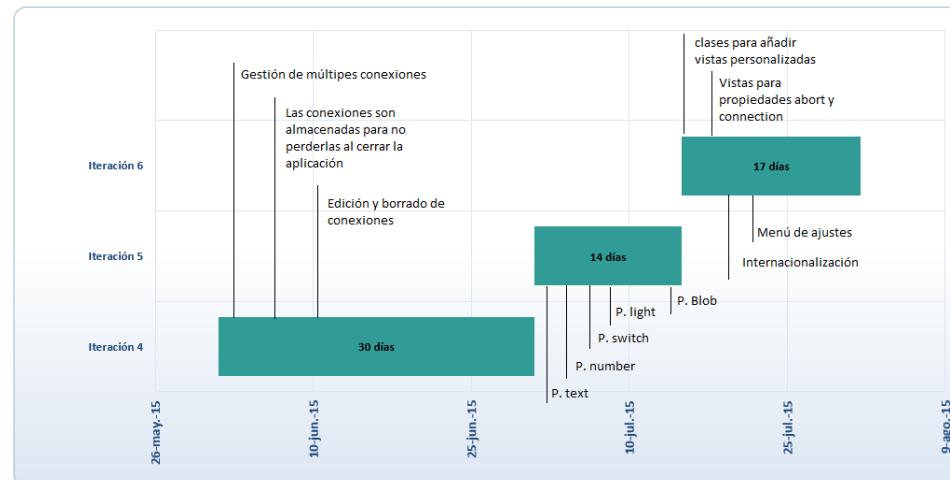


Figura 4.5: Diagrama de Gantt de las iteraciones finales

Capítulo 5

Análisis

En este capítulo vamos a desgranar el proceso de análisis del proyecto, describiendo cada una de las partes que se han realizado.

5.1. Análisis de requisitos

El primer paso en el análisis de un desarrollo software es identificar los requisitos funcionales y no funcionales. Estos requisitos son los que deberá garantizar el producto final y son generados a partir de las entrevistas con el cliente y los objetivos marcados para el software.

Nuestra metodología es ágil basada en iteraciones incrementales por lo que los requisitos son analizados en cada iteración, pudiendo ser modificados según las necesidades.

5.1.1. Requisitos funcionales

Los requisitos funcionales son las características que debe satisfacer el sistema, es decir, todas aquellas funciones que debe cumplir el producto final:

- **RF-1.** Conectarse con un servidor INDI.
- **RF-2.** Gestionar conexiones (crear, editar y borrar).
- **RF-3.** Listar todos los dispositivos de una conexión INDI.
- **RF-4.** Listar todas las propiedades de un dispositivo INDI.
- **RF-5.** Tener más de una conexión INDI simultáneamente.
- **RF-6.** Mostrar un log para cada conexión.
- **RF-7.** Agrupar las propiedades por grupos.

- **RF-8.** Editar las propiedades INDI:

- **RF-8.1.** Propiedad Blob.
- **RF-8.2.** Propiedad Switch.
- **RF-8.3.** Propiedad Number.
- **RF-8.4.** Propiedad Text.
- **RF-8.5.** Propiedad Light.

5.1.2. Requisitos no funcionales

Los requerimientos no funcionales, como su nombre sugiere, son aquellos requerimientos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste:

- **RN-1.** Las interfaces deben seguir las recomendaciones de diseño establecidas por Android.
- **RN-2.** Se deben usar las clases y elementos de interfaz recomendados para la última versión de Android y usar las bibliotecas de compatibilidad.
- **RN-3.** Controlar la hebra principal para no sobre cargarla, creando nuevas hebras en paralelo mejorando así el rendimiento.
- **RN-4.** Crear interfaces específicas para las propiedades genéricas de INDI y para dispositivos conocidos.
- **RN-5.** Utilizar licencias libres para publicar el proyecto como Software libre
- **RN-6.** Adaptar la aplicación a distintos tamaños de pantalla.
- **RN-7.** Diseñar el software para facilitar la extensibilidad de las vistas de dispositivos y propiedades.
- **RN-8.** Internacionalización de la aplicación: Mínimo inglés y castellano.

5.2. Casos de uso

Un caso de uso es una descripción de los pasos o las actividades que deberán realizarse para llevar a cabo algún proceso. Los personajes o entidades que participarán en un caso de uso se denominan actores

5.2.1. Descripción de actores

- **Ac-1.** Usuario.
 - Descripción: Persona que utilizará la aplicación.
 - Características: Es el usuario estándar de la aplicación.
 - Relaciones: Ninguna.
 - Atributos: Ninguno.
 - Comentarios: El usuario no tiene ningún conocimiento previo sobre la aplicación aunque probablemente pueda estar familiarizado con INDI y la astronomía

5.2.2. Descripción casos de uso

- **CU-1.** Añadir una conexión.
 - **Actores:** Usuario.
 - **Tipo:** Primario, esencial.
 - **Referencias:**
 - **Precondición:**
 - **Postcondición:** La nueva conexión será añadida a la lista y guardada.
 - **Autor:** Jaime Torres Benavente.
 - **Versión:** 1.0.
 - **Propósito:** Añadir una nueva conexión.
 - **Resumen:** El usuario llenará una serie de campos y marcará unas opciones para añadir una nueva conexión a la lista.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa el botón para añadir una nueva conexión.	
		2 El sistema muestra el formulario para añadir nuevas conexiones.
3	Usuario: Rellena los campos del formulario, marca las opciones y pulsa en el botón de añadir.	
		4 El sistema almacena la conexión y la añade a la lista de conexiones.

Tabla 5.1: CU-1. Añadir nueva conexión.

▪ **CU-2.** Editar una conexión.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** La conexión debe existir y estar en estado “desconectada”.
- **Postcondición:** La conexión sera editada y guardada.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una conexión existente.
- **Resumen:** El usuario llenará una serie de campos y marcará unas opciones para editar la conexión.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa el botón para desplegar el menú lateral.	
		2 El sistema muestra el menú lateral con las conexiones, su estado y sus dispositivos.
3	Usuario: Pulsa el botón editar para una conexión concreta.	
		4 El sistema muestra el formulario para la edición de una conexión.
5	Usuario: edita los campos del formulario, marca las opciones y pulsa en el botón de editar.	
		6 El sistema almacena la conexión

Tabla 5.2: CU-2. Editar una conexión.

■ **CU-3.** Borrar conexiones.

- **Actores:** Usuario.
- Tipo: Primario, esencial.
- **Referencias:**
- **Precondición:** Las conexiones deben existir.
- **Postcondición:** Las conexiones serán borradas.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Borrar conexiones.
- **Resumen:** El usuario seleccionará de entre las conexiones disponibles, una selección para que sean borradas.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa el botón para desplegar el menú superior derecho.	
		2 El sistema muestra el menú superior.
3	Usuario: Pulsa el botón para borrar conexiones.	
		4 El sistema muestra el formulario con una lista de todas las conexiones
5	Usuario: selecciona aquellas conexiones que desee borrar.	
		6 El sistema Borra las conexiones seleccionadas

Tabla 5.3: CU-3. Borrar conexiones.

- **CU-4.** Editar los ajustes.

- **Actores:** Usuario.
- **Tipo:** Opcional, esencial.
- **Referencias:**
- **Precondición:**
- **Postcondición:** Los ajustes serán guardados.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar los ajustes.
- **Resumen:** El usuario establecerá las distintas configuraciones.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa el botón para desplegar el menú superior derecho.		
		2	El sistema muestra el menú superior.
3	Usuario: Pulsa el botón para ver los ajustes.		
		4	El sistema muestra la pantalla cin con una lista de ajustes y su estado.
5	Usuario: editada los ajustes que considere.		
		6	El sistema guarda el estado de cada configuración.

Tabla 5.4: CU-4. Editar los ajustes.

■ **CU-5.** Conectarse a un servidor.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** La conexión debe haber sido añadida previamente. La conexión debe estar desconectada.
- **Postcondición:** Se añaden los dispositivos de la conexión a la lista (si los hubiese).
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Conectarse a un servidor.
- **Resumen:** El usuario se conectará a un servidor.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa el botón para desplegar el menú lateral izquierdo.		
		2	El sistema muestra el menú lateral izquierdo con la lista de conexiones y dispositivos.
3	Usuario: Pulsa el botón para conectarse.		
		4a	El sistema esconde el menú lateral y realiza la conexión. A partir de ahora la conexión se mantiene en segundo plano para refrescar los dispositivos añadidos o borrados que serán listados al desplegar el menú lateral izquierdo.

Tabla 5.5: CU-4. Conectarse a un servidor.

Curso alterno	
4b	Si el servidor no responde, o los datos de la conexión no son correctos, el sistema muestra una alerta para informar al usuario.

Tabla 5.6: Curso alterno de CU-5. Conectarse a un servidor.

■ **CU-6.** Desconectarse de un servidor.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** La conexión debe haber sido añadida previamente. La conexión debe estar conectada.
- **Postcondición:** Se borran de la lista los dispositivos (si los hubiera).
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Desconectarse de un servidor.
- **Resumen:** El usuario se desconecta de un servidor.

Curso normal			
Actor		Sistema	
1	Usuario: Pulsa el botón para desplegar el menú lateral izquierdo.		
		2	El sistema muestra el menú lateral izquierdo con la lista de conexiones y dispositivos.
3	Usuario: Pulsa el botón para desconectarse.		
		4	El sistema esconde el menú lateral y realiza la desconexión. .

Tabla 5.7: CU-6. Desconectarse de un servidor.

■ **CU-7.** Salir de la aplicación.

- **Actores:** Usuario.
- **Tipo:** Secundario, esencial.
- **Referencias:**
- **Precondición:** La aplicación debe estar iniciada.
- **Postcondición:** Se cierran todas las conexiones, hebras y procesos liberando todos los recursos de la aplicación.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Salir de la aplicación.
- **Resumen:** El usuario cierra la aplicación explícitamente.

Curso normal			
Actor		Sistema	
1	Usuario: Pulsa el botón para desplegar el menú superior derecho.		
		2	El sistema muestra el menú superior.
3	Usuario: Pulsa el botón salir.		
		4	El sistema comprueba cada conexión y se desconecta de todas, cerrando todas las hebras. Después cierra la aplicación.

Tabla 5.8: CU-7. Salir de la aplicación.

■ **CU-8.** Mostrar dispositivo.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** La conexión debe haber sido añadida previamente. La conexión debe estar conectada.
- **Postcondición:** Se listan todas las propiedades del dispositivo. Cualquier cambio en las propiedades será mostrado en la lista en tiempo real.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Mostrar las propiedades de un dispositivo.
- **Resumen:** El usuario selecciona un dispositivo para mostrar la lista de sus propiedades.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa el botón para desplegar el menú lateral izquierdo.	
		2 El sistema muestra el menú lateral izquierdo con la lista de conexiones y dispositivos.
3	Usuario: Pulsa sobre el dispositivo deseado.	
		4 El sistema esconde el menú lateral y muestra una pantalla tabulada con todas las vistas especiales que tenga el dispositivo (si las tiene) más la vista por defecto con la lista de propiedades y la ayuda general de la aplicación.

Tabla 5.9: CU-8. Mostrar dispositivo.

■ **CU-9.** Cambiar vista de dispositivo.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:**
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Cambiar entre las vistas de un dispositivo.
- **Resumen:** El usuario cambia de vista de un dispositivo entre las disponibles.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el nombre de la pestaña correspondiente o desliza el dedo por la pantalla.	
		2 El sistema muestra la vista correspondiente.

Tabla 5.10: CU-9. Cambiar vista de dispositivo.

■ **CU-10.** Editar propiedad *text*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *text*.
- **Resumen:** El usuario pulsará sobre una propiedad *text* para editarla.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa sobre una propiedad de tipo <i>text</i> .		
		2a	El sistema muestra la vista para la edición de las propiedades <i>text</i> con todos los elementos de la propiedad concreta.
3	Usuario: edita los elementos que deseé y pulsa el botón de actualizar		
		4	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.11: CU-10. Editar una propiedad *text*.

Curso alterno	
2b	Si la propiedad es de solo lectura, el sistema muestra una alerta.

Tabla 5.12: Curso alterno de CU-10. Editar una propiedad *text*.

■ **CU-11.** Editar propiedad *number*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *number*.
- **Resumen:** El usuario pulsará sobre una propiedad *number* para editarla.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa sobre una propiedad de tipo <i>number</i> .		
		2a	El sistema muestra la vista para la edición de las propiedades <i>number</i> con todos los elementos de la propiedad concreta.
3	Usuario: edita los elementos que deseé y pulsa el botón de actualizar		
		4a	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.13: CU-11 Editar una propiedad *number*.

Curso alterno	
2b	Si la propiedad es de solo lectura, el sistema muestra una alerta.
4b	Si algún valor de algún elemento editado está fuera de rango o tiene un formato erróneo, el sistema mostrará una alerta.

Tabla 5.14: Curso alterno de CU-11. Editar una propiedad *number*.

■ **CU-12.** Editar propiedad *switch*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *switch*.
- **Resumen:** El usuario pulsará sobre una propiedad *switch* para editarla.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa sobre una propiedad de tipo <i>switch</i> .		
		2a	El sistema muestra la vista para la edición de las propiedades <i>switch</i> con todos los elementos de la propiedad concreta.
3	Usuario: edita los elementos que deseé y pulsa el botón de actualizar		
		4	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.15: CU-12 Editar una propiedad *switch*.

Curso alterno	
2b	Si la propiedad es de solo lectura, el sistema muestra una alerta.

Tabla 5.16: Curso alterno de CU-12. Editar una propiedad *switch*.

■ **CU-13.** Editar propiedad *blob*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *blob*.
- **Resumen:** El usuario pulsará sobre una propiedad *blob* para editarla.

Curso normal			
Actor		Sistema	
1	Usuario: Pulsa sobre una propiedad de tipo <i>blob</i> .		
		2a	El sistema muestra la vista para la edición de las propiedades <i>blob</i> con todos los elementos de la propiedad concreta.
3	Usuario: edita los elementos que deseé y pulsa el botón de actualizar		
		4	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.17: CU-13 Editar una propiedad *blob*.

Curso alterno	
2b	Si la propiedad es de solo lectura, el sistema muestra una alerta.

Tabla 5.18: Curso alterno de CU-13. Editar una propiedad *blob*.

■ **CU-14.** Editar propiedad *connection*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *connection*.
- **Resumen:** El usuario pulsará sobre una propiedad *connection* para editarla.

Curso normal			
Actor		Sistema	
<i>1</i>	Usuario: Pulsa sobre una propiedad de tipo <i>connection</i> .		
		<i>2</i>	El sistema muestra la vista para la edición de las propiedades <i>connection</i> con un <i>switch</i> para conectar o desconectar la propiedad.
<i>3</i>	Usuario: pulsa sobre el <i>switch</i> para conectar o desconectar la propiedad y después pulsa en el botón actualizar.		
		<i>4</i>	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.19: CU-14 Editar una propiedad *connection*.

■ **CU-15.** Editar propiedad *abort*.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
- **Postcondición:** La propiedad es editada y enviada al servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Editar una propiedad *abort*.
- **Resumen:** El usuario pulsará sobre una propiedad *abort* para editarla.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa sobre una propiedad de tipo <i>abort</i> .		
		2	El sistema muestra la vista para la edición de las propiedades <i>abort</i> con un botón para abortar.
3	Usuario: pulsa sobre el botón para abortar.		
		4	El sistema cierra la vista de edición y actualiza la vista de la propiedad.

Tabla 5.20: CU-15 Editar una propiedad *abort*.

■ **CU-16.** Guardar blob.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). El dispositivo debe tener una propiedad de tipo *blob*.
- **Postcondición:** Se guardará en la carpeta de la aplicación un archivo con el *blob*.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Guardar un *blob*.
- **Resumen:** El usuario guardará un *blob* que haya recibido.

Curso normal			
Actor		Sistema	
1	Usuario: Pulsa sobre el botón guardar en la vista de una propiedad de tipo <i>blob</i> .		
		2a	El sistema guarda el <i>blob</i> en la carpeta de la aplicación y muestra un mensaje por pantalla informando del éxito de la acción.

Tabla 5.21: CU-16 Guardar un *blob*.

Curso alterno	
2b	Si la propiedad <i>blob</i> no tiene ningún dato que guardar se muestra una alerta informando al usuario.

Tabla 5.22: Curso alterno de CU-16. Guardar un *blob*.

■ **CU-17.** Mostrar un blob.

- **Actores:** Usuario.
- **Tipo:** Primario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). El dispositivo debe tener una propiedad de tipo *blob*.
- **Postcondición:** Se pasará al sistema operativo **Android** el archivo para que muestre una lista de posibles aplicaciones instaladas que puedan manejarlo.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Mostrar un *blob*.
- **Resumen:** El usuario guardará un blob que haya recibido.

Curso normal			
	Actor	Sistema	
1	Usuario: Pulsa sobre el botón mostrar en la vista de una propiedad de tipo <i>blob</i> .		
		2a	El sistema guarda el blob en la carpeta de la aplicación y envía el archivo al sistema operativo para poder mostrar el blob con una aplicación adecuada según el formato.

Tabla 5.23: CU-17 Mostrar un *blob*.

Curso alterno	
2b	Si la propiedad blob no tiene ningún dato que mostrar se muestra una alerta informando al usuario.

Tabla 5.24: Curso alterno de CU-17. Mostrar un *blob*.

■ **CU-18.** Ver el log.

- **Actores:** Usuario.
- **Tipo:** Opcional, esencial.
- **Referencias:**
- **Precondición:**
- **Postcondición:**
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Ver el log.
- **Resumen:** El usuario podrá ver el log de cada una de las conexiones guardadas.

Curso normal			
Actor		Sistema	
1	Usuario: Pulsa el botón para desplegar el menú superior derecho.		
		2	El sistema muestra el menú superior.
3	Usuario: Pulsa el botón para ver el log.		
		4	El sistema muestra una vista tabulada con cada uno de los log que corresponden a cada una de las conexiones guardadas

Tabla 5.25: CU-18. Ver el log.

■ **CU-19.** Ocultar una propiedad.

- **Actores:** Usuario.
- **Tipo:** Secundario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). La propiedad debe estar visible.
- **Postcondición:** La propiedad se marca como oculta.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Ocultar una propiedad.
- **Resumen:** El usuario ocultará una propiedad visible.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el ícono de visibilidad para ocultar una propiedad	
		<p>2a</p> <p>El sistema marca la propiedad como “no visible” y cambia el ícono de de visibilidad en la vista de la propiedad.</p>

Tabla 5.26: CU-19. Ocultar una propiedad.

Curso alterno	
2b	Si la lista de propiedades tiene marcada la opción de “ocultar propiedades”, la propiedad desaparecerá de la vista. Si el grupo al que pertenece la propiedad no tiene ninguna propiedad visible, también desaparecerá de la vista.

Tabla 5.27: Curso alterno de CU-19. Ocultar una propiedad.

■ **CU-20.** Mostrar una propiedad.

- **Actores:** Usuario.
- **Tipo:** Secundario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). La propiedad debe estar “no visible”. Debe estar activada la opción de “ver todas las propiedades” para que las que estén marcadas como ocultas se añadan también a la vista
- **Postcondición:** La propiedad se marcará como visible.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Mostrar una propiedad.
- **Resumen:** El usuario marcará una propiedad como visible.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el ícono de visibilidad para marcar una propiedad como visible.	
		2 El sistema marca la propiedad como visible y cambia el ícono de visibilidad en la vista de la propiedad.

Tabla 5.28: CU-20. Mostrar una propiedad.

- **CU-21.** Activar la visibilidad de todas las propiedades ocultas.
 - **Actores:** Usuario.
 - **Tipo:** Secundario, esencial.
 - **Referencias:**
 - **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
 - **Postcondición:** Se muestran todas las propiedades sea cual sea su visibilidad.
 - **Autor:** Jaime Torres Benavente.
 - **Versión:** 1.0.
 - **Propósito:** Activar la visibilidad de todas las propiedades ocultas.
 - **Resumen:** El usuario activa la visibilidad de todas las propiedades, estén marcadas como ocultas o no.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el botón flotante para activar la visibilidad de todas las propiedades.	
		2 El sistema refresca la lista de propiedades añadiéndolas todas aunque su estado sea “no visible”.

Tabla 5.29: CU-21. Activar la visibilidad de todas las propiedades ocultas.

- **CU-22.** Desactivar la visibilidad de todas las propiedades ocultas.
 - **Actores:** Usuario.
 - **Tipo:** Secundario, esencial.
 - **Referencias:**
 - **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8).
 - **Postcondición:** Se ocultan todas las propiedades cuyo estado sea “no visible”.
 - **Autor:** Jaime Torres Benavente.
 - **Versión:** 1.0.
 - **Propósito:** Desactivar la visibilidad de todas las propiedades ocultas.
 - **Resumen:** El usuario desactiva la visibilidad de todas las propiedades con el estado “no visible”.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el botón flotante para desactivar la visibilidad de todas las propiedades ocultas.	
		2 El sistema refresca la lista de propiedades mostrando solo aquellas que estén visibles.

Tabla 5.30: CU-21. Desactivar la visibilidad de todas las propiedades ocultas.

■ **CU-23.** Expandir un grupo.

- **Actores:** Usuario.
- **Tipo:** Secundario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). El grupo debe estar contraído.
- **Postcondición:** se expande el grupo seleccionado.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Expandir un grupo.
- **Resumen:** El usuario Expande uno de los grupos de la lista que esté contraído.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el nombre del grupo que desee expandir.	
		2 El sistema refresca la lista de propiedades mostrando el grupo expandido y todas las propiedades que contenga.

Tabla 5.31: CU-23. Expandir un grupo.

■ **CU-24.** Contraer un grupo.

- **Actores:** Usuario.
- **Tipo:** Secundario, esencial.
- **Referencias:**
- **Precondición:** El usuario debe haber seleccionado un dispositivo (CU-8). El grupo debe estar expandido.
- **Postcondición:** se contrae el grupo seleccionado.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Contraer un grupo.
- **Resumen:** El usuario contrae uno de los grupos de la lista que esté expandido.

Curso normal		
	Actor	Sistema
<i>1</i>	Usuario: Pulsa sobre el nombre del grupo que desee contraer.	
		<i>2</i> El sistema refresca la lista de propiedades mostrando el grupo contraído.

Tabla 5.32: CU-24. Contraer un grupo.

■ **CU-25.** Abrir la aplicación.

- **Actores:** Usuario.
- **Tipo:** primaria, esencial.
- **Referencias:**
- **Precondición:**
- **Postcondición:** La aplicación queda iniciada. Si alguna conexión tiene activada la opción de “autoconectar”, se realizará la conexión con el servidor.
- **Autor:** Jaime Torres Benavente.
- **Versión:** 1.0.
- **Propósito:** Iniciar la aplicación.
- **Resumen:** El usuario inicia la aplicación.

Curso normal		
	Actor	Sistema
1	Usuario: Pulsa sobre el icono de la aplicación.	
		2 El sistema carga las configuraciones o las crea en caso de no existir. Después muestra la interfaz de la aplicación mostrando la sección de ayuda.

Tabla 5.33: CU-25. Iniciar la aplicación.

5.3. Diagramas de casos de uso

Los diagramas de casos de uso sirven para representar las relaciones que existen entre los diferentes actores y el sistema. Dado que en nuestro caso solo hay un actor, todos los casos de uso son iniciados por él.



Figura 5.1: Diagrama de casos de uso.

5.4. Storyboard

A continuación, en la figura 5.2 se muestra el storyboard de la aplicación, donde se pueden ver las distintas pantallas por las que puede pasar la aplicación.

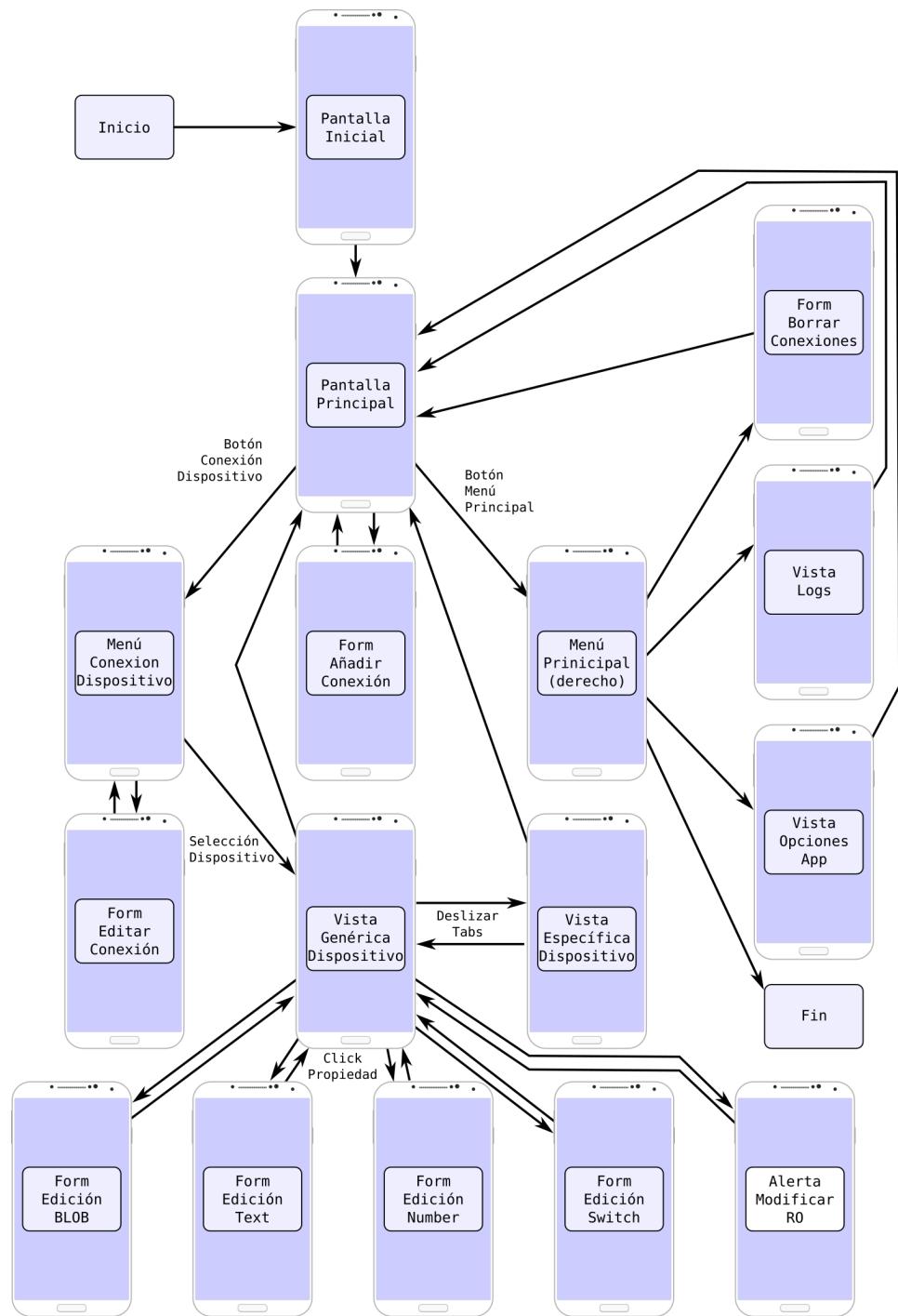


Figura 5.2: Storyboard de la aplicación.

Capítulo 6

Diseño e Implementación

El proyecto tiene dos partes claramente diferenciadas a la hora de plantear el diseño:

- **Diseño de clases.**
- **Diseño de las interfaces de usuario.**
- **Mecanismo de adición de vistas.**

Además, hay que tener en cuenta que uno de los objetivos principales es que la aplicación sea fácilmente extensible y que permita añadir nuevas interfaces de usuario para propiedades y dispositivos por lo que es parte importante de la fase de diseño, en la que se describirán todos detalladamente.

6.1. Diseño de clases

En todo proyecto de software es muy importante diseñar correctamente las clases antes de comenzar la fase de implementación. Un mal diseño puede provocar retrasos en la fase de implementación, incluso obligando a retroceder y rediseñarlas.

Podemos dividir el diseño en cuatro bloques:

- **Diseño de las actividades de Android.**
- **Diseño del cliente INDI.**
- **Diseño de las clases manejadoras de propiedades.**
- **Diseño de las clases manejadoras de dispositivos.**

6.1.1. Diseño de las actividades de Android

Las actividades en android son el cuerpo principal de las aplicaciones. Son ejecutadas en la hebra principal del sistema y gestionan la interfaz principal. Se pueden tener tantas actividades como se desee. En nuestro caso tenemos dos actividades:

- **Front activity**
- **Main activity**

Front activity

Esta actividad inicia la aplicación, mostrando una pantalla de inicio con la información como portada. Una vez lanzada la aplicación esta actividad no volverá a ejecutarse a no ser que el sistema cierre la aplicación o que lo haga el usuario explícitamente. Como podemos ver en el diagrama de la figura 6.1, la actividad solo tiene una clase que hereda de `AppCompatActivity`. Esta clase pertenece a la biblioteca de compatibilidad de `android`. Además, esta actividad tiene un objeto de la clase `UpdateProgress`. Esta clase sirve para ejecutar código en una hebra separada en la que controlar el progreso.

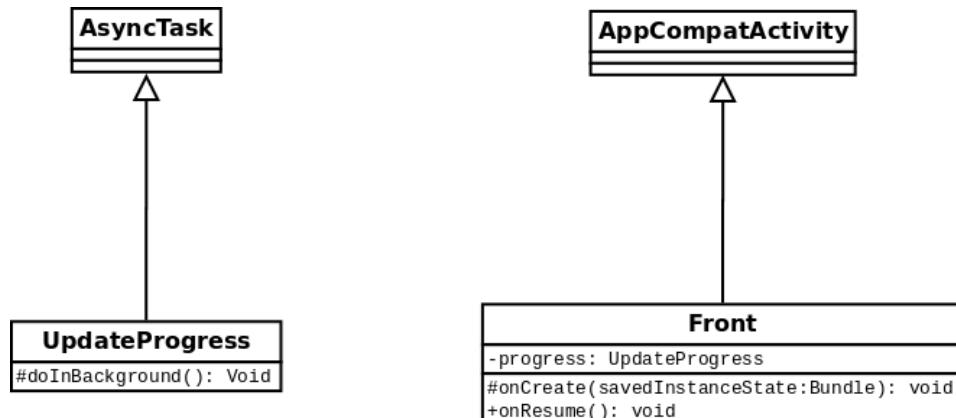


Figura 6.1: Diagrama de clases de la actividad front

Main activity

La actividad principal es el núcleo de la aplicación. Esta clase es la más importante ya que su ciclo de vida condiciona el ciclo de vida de la aplicación. Es por ello que esta clase es la más compleja. En la figura 6.2 podemos ver el diagrama de clases. Las clases que no contienen ninguna especificación

pertenecen a **Android** y simplemente se añaden para indicar las relaciones que las clases implementadas.

Dado que esta clase es la responsable de la visualización de los distintos menús, necesitamos declarar objetos de las clases que representan cada uno de los elementos visuales principales, tales como `NavigationView`[12] o `TabLayout`[13].

Por otro lado, la actividad implementa una serie de escuchadores que le permiten capturar los eventos disparados por los distintos botones de la interfaz de usuario para realizar las acciones oportunas.

Además de estas clases, se ha diseñado la clase `Settings` que representa las configuraciones generales de la aplicación: notificaciones y carpeta por defecto.

En el diagrama también podemos ver la clase `Connection`. Esta clase representa las conexiones con el servidor y es por ella que se describen en el apartado correspondiente.

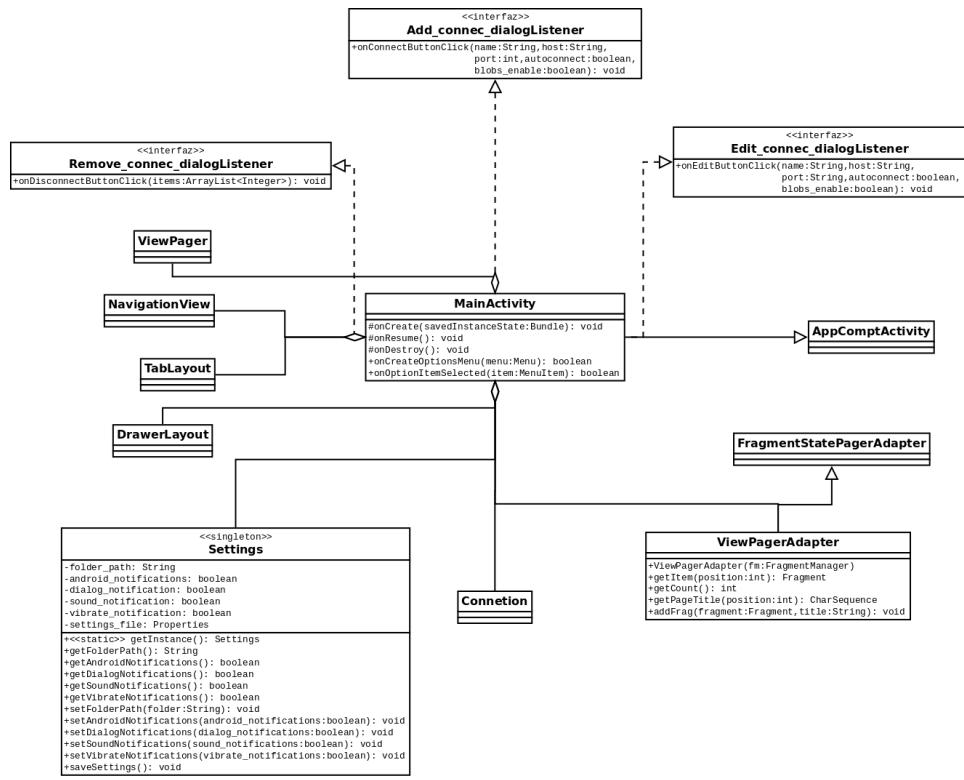


Figura 6.2: Diagrama de clases de la actividad principal

6.1.2. Diseño del cliente INDI

La aplicación se basa en integrar la biblioteca “**INDI for Java**” para poder crear un cliente que nos permita conectarnos a cualquier servidor. Las conexiones se hacen creando conexiones TCP/IP a través de la red. **Android** establece unas restricciones muy fuertes respecto a la apertura y cierre de **sockets** de red. Dado que la arquitectura del sistema esta basada en una única hebra principal que gestiona la interfaz de usuario, si iniciamos algún proceso que pueda bloquear dicha hebra, bloquearíamos todo el sistema. Por ello cualquier proceso de comunicación debe ejecutarse en una hebra secundaria. Además solo se pueden ejecutar acciones sobre la interfaz en la hebra principal.

Con estas restricciones, el primer paso necesario es extraer todo el código relativo a las conexiones a una hebra por conexión. Para ello se ha diseñado la clase **Connection**. El objetivo principal de esta clase es lanzar en una hebra la apertura de la conexión y el intercambio de información con el servidor. Pero para poder utilizar esa información y mostrarla en pantalla necesitamos ejecutar en la hebra principal dichas acciones.

Android nos facilita una clase para resolver este problema, aparentemente sin solución. La clase `AsyncTask` tiene la peculiaridad de permitir ejecutar código en una hebra a parte y a la vez enviar información a la hebra principal para gestionarla adecuadamente. Por ello la clase `Connection` lanza una hebra pero almacena en sus atributos el resultado de la conexión al servidor, permitiendo a la actividad principal procesar esos datos y mostrarlos adecuadamente. La actividad principal tendrá tantos objetos `Connection` como conexiones se hayan añadido en la interfaz de usuario. Cada objeto `Connection` creará una cliente `INDI` que le enviará todos los dispositivos y atributos que tenga y le irá informando de cualquier cambio para que estos se reflejen en la interfaz de usuario. Todas estas hebras se lanzan en paralelo, de forma que no ralenticen las acciones en la interfaz de usuario.

Por otra parte, para que todo funcione debemos diseñar una clase cliente **INDI** para crear la conexión, y escuchar cualquier cambio en propiedades, dispositivos o en la propia conexión. Por ello esta clase implementa las tres interfaces de **INDI**

Para facilitar el diseño, también se ha creado una clase `device`. Esta clase se utiliza para procesar las propiedades de un dispositivo **INDI**. Cada propiedad pertenece a un grupo pero a priori no puedes conocer que grupos hay. Además las propiedades no llegan según un orden por lo que hay que comprobar por cada una a que grupo pertenece, si el grupo existe ya o si hay que crearlo. De la misma forma, cuando se borra una propiedad hay que comprobar si era la última de su grupo, en cuyo caso habrá que borrarlo. La clase `device` facilita estas operaciones, añadiendo una capa de abstracción más para poder obtener las propiedades organizadas y listas para ser mostradas en la interfaz de usuario.

Finalmente, necesitamos representar la lista de propiedades y dispositivos. Para ello usamos las *listas expandibles de Android*[14]. Estas listas nos permiten tener dos niveles. En el primer nivel mostramos los grupos y en el segundo los elementos. Internamente tenemos una lista o *adaptador* de propiedades que añadimos a la clase `PropertyArrayAdapter`. Cada objeto de esta clase representa un dispositivo con todas sus propiedades.

En este punto cabe destacar que podemos tener propiedades ocultas. Dichas propiedades no deben ser añadidas al adaptador, ya que todos los elementos de este son mostrados. Para controlarlo, la clase `Connection` es la encargada de construir los *adaptadores* a partir de la información del objeto de la clase `IndiClient`. Por ello en la clase `Connection` gestionamos las propiedades que están ocultas para no agregarlas al *adaptador* que le corresponda.

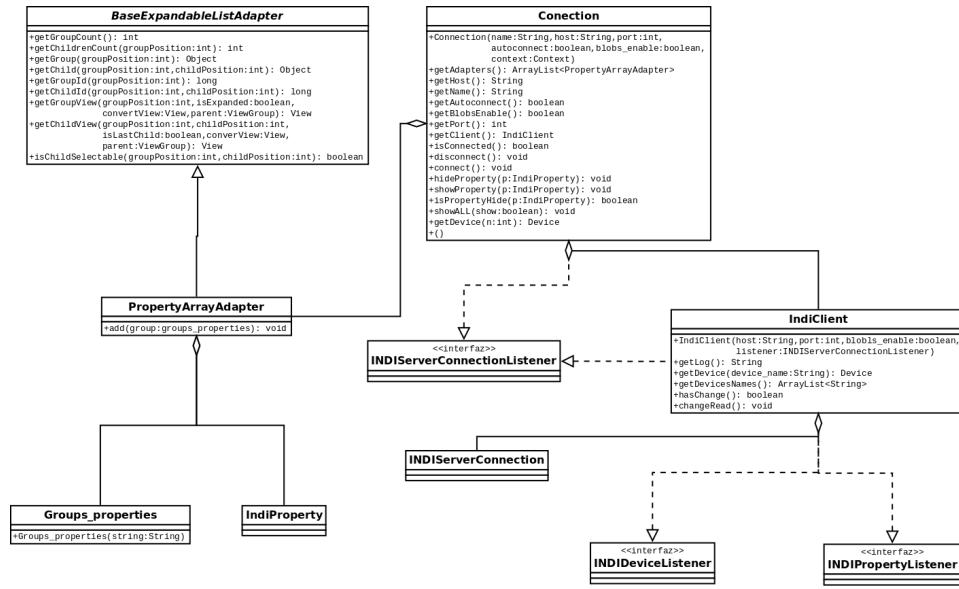


Figura 6.3: Diagrama de clases asociadas a INDI

6.1.3. Diseño de las clases manejadoras de propiedades

Como se explicó en la introducción, **INDI** maneja 5 tipos de propiedades:

- **Text.**
- **Number.**
- **Switch.**
- **Blob.**
- **Light.**

Para manejar cada una de estas propiedades se crea una clase que recibirá un objeto `INDIProperty` (del que heredan todos los tipos) y según el tipo informarán de que pueden manejar dicha propiedad y construirán las interfaces de usuario para mostrarla y editarla.

Gracias a la creación de la interfaz de Java `UIPropertyManager` podemos añadir más manejadores de propiedades. Para ilustrar su uso se han creado dos manejadores más:

- **Connection.**
- **Abort.**

Estas dos propiedades son de tipo **Switch** pero tienen la peculiaridad de que siempre tienen la misma estructura para cualquier dispositivo que las use: mismo número de elementos, mismo nombre para cada elemento, etc. Por ello podemos analizar la propiedad recibida y ver si es de esos tipos, informando de que podemos manejarla, y construyendo vistas específicas para esas propiedades (como son de tipo **Switch** su vista por defecto sería la de todas las propiedades de este tipo).

En la figura 6.4 podemos ver el diagrama de clases que ilustra la creación de los manejadores, implementando las interfaces para manejar propiedades y, adicionalmente, para manejar la pulsación sobre un objeto **View** de Android.

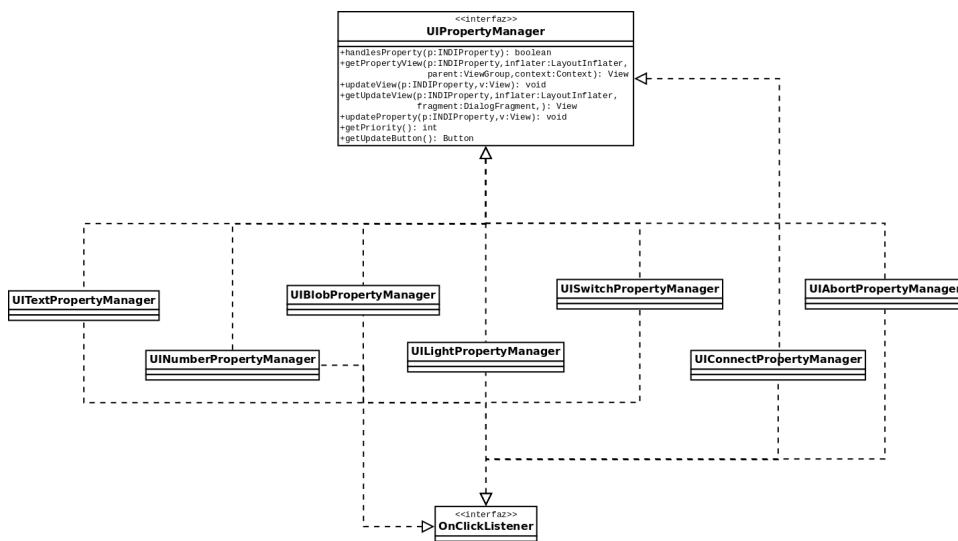


Figura 6.4: Diagrama de clases asociadas a los manejadores de propiedades

6.1.4. Diseño de las clases manejadoras de dispositivos

Finalmente, necesitamos manejar y decidir como mostramos una propiedad y sus elementos. Para implementar esta funcionalidad se ha optado por utilizar los elementos visuales de Android **tabs** que permiten mostrar vistas tabuladas. Con esto, vamos a crear una vista por defecto para cualquier dispositivo. Esta vista es la utilizada en las secciones anteriores: *las listas expandibles de android*.

En la figura 6.5 podemos ver que tenemos una clase **DefaultDeviceView** que se mostrará por defecto para todos los dispositivos. Este es el comportamiento normal de la aplicación.

Siguiente con el objetivo principal de facilitar la extensibilidad para añadir nuevas vistas de dispositivo, se ha diseñado una clase abstracta, **DeviceView**, para facilitar la creación de nuevas vistas. Solo tenemos que crear una clase que herede de esta clase abstracta e implementar los métodos. Al ser una clase que hereda de **Fragment**, dentro de ella tenemos libertad para mostrar las propiedades del dispositivo para personalizarlo. Cada nueva vista sería una tabulación más en la vista tabulada.

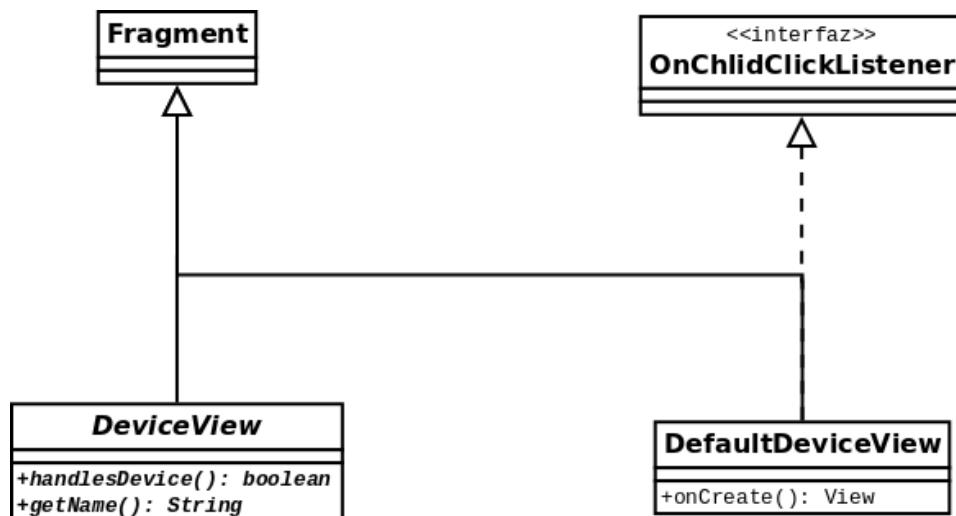


Figura 6.5: Diagrama de clases asociadas a los manejadores de dispositivos

6.2. Diseño de las interfaces de usuario

Las interfaces de usuario deben ser objeto de un cuidadoso diseño dado que estamos construyendo una aplicación móvil, y el éxito dependerá en

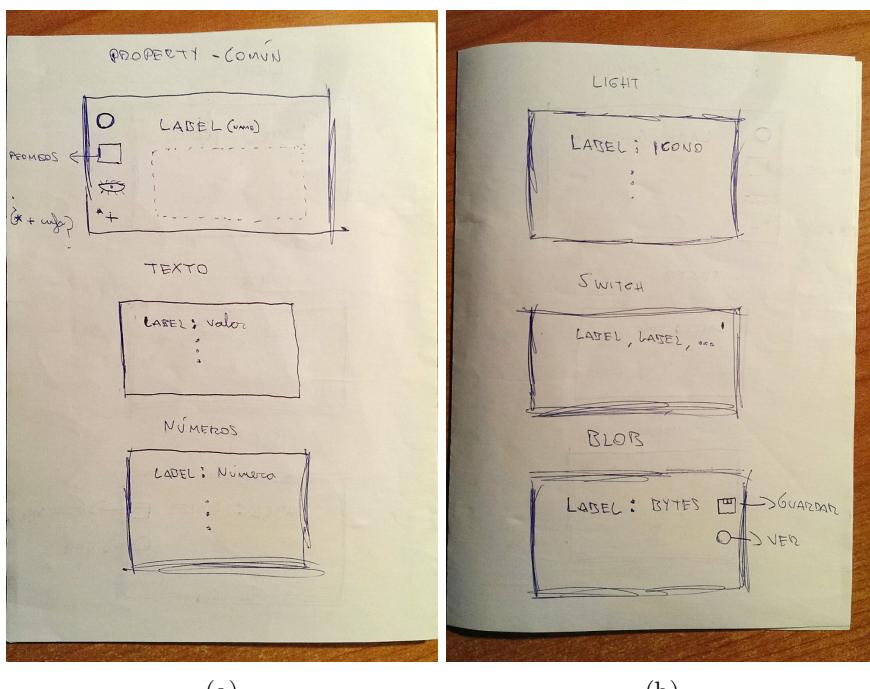


Figura 6.6: Bocetos de interfaces

gran medida de una buena interfaz de usuario que sea útil, clara y fácil de usar.

A lo largo del diseño de las interfaces de usuario, se fueron mejorando sucesivamente las distintas vistas desde bocetos en papel (figuras 6.6a y 6.6b) hasta primeras versiones ya implementadas en la aplicación (figuras 6.7 y 6.8).

En cada una de las iteraciones del proceso **Scrum** se fueron rediseñando y mejorando las interfaces de usuario hasta llega a las vistas finales descritas en los siguientes apartados.

Para poder abordar mejor el diseño, se han separados las interfaces en los siguientes grupos:

- **Interfaz principal de la aplicación.**
- **Interfaz para listar las propiedades.**
- **Interfaz de propiedad.**
- **Interfaz de diálogos**
- **Interfaz del log**

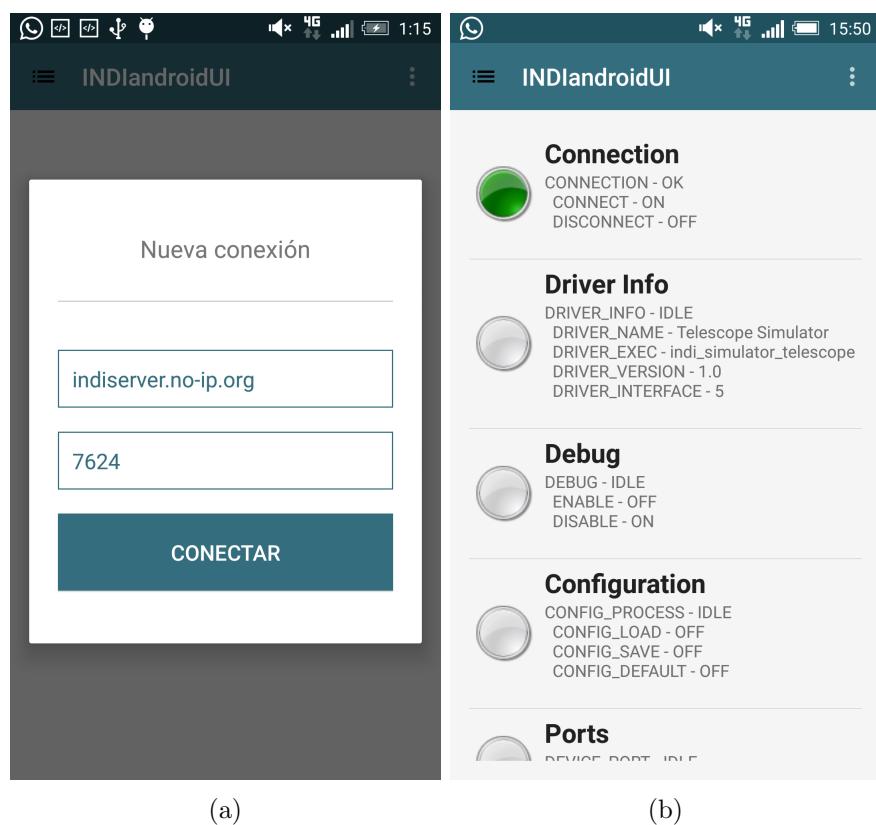


Figura 6.7: Capturas de interfaces

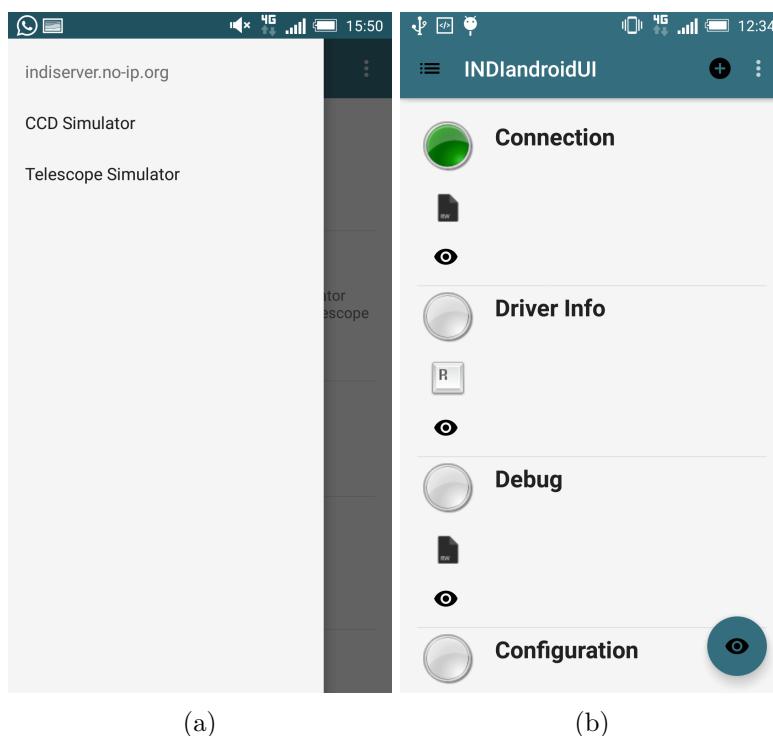


Figura 6.8: Capturas de interfaces

- **Interfaz de los ajustes**

6.2.1. Interfaz principal de la aplicación

La interfaz principal es la más importante de cara al usuario. Define como se va a organizar toda la información y la navegabilidad dentro de la aplicación.

En cada iteración se han añadido nuevas funcionalidades a la aplicación lo cual provocaba tener que definir aspectos de la interfaz de usuario principal.

Por otro lado, en un esfuerzo por seguir las directrices de **Android** y las tendencias actuales en las aplicaciones móviles, se han incorporado los elementos más actuales tales como la barra de navegación lateral con estilo *material design* o tener vistas tabuladas. En las figuras 6.9 y 6.10 podemos ver la interfaz principal de la aplicación.

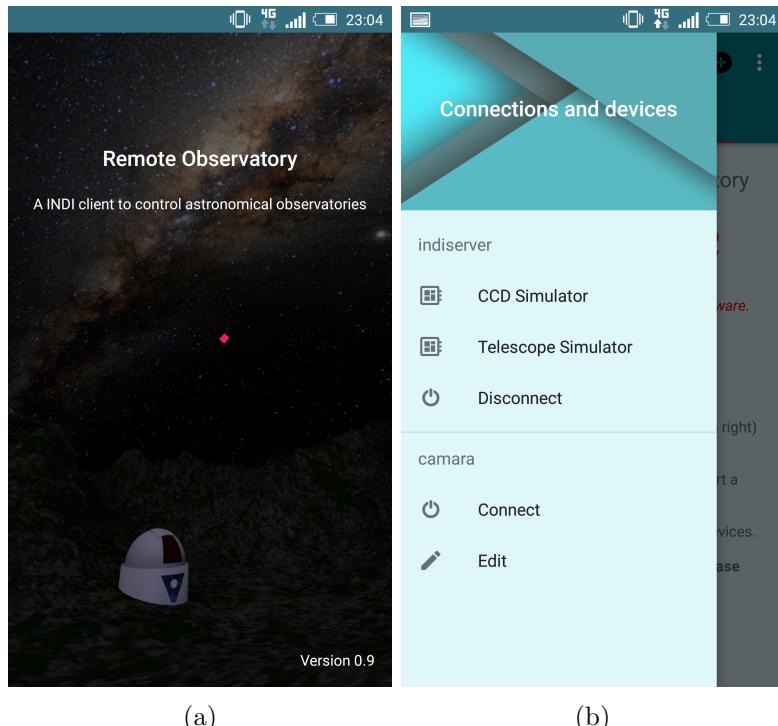


Figura 6.9: Capturas de la interfaz principal

6.2.2. Interfaz para listar las propiedades

Una vez definido el aspecto general de la aplicación y los modos de navegación, el principal objetivo es poder listar las propiedades de los dispositivos. Desde las primeras reunión tuvimos claro que debíamos usar una lista estándar de **Android**, aunque debíamos abordar un tema importante: la posibilidad de agrupar las propiedades de un mismo grupo. Hasta la **cuarta iteración** no se decidió utilizar la que a la poste sería la solución definitiva, **las listas expandibles**.

En la figura 6.11 podemos ver las listas con los grupos contraídos y extendidos. Además podemos ver como, en un nuevo esfuerzo por seguir las tendencias más actuales, se ha añadido un **botón flotante**[15] para mostrar u ocultar las propiedades con visibilidad “oculta”.

6.2.3. Interfaz de propiedad

Una vez definida la lista de las propiedades, abordamos el problema de la interfaz para una propiedad. Para ello analizamos todos los elementos en común y todas las diferencias entre los distintos tipos de propiedades. En la figura 6.9a podemos ver la parte común a todas las propiedades:

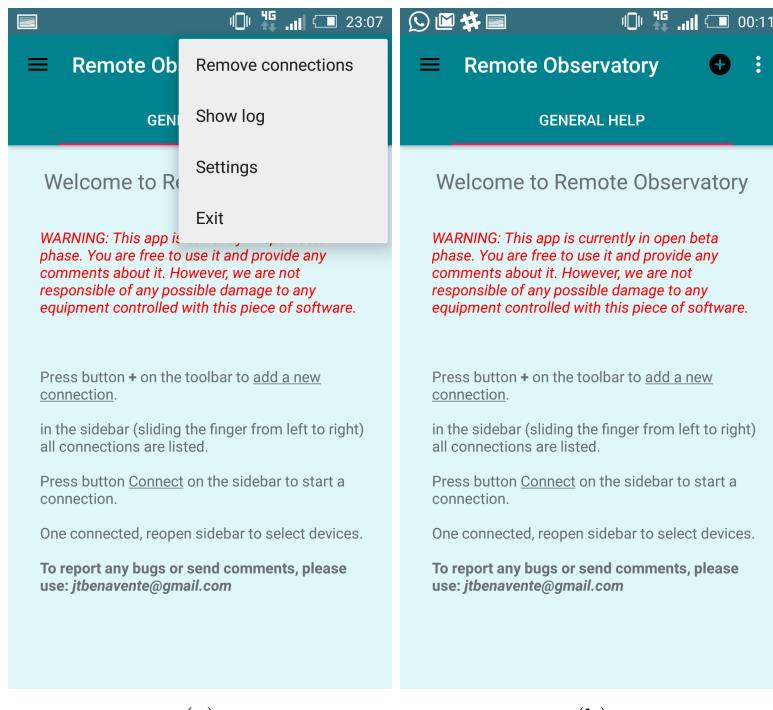


Figura 6.10: Capturas de la interfaz principal

- **Nombre de la propiedad.**
- **Estado de la propiedad.**
- **Permisos de escritura y/o lectura.**
- **Visibilidad.**

Todas las propiedades comparten estos cuatro elementos, siendo la parte central de la vista la que se adaptará según el tipo.

Text, Number y Switch

De cara a los items de la lista, las propiedades *text* y *number* son idénticas salvo que el valor de los elementos del primero son texto y del segundo son números. Cada elemento se coloca debajo del anterior creando otra lista (una propiedad puede tener uno o más elementos).

En la figura 6.12 podemos ver ambas vistas en la interfaz de las propiedades.

Los **Switch** tienen una vista similar a las dos propiedades anteriores, salvo que el valor que está en estado “on” es negro y los que están en estado “off” es gris (figura 6.12).

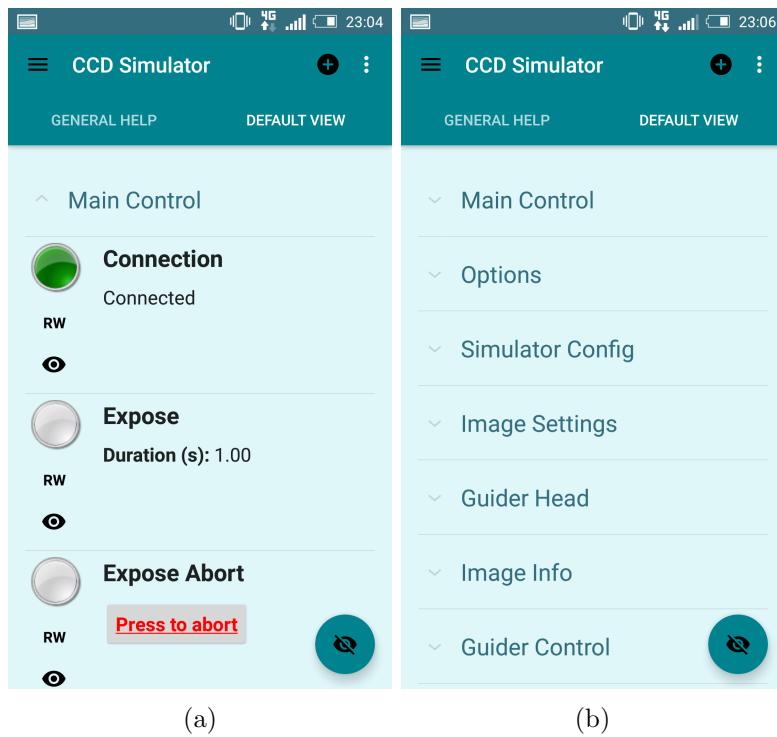


Figura 6.11: Lista de propiedades de un dispositivo

Blobs

Los **Switch** son ficheros enviados por el servidor por lo que es más interesante mostrar el tipo de archivo que recibimos y su tamaño. Como se puede ver en la figura 6.13, además de estos dos datos tenemos los botones para poder guardarlos o enviarlos a **Android** para poder abrirlos con la aplicación adecuada.

Light

Esta propiedad es muy similar a los *switch* salvo que el valor de cada uno de los elementos de la propiedad será un ícono representando una luz roja, verde, gris o amarilla.

6.2.4. Interfaz de diálogos

Los diálogos son una herramienta visual esencial que permiten superponer una vista para llamar la atención del usuario, bien sea para informar o para ejecutar alguna acción como editar propiedades.

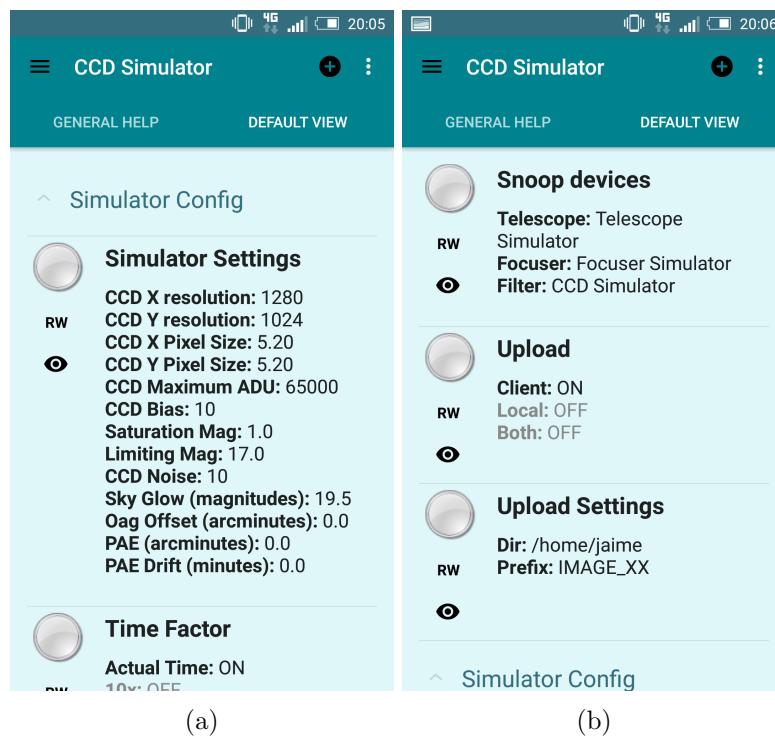


Figura 6.12: Propiedades text, number y switch

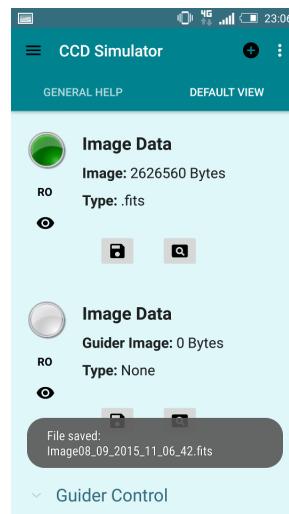


Figura 6.13: Vista de un Blob

Diálogos para editar propiedades

Estos diálogos permiten editar los elementos de aquellas propiedades que lo permitan:

- **Dialogo de edición de una propiedad Text:** Se muestra la lista de elementos con su **label** y su valor en un cuadro editable (6.14).
- **Dialogo de edición de una propiedad Number:** Se muestra la lista de elementos con su **label** y sus valores máximos y mínimos. Debajo se puede ver el campo editable con el valor (6.16).
- **Dialogo de edición de una propiedad Switch:** Se muestra cada elemento con su **label** seguido de un botón de estados para indicar si el elemento está activo **on** o no **off** (6.16) .

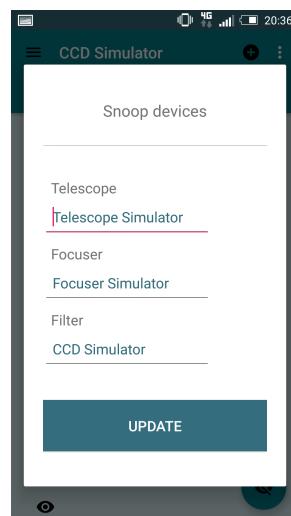


Figura 6.14: Dialogo para editar una propiedad text

Diálogos para gestión de las conexiones

Estos diálogos permiten añadir, editar o borrar conexiones. Como cada conexión se define por un nombre, un URL del servidor, un puerto y las opciones de activar la recepción de blobs y la autoconexión, todos estos campos son comunes a la vista para añadir y para editar. La vista para editar añade un botón para poder borrar el log de la conexión (figuras 6.17 y 6.18).

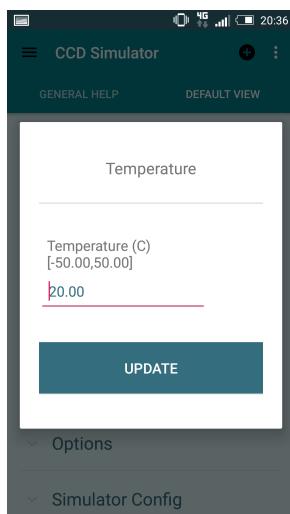


Figura 6.15: Dialogo para editar una propiedad number

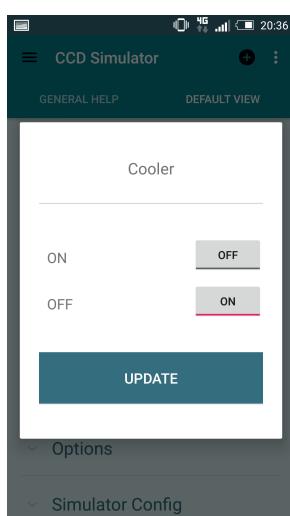


Figura 6.16: Dialogo para editar una propiedad switch

La vista para borrar conexiones es una simple lista con todas las conexiones que haya y un checkbox para marcar las que deseamos borrar (figura 6.19).

Diálogos para informar de errores o avisos

Estas vistas informan al usuario de un error al introducir datos, errores de desconexión o conexión y de alerta cuando se añaden o quitan nuevos

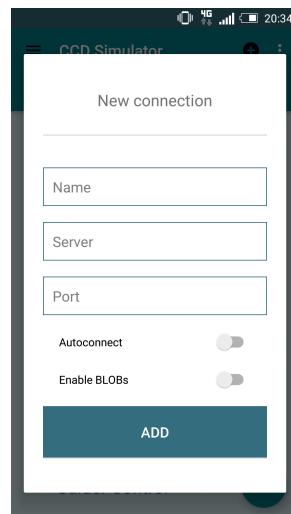


Figura 6.17: Dialogo para añadir una conexión

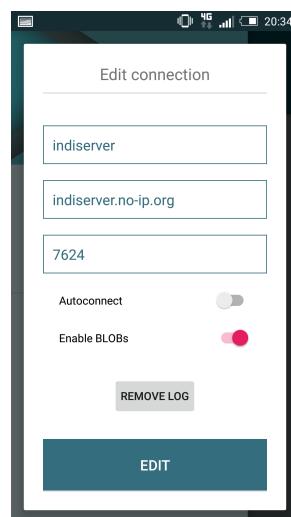


Figura 6.18: Dialogo para editar una conexión

dispositivos (6.20).

6.2.5. Interfaz de log

Esta es una interfaz muy simple para poder ver los *logs* de cada una de las conexiones guardadas en la aplicación. Para ello se crea una vista tabulada con todas ellas (figura 6.21).

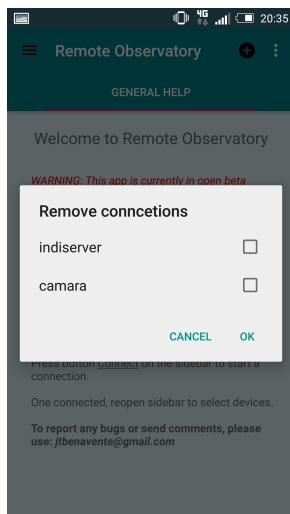


Figura 6.19: Dialogo para borrar conexiones

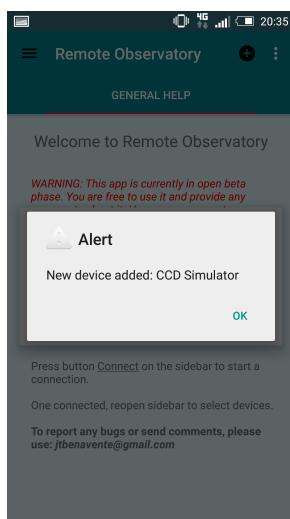


Figura 6.20: Dialogo de aviso

6.2.6. Interfaz de los ajustes

Los ajustes tienen una vista especial para poder editar las diferentes notificaciones y para poder establecer una carpeta para almacenar los archivos de la aplicación. Como podemos ver en la figura 6.22, cada una de las notificaciones tienen un elemento **switch** de **Android** para marcar el estado. Al final de la vista encontramos un campo editable para poner la ruta que deseemos para la carpeta.



Figura 6.21: Vista de los ajustes

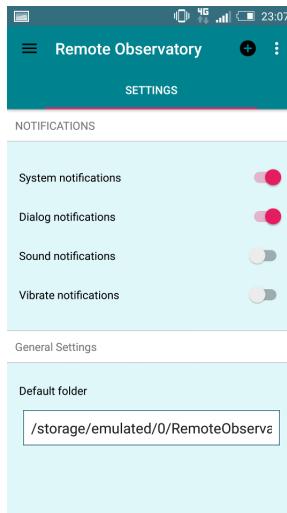


Figura 6.22: Vista de los ajustes

6.3. Mecanismo de adición de vistas

Uno de los objetivos principales del proyecto es facilitar a desarrolladores externos al proyecto, la posibilidad de agregar nuevas clases para gestionar vistas específicas de propiedades y dispositivos, liberando al desarrollador de la necesidad de conocer el código de la aplicación. Para ello se han diseñado una serie de mecanismos y pasos a seguir para poder añadir vistas personalizadas.

6.3.1. Propiedades

Para poder añadir una nueva vista debemos seguir tres pasos:

- Crear una clase que implemente la interfaz `UIPropertyManager`.
- Crear una interfaz de usuario para android en XML tanto para la vista de elemento de lista como para la vista para editar una propiedad.
- Declarar un objeto de la clase creada y añadirlo al vector de manejadores.

Interfaz `UIPropertyManager`

En la sección de diseño de clases hemos podido ver la interfaz que permite crear los distintos tipos de manejadores de propiedades.

Fragmento de código 6.1: Interfaz `UIPropertyManager`

```
1 /**
2  * Interface to handle views and indi propertys
3  */
4 public interface UIPropertyManager {
5     /**
6      * Check if this class can represent p
7      *
8      * @param p Indi property
9      * @return True/false if class can represent p
10     */
11    boolean handlesProperty (INDIProperty p);
12
13    /**
14     * Create and return a view sets with p elements
15     *
16     * @param p Indi property
17     * @param inflater Layout inflater to inflate view
18     * @param parent ViewGroup to inflate view
19     * @param context Context to allow use Activity methods
20     * @return View
21     */
22    View getPropertyView (INDIProperty p, LayoutInflator inflater,
23                          ViewGroup parent, Context context);
24
25    /**
26     * update view v with Indi property p elements
27     *
28     * @param p Indi property
29     * @param v View
30     */
31    void updateView (INDIProperty p, View v);
32
33    /**
34     * Create a new view dialog to allow set Indi property p elements
35     *
36     * @param p Indi property
37     * @param inflater Layout inflater to inflate view
38     * @return View
```

```

39     */
40     View getUpdateView(INDIProperty p, LayoutInflater inflater,
41                         DialogFragment fragment);
42
43     /**
44      * Update property with change saved at view v
45      *
46      * @param p Indi property
47      * @param v View
48      */
49     void updateProperty(INDIProperty p, View v);
50
51     /**
52      * Get priority
53      *
54      * @return priority
55      */
56     int getPriority();
57
58     /**
59      * Get update button reference
60      *
61      * @return update button
62      */
63     Button getUpdateButton();
64 }
```

Como podemos ver en el fragmento de código B.1, las funciones que debemos implementar son:

- **handlesProperty**: Esta función recibe una propiedad y debe comprobar si puede manejarla o no devolviendo un valor booleano.
- **getPropertyView**: Una vez que comprobamos que podemos manejar la propiedad, esta función debe devolvernos la vista “inflada” que se mostrará en la lista de propiedades. Los parámetros inflater, parent y context son necesarios para poder inflar la vista.
- **updateView**: Esta función es la que debe coger la vista anteriormente creada, y debe llenarla a partir de la propiedad p.
- **getUpdateView**: Esta función debe crear la vista para el modo de edición de la propiedad. Esta vista se pintará en un diálogo de tipo **Fragment**. Al igual que antes, necesitamos las variables inflater y fragment para “inflar” la vista.
- **updateProperty**: La vista para editar propiedades permite al usuario editar una propiedad. Una vez que el usuario ha terminado la edición, debemos actualizar la propiedad con los valores de la vista y enviar la información al servidor usando la función de la propiedad de **INDI**, **sendChangesToDriver**.

- **getPriority:** Esta función debe devolver una valor numérico para establecer la prioridad que tendrá la vista respecto a otras vistas o a la vista por defecto de la propiedad. Cuanto mayor sea el valor, más prioridad tendrá
- **getUpdateButton:** Esta función debe devolver un objeto “Button” si la vista tendrá el botón para actualizar, o null en caso de que no haya botón.

Una vez que tenemos la clase creada debemos diseñar la interfaz de usuario en un archivo *XML* como el del archivo del fragmento de código 6.2 que representa la vista de elemento de lista de la propiedad `Connection`. Además de este, debemos crear otra vista para mostrar en el diálogo de edición de propiedades. No hay ningún tipo de restricción para estas vistas. El desarrollador es libre de poner lo que necesite en ambas interfaces de usuario.

Fragmento de código 6.2: Vista XML en android

```
1 <?xml version="1.0" encoding="utf-8"?>
2
3 <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
4     android"
5         android:layout_width="match_parent" android:layout_height="
6             wrap_content"
7         android:padding="5dp">
8
9     <ImageView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:id="@+id/idle"
13         android:layout_marginRight="10dp"
14         android:layout_alignParentLeft="true" />
15
16     <TextView
17         android:layout_width="wrap_content"
18         android:layout_height="wrap_content"
19         android:text="New Text"
20         android:id="@+id/name"
21         android:layout_alignParentTop="true"
22         android:layout_toRightOf="@+id/idle"
23         android:layout_marginLeft="10dp"
24         android:textAppearance="@android:style/TextAppearance.Holo.
25             Large"
26         android:textStyle="bold" />
27     <TextView
28         android:layout_width="wrap_content"
29         android:layout_height="wrap_content"
30         android:text=""
31         android:id="@+id/label"
32         android:layout_below="@+id/name"
33         android:layout_toRightOf="@+id/idle"
34         android:layout_marginTop="10dp"
35         android:layout_marginLeft="10dp"
```

```
33     android:textAppearance="@android:style/TextAppearance.Holo.
34             Medium" />
35
36     <TextView
37         android:layout_width="wrap_content"
38         android:layout_height="wrap_content"
39         android:text=""
40         android:id="@+id/type"
41         android:layout_below="@+id/label"
42         android:layout_toRightOf="@+id/idle"
43         android:layout_marginTop="10dp"
44         android:layout_marginLeft="10dp"
45         android:textAppearance="@android:style/TextAppearance.Holo.
46             Medium" />
47
48     <ImageButton
49         android:layout_width="wrap_content"
50         android:layout_height="wrap_content"
51         android:layout_margin="30dp"
52         android:src="@drawable/ic_save_black_24dp"
53         android:id="@+id/save_button"
54         android:layout_below="@+id/type"
55         android:layout_toRightOf="@+id/idle"/>
56
57     <ImageButton
58         android:layout_width="wrap_content"
59         android:layout_height="wrap_content"
60         android:src="@drawable/ic_pageview_black_24dp"
61         android:id="@+id/view_button"
62         android:layout_alignTop="@+id/save_button"
63         android:layout_toRightOf="@+id/save_button"
64         android:layout_toEndOf="@+id/save_button" />
65
66     <TextView
67         android:layout_width="wrap_content"
68         android:layout_height="wrap_content"
69         android:id="@+id/perm"
70         android:textStyle="bold"
71         android:text=""
72         android:layout_margin="8dp"
73         android:paddingTop="10dp"
74         android:paddingLeft="4dp"
75         android:layout_alignParentLeft="true"
76         android:layout_below="@+id/idle"
77         android:textColor="#ff000000" />
78
79     <ImageView
80         android:layout_width="wrap_content"
81         android:layout_height="wrap_content"
82         android:id="@+id/visibility"
83         android:layout_margin="10dp"
84         android:layout_alignParentLeft="true"
85         android:layout_below="@+id/perm"/>
86
87 </RelativeLayout>
```

Una vez creada la clase y las dos vistas en archivos XML, debemos añadir un objeto de la nueva clase manejadora, a la lista con el resto de objetos para manejar propiedades. Según la prioridad establecida, el nuevo maneja-

dor de propiedades se colocarán en una posición concreta. En el segmento de código B.6 podemos ver esta lista. La función que se observa es llamada al inicio del programa para establecer todas las clases que pueden manejar propiedades. Para cada objeto, llamamos al método de la clase Config `addUiPropertyManager(nuevo manejador)`

Fragmento de código 6.3: Lista de objetos manejadores de propiedades

```
1 private void setUiProperties() {  
2     //add UI object  
3     Config.init();  
4     Config.addUiPropertyManager(new UIBlobPropertyManager());  
5     Config.addUiPropertyManager(new UITextPropertyManager());  
6     Config.addUiPropertyManager(new UISwitchPropertyManager());  
7     Config.addUiPropertyManager(new UINumberPropertyManager());  
8     Config.addUiPropertyManager(new UILightPropertyManager());  
9     Config.addUiPropertyManager(new UIConnecPropertyManager());  
10    Config.addUiPropertyManager(new UIAbortPropertyManager());  
11    Config.addUiPropertyManager(new UIAbortPropertyManager());  
12 }  
13 }
```

Para ilustrar la posibilidad de añadir nuevas vistas, se han creado dos manejadores nuevos: `connection` y `abort` (figura B.1).

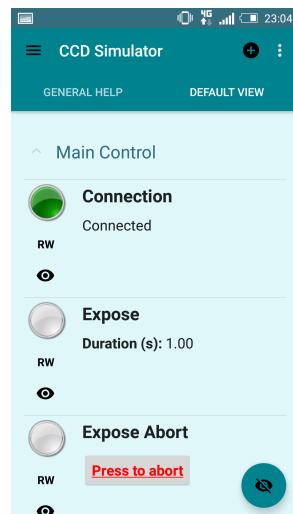


Figura 6.23: Vista de las propiedades connection y abort

6.3.2. Dispositivos

Al igual que con las propiedades, el software está diseñado para facilitar el diseño de nuevas clases para tener vistas especiales para algún dispositivo.

En el caso de las propiedades, se establecían prioridades para poder elegir de entre todas las posibilidades, cual iba a ser la interfaz de la propiedad concreta. En el caso de los dispositivos tenemos una interfaz de usuario por defecto, que lista las propiedades en una lista expandible. Pero se pueden añadir tantas como se deseen, pudiendo pasar de una a otra ya que se irán añadiendo al panel tabulado de la interfaz de usuario principal de la aplicación.

Para poder crear nuevas vistas para un dispositivo se ha diseñado la clase abstracta `DeviceView`.

Fragmento de código 6.4: Clase abstracta DeviceView

```

1 public abstract class DeviceView extends Fragment {
2
3     static Device device;
4     protected int layout;
5
6     /**
7      * Check if this class can represent p
8      *
9      * @param dev Indi property
10     * @return True/false if class can represent p
11     */
12    public abstract boolean handlesDevice (Device dev);
13
14    public abstract String getName();
15
16    @Override
17    public abstract View onCreateView(LayoutInflater inflater,
18                                     ViewGroup container, Bundle savedInstanceState);
19 }
```

Para poder crear una nueva vista de dispositivo debemos:

- Crear una clase que herede de `DeviceView`.
- Crear un archivo *XML* con la definición de la interfaz de usuario.
- Añadir un objeto de la clase a la lista de clases manejadoras de dispositivos.

Como podemos ver en el fragmento de código B.7, tenemos dos atributos que son el dispositivo que vamos a manejar, y la referencia de **Android** al recurso donde se encuentra el archivo *XML* con la definición de la vista. Además debemos implementar las siguientes funciones:

- **handlesDevice**: Esta función recibe un dispositivo y debe comprobar si puede manejarlo o no devolviendo un valor booleano.
- **getName**: Esta función debe devolver el nombre que se desea que aparezca en la parte superior de la vista (vista tabulada).
- **onCreateView**: La clase abstracta hereda de **Fragment** y por ello esta función debe ser implementada, ya que se llamará cuando se llame a la clase para pintar la interfaz de usuario[16]. En ella deberemos crear la vista y realizar todas las inicializaciones que queramos.

Al igual que ocurría con las propiedades, debemos crear una vista definiéndola en un archivo *XML* con los elementos visuales de **Android** que necesitemos.

Por último debemos agregar un objeto de nuestra clase a la lista de manejadores de dispositivos, como se ve en el fragmento de código B.9 llamando al método de la clase **Config addDeviceView(nuevo manejador)**

Fragmento de código 6.5: Lista de objetos manejadores de dispositivos

```
1 private void setDeviceViews() {  
2     Config.addDeviceView(new DeviceMeteoView());  
3 }
```

6.4. Implementación

Para un proyecto de desarrollo de software, las fases de análisis y diseño son claves para realizar la implementación. Uno de los objetivos del un proyecto fin de grado es demostrar la adquisición de competencias para realizar un correcto análisis de requisitos y un diseño exhaustivo. Estas fases previas realizadas correctamente aseguran una implementación más rápida. Por ello, la fase de implementación ha consistido en llevar a cabo todos los diagramas y especificaciones descritas en el diseño.

El código de la aplicación esta publicado bajo una licencia de **Software Libre** y puede ser consultada en <https://github.com/torresj/indi-android-ui>.

Para la implementación de este software, se ha usado el entorno de desarrollo **Android Studio** [17] (Figura 6.24). Este entorno es el recomendado por Android para realizar aplicaciones móviles.

6.4. Implementación

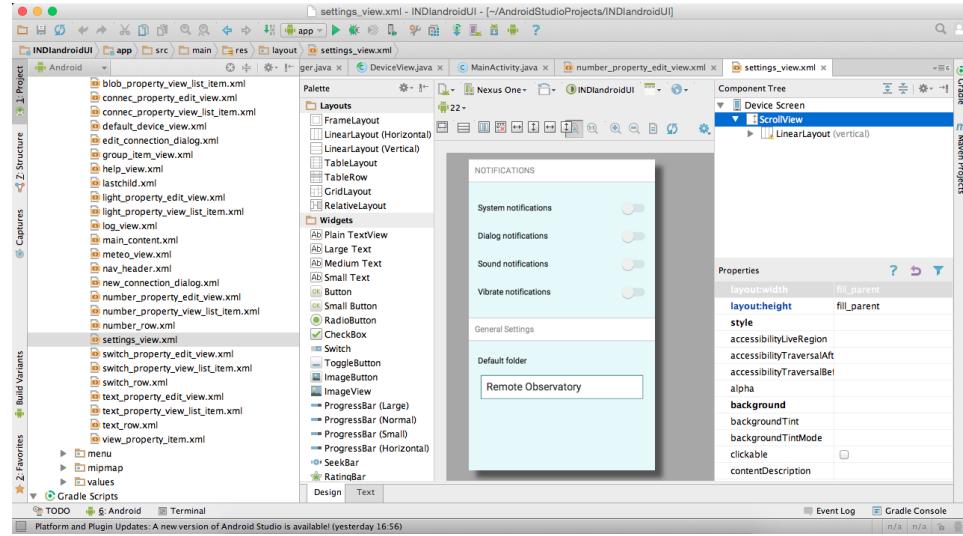


Figura 6.24: Android Studio

También se ha usado para el desarrollo el procesador de textos **Sublime text** [19].

Para el control de versiones y publicación del código se ha utilizado **github** y la herramienta en linea **git** para poder subir los cambios desde un terminal [18].

Capítulo 7

Pruebas

Una vez diseñada la aplicación y realizada la implementación, debemos asegurarnos de que todas las partes funcionan y cumplen los objetivos. Para ellos debemos iniciar la fase de pruebas, si bien es cierto que algunas pruebas se han realizada durante el proceso de implementación. Además, al utilizar una metodología de desarrollo basada en iteraciones, debemos tener en cuenta que cada una de las iteraciones ha tenido una fase final, previa a la entrega, de pruebas.

Podemos agrupar las pruebas de la aplicación en:

- **Pruebas unitarias.**
- **Pruebas de integración.**
- **Pruebas de sistema.**
- **Pruebas de aceptación.**

7.1. Pruebas Unitarias

Las pruebas unitarias son test realizados por el equipo de desarrollo y que se deben llevar a cabo subdividiendo las aplicaciones al nivel de clases o módulos fáciles de probar e independientes unos de otros. Estos test requieren cambiar el código para ser realizados, comprobando que cada módulo realiza la función esperada, que maneja los rangos de datos de entrada, su comportamiento al forzar los errores, etc.

Para las pruebas unitarias se ha dividido la aplicación en los siguientes módulos:

- **Conexión INDI.**

- **Lista de propiedades**
- **Manejadores de propiedades**
- **Creación de ficheros de configuración**

7.1.1. Conexión INDI

La conexión con un servidor INDI es vital para el desarrollo del resto de la aplicación. Por ello, en la primera iteración del proceso **Scrum** se marcó como requisito realizar una clase para implementar un cliente INDI. El objetivo era testear que podíamos conectarnos y obtener dispositivos y propiedades.

Para la consecución de la prueba se ha montado un servidor en una *Raspberry Pi* con varios simuladores INDI.

Objetivo de la prueba

El objetivo de la prueba es comprobar la correcta conexión a un servidor **INDI** desde una aplicación **Android** utilizando las bibliotecas de **INDI for Java**.

Descripción de la prueba

Se van a comprobar todos los posibles caminos de ejecución para asegurar que el módulo funciona correctamente:

- **Conexión a un servidor:** Nos conectamos a un servidor para comprobar el éxito del proceso.
- **Desconexión de un servidor:** Nos desconectamos de un servidor para comprobar el éxito del proceso.
- **Nos conectamos y apagamos el servidor:** Probamos la gestión de errores al caerse el servidor.
- **Conexión a un servidor con datos erróneos:** Comprobamos la conexión cuando los datos son erróneos o el servidor está desconectado o inaccesible.

Criterio de completitud

Las pruebas serán satisfactorias si el módulo gestiona correctamente todos los casos definidos anteriormente, y además se conecta de forma satisfactoria al servidor.

7.1.2. Listas de propiedades

Comprobamos que la clase que debe mostrar elementos funciona correctamente. Este módulo recibe como entrada un conjunto de propiedades y devuelve una lista para ser mostrada por pantalla.

Objetivo de la prueba

El objetivo de la prueba es comprobar que maneja correctamente un conjunto de datos de entrada para crear las listas de propiedades agrupadas por grupos

Descripción de la prueba

La prueba consiste en crear una lista de entrada con propiedades y comprobar que la salida es una lista de dos niveles con las propiedades agrupadas según su grupo. Para ellos se han realizado las siguientes comprobaciones:

- **Lista con distintos tipos de propiedades:** Creamos una lista de propiedades de distintos grupos y comprobamos la construcción de la lista de dos niveles.
- **Añadir una propiedad de un grupo nuevo:** Se añade una propiedad nueva perteneciente a un grupo aún sin crear.
- **Añadir una propiedad nueva con grupo ya existente:** Se añade una propiedad con un grupo que ya tiene más elementos.
- **Borramos una propiedad perteneciente a un grupo con más elementos:** Borramos una propiedad y comprobamos que se borra del grupo concreto.
- **Borramos una propiedad perteneciente a un grupo cuyo único elemento es la propia propiedad:** Borramos una propiedad de un grupo que quedará vacío al borrarla, por lo que debe desaparecer.

Criterio de completitud

Las pruebas serán satisfactorias si el módulo gestiona correctamente todos los casos definidos anteriormente, creando y borrando convenientemente las propiedades y grupos

7.1.3. Manejadores de propiedades

Los manejadores de propiedades deben, además de crear interfaces de usuario, gestionar la edición de cada una de las propiedades, manejando los errores derivados de entradas de datos erróneos.

Objetivo de la prueba

El objetivo de la prueba es comprobar que las clases de cada una de las propiedades, son capaces de procesar los datos de entrada y gestionar los errores derivados de ellos.

Descripción de la prueba

La prueba consiste en realizar una batería de test de entrada para cada una de las propiedades comprobando:

- **Datos fuera de rango (propiedad number).**
- **Datos en formato erróneo (propiedad number)**
- **Datos que no cumplen las condiciones de la propiedad (reglas en las propiedades Switch)**

Criterio de completitud

Las pruebas serán satisfactorias si el módulo gestiona correctamente todos los casos definidos anteriormente, informando de los distintos errores y gestionándolos correctamente.

7.1.4. Creación de ficheros de configuración

La aplicación debe ser capaz de crear una serie de ficheros de configuración para guardar información de forma permanente.

Objetivo de la prueba

Comprobar la creación correcta de los ficheros si no existen, o añadir los datos necesarios en caso de que ya existan

Descripción de la prueba

La prueba consiste iniciar el proceso de creación de ficheros de configuración, teniendo en cuenta los casos en que los ficheros existan o que no existan.

Hay que tener en cuenta que si existen, se debe añadir la información al final del archivo o sustituir el contenido según el caso.

Criterio de completitud

Las pruebas serán satisfactorias si el módulo gestiona correctamente todos los casos definidos anteriormente, creando y/o editando correctamente los ficheros de configuración.

7.2. Pruebas de integración

Las pruebas de integración nos permiten comprobar que las combinaciones entre módulos funcionan correctamente. Es habitual comprobar en este tipo de pruebas que los módulos de más bajo nivel se comunican correctamente con las interfaces de usuario verificando que las especificaciones de diseño sean alcanzadas.

Para este proyecto se han realizado las siguientes pruebas:

- **Conexión y desconexión con un servidor.**
- **Interacción con las propiedades.**
- **Gestión de las conexiones.**
- **Ajustes generales.**

7.2.1. Conexión con un servidor

En las pruebas unitarias hemos podido comprobar el módulo que realiza la conexión con el servidor. Ahora vamos a comprobar su integración en el resto de la aplicación.

Objetivo de la prueba

El objetivo es comprobar cómo el módulo de conexión con el servidor se integra con el resto de clases de la interfaz para mostrar los dispositivos y actualizar cualquier cambio en tiempo real que pueda sufrir cualquier dispositivo o propiedad

Descripción de la prueba

Para poder comprobar que la interacción entre el módulo que gestiona la conexión con el servidor y el resto de clases de la interfaz funciona correctamente, debemos realizar las siguientes comprobaciones:

- **Conexión y desconexión con un simulador.**
- **Conexión y desconexión con equipos reales.**

Montamos un servidor de pruebas con varios dispositivos simulados y realizamos conexiones y desconexiones comprobando que el servidor está encendido y apagado. Además, teniendo una conexión abierta, apagamos el servidor y/o desconectamos algún dispositivo para ver como el sistema gestiona la situación.

Por último realizamos las mismas comprobaciones pero utilizando un telescopio y una cámara.

Criterio de completitud

Las pruebas serán satisfactorias siempre y cuando el sistema nunca falle, gestionando cualquier situación de error y permitiendo la conexión y desconexión de servidores simultáneamente, mostrando los dispositivos y las propiedades.

7.2.2. Interacción con las propiedades

Una de las pruebas unitarias consistía en comprobar las clases que gestionan las propiedades. El siguiente paso es comprobar su integración con las distintas vistas para editarlas, su correcta comunicación y la gestión de los datos de entrada.

Objetivo de la prueba

El objetivo es comprobar que podemos cambiar las propiedades a partir de unos datos introducidos en las vistas de la interfaz de usuario.

Descripción de la prueba

Para cada propiedad, seleccionaremos la vista de edición, y cambiaremos los parámetros utilizando: datos correctos, dato mal construidos y datos fuera de rango. El sistema debe gestionar los errores para cada una de las propiedades:

- **Text:** Editaremos una propiedad *text* y comprobaremos que se realiza el cambio correctamente.
- **Number:** Editaremos los elementos de una propiedad *number* con valores correctos, con formato erróneo y fuera de rango.
- **Switch:** Editaremos los elementos de una propiedad *switch* con valores correctos y forzando situaciones que contravengan las reglas de la propiedad, como tener varias opciones activadas cuando solo se permite una, o desactivar todas las opciones cuando tiene que haber al menos una activa.
- **Blobs:** En los blobs necesitamos comprobar la recepción de archivos, que podemos guardarlo y mostrarlo.

Criterio de completitud

Las pruebas serán satisfactorias siempre y cuando el sistema nunca falle, gestionando cualquier situación de error y permitiendo editar la propiedad o informando de cualquier problema con los datos.

7.2.3. Gestión de las conexiones

Una de las configuraciones más importantes que se guardan en memoria son las conexiones, para tenerlas memorizadas aunque cerremos la aplicación. Para esta prueba debemos comprobar que el módulo de configuraciones guarda, borra, edita y crea correctamente las conexiones según las acciones llevadas a cabo por el usuario en la interfaz.

Objetivo de la prueba

El objetivo es comprobar que podemos crear, borrar y editar conexiones, y que estos cambios son guardados por la aplicación.

Descripción de la prueba

Para poder comprobar la gestión de las conexiones, primero realizamos las siguientes comprobaciones básicas:

- **Creamos una conexión.**
- **Editamos una conexión.**
- **Borramos una conexión.**

Además de realizar estas acciones, comprobaremos que al cerrar la aplicación y al abrirla se mantiene la integridad de los datos.

Criterio de completitud

Las pruebas serán satisfactorias siempre y cuando el sistema nunca falle, almacenando correctamente las conexiones y mostrándolas para su interacción con el usuario.

7.2.4. Ajustes generales

La aplicación cuenta con unas sencillas opciones de configuración para activar o desactivar los distintos tipos de notificaciones y la ruta de la carpeta por defecto en el sistema. Debemos comprobar que la activación de cada una de ellas provoca la consiguiente acción cuando se reciben notificaciones.

Objetivo de la prueba

El objetivo es doble. Por un lado, poder cambiar el estado de las opciones y que estén sean persistentes aunque cerremos la aplicación.

Por otro lado se debe comprobar que la activación de cada una de ellas permite la notificación concreta al recibir cualquier cambios en los dispositivos o en las conexiones.

Descripción de la prueba

Las opciones de configuración de las notificaciones son:

- **Notificaciones del sistema:** Activa la recepción de avisos cuando la aplicación está en segundo plano.
- **Notificaciones mediante diálogos emergentes:** Activa las notificaciones mediante un dialogo informando del acontecimiento ocurrido.
- **Notificaciones por sonido:** Activa la reproducción de un sonido al recibir una notificación.
- **Notificaciones por vibración:** Activa la vibración del dispositivo móvil al recibir una notificación.

La prueba consiste en activar cada una de ellas y conectarnos a un servidor de pruebas. Cada vez que se pierde la conexión con un servidor se envía una notificación por lo que aprovechamos esta situación para probar cada una de las distintas notificaciones. Además estas pruebas las realizamos utilizando dispositivos emulados y dispositivos reales.

Por último, comprobamos que el estado de las opciones es permanente aunque cerramos la aplicación, es decir, cada vez que modificamos cualquier opción, esta queda guardada en la configuración.

Criterio de completitud

Las pruebas serán satisfactorias siempre y cuando el sistema nunca falle al recibir una notificación, y que para cada una de las opciones de configuración, se produzca la acción adecuada.

7.3. Pruebas de sistema

Las pruebas de sistema son una serie de test que deben hacer un equipo ajeno al proyecto. En este caso las pruebas han sido llevadas a cabo por el Dr. Sergio Alonso Burgos, como director del proyecto y como cliente.

El objetivo de estas pruebas es comprobar los casos de uso, requerimientos y objetivos establecidos en la fase de diseño. Entre las pruebas que se han realizado destacamos las siguientes:

- **Funcionalidad.**
- **Usabilidad.**
- **Recuperación a errores.**

Las pruebas de sistema se han llevado a cabo al finalizar cada una de las iteraciones del proceso iterativo de desarrollo del software para comprobar la satisfacción de los objetivos y la necesidad de plantear una nueva lista de requerimientos y una nueva iteración.

7.4. Pruebas de aceptación

Una vez que la aplicación ha pasado todas las pruebas anteriores, consideramos superado un primer corte y vemos razonable designar a un grupo de usuarios ajenos al proyecto para realizar un uso normal del software y poder extraer conclusiones sobre su aceptación en un grupo reducido de personas.

Los usuarios potenciales de este tipo de aplicación son los aficionados a la astronomía, por lo que se decidió utilizar la página oficial de **INDI** para ofrecer la versión *Beta* de la aplicación. Los usuarios pueden probar el software y utilizar el foro para expresar sus opiniones y comentarios, ayudando al equipo de desarrollo a conocer el grado de aceptación y la necesidad de realizar cualquier tipo de cambio previo a su publicación final. A este tipo de usuarios se les conoce como *betatester*. En estos momentos hay varios usuarios probando la aplicación y mandando informes de errores.

Capítulo 8

Difusión del proyecto

Dada la practicidad del proyecto y que existe realmente una demanda del mismo (limitada pero existente) se han llevado a cabo diversas actividades para conseguir la máxima difusión del mismo:

- Página Web del proyecto.
- Publicación en la Google Play.
- Mantenimiento del repositorio en GitHub.
- Difusión en la página oficial de la biblioteca INDI.
- Difusión en el foro oficial de la biblioteca INDI.

En las siguientes secciones se comentan más en detalle cada una de dichas actividades.

8.1. Página web del proyecto

Se ha registrado el siguiente dominio: <http://remoteobservatory.info> así como contratado un hosting para poder crear una página web del proyecto (figura 8.1). La idea principal de dicha página es dar a conocer la aplicación de un modo visual y atractivo. Aunque probablemente el público objetivo de la misma no necesite grandes alicientes visuales para decidir probarla, se ha querido dar un diseño actual. Se han tenido en cuenta las siguientes consideraciones a la hora de crear dicho sitio web:

- Crear un sitio claro, directo y con la información más relevante sobre la aplicación: descripción del proyecto, ayuda (incluyendo el manual de usuario) y una sección de descargas.
- Crear un sitio visualmente atractivo y *responsive* para asegurarnos que se ve de manera adecuada tanto en ordenadores personales como en dispositivos móviles (figura 8.2).



Figura 8.1: Captura de la Página Web en Firefox (PC, Linux)

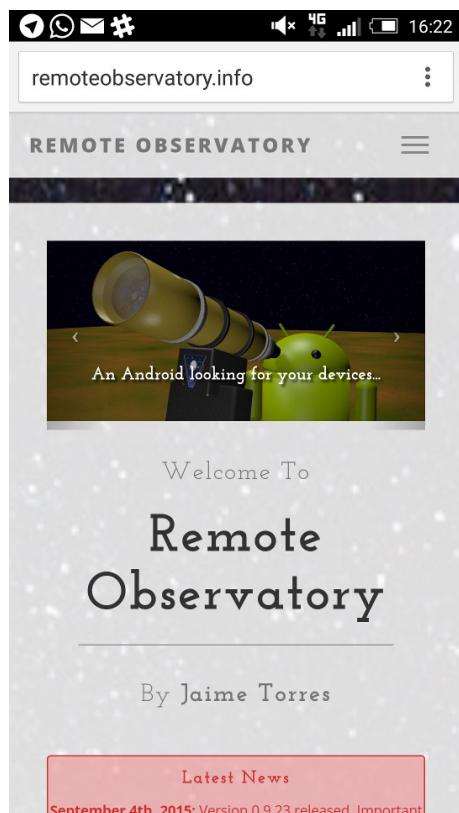


Figura 8.2: Captura de la Página Web en Teléfono Móvil

- Ilustrar el sitio con algunas imágenes que intenten transmitir la idea fundamental del proyecto: poder controlar dispositivos astronómicos de manera sencilla desde un móvil.
- Ofrecer claramente información de contacto con el desarrollador de la aplicación para que sea fácil establecer canales de comunicación entre los usuarios y el mismo (indispensable para depurar errores y obtener ideas sobre posibles nuevas funcionalidades).
- Dotar a la página de ciertos toques de humor geek que probablemente saquen alguna sonrisa a los aficionados a la astronomía.

Para conseguir estos objetivos se ha optado por usar una plantilla basada en Bootstrap (<http://getbootstrap.com/> ESTO SE PONE COMO BIBLIOGRAFÍA), uno de los frameworks HTML, CSS y JS más utilizados en la actualidad. Entre otras ventajas, permite desarrollar rápidamente un sitio web con un aspecto depurado en muy poco tiempo, especialmente si se utiliza una plantilla predefinida (que luego se modifica acorde al proyecto actual). En este caso, se utilizó una plantilla denominada

Bussiness Casual (<http://startbootstrap.com/template-overviews/business-casual/> OTRA BIBLIO).

8.2. Publicación en Google Play

La inmensa mayoría de los usuarios de Android instalan sus aplicaciones móviles utilizando la Google Play. Aunque otras vías de instalación son posibles (como la distribución del APK y su instalación manual), se ha creído conveniente añadir la aplicación a dicha tienda online (figura 8.3). Evidentemente la aplicación ha sido publicada sin ningún coste y totalmente accesible. Se ha incluido en la categoría **Herramientas**. Se ha procurado poner una descripción clara de la aplicación para que los posibles usuarios sepan si les puede ser útil o no.

Es interesante reseñar que en el poco tiempo que lleva publicada la aplicación algunos usuarios desconocidos (con los que no se ha contactado por ningún otro medio de manera explícita) ya se han instalado la aplicación. También es digno de mencionar que un usuario ha calificado muy mal la aplicación porque considera que “debería tener un telescopio público de prueba”, lo que pone de manifiesto que siempre podemos encontrar gente dispuesta a pedir absolutamente de todo.

8.3. Mantenimiento del repositorio en GitHub

Dado el carácter libre de la aplicación, todo el código de la misma está ubicada en un repositorio en GitHub (<https://github.com/BIBLIOGRAFIA>). La dirección de dicho repositorio es la siguiente: <https://github.com/torresj/indi-android-ui>. Como es habitual en este tipo de repositorios, se encuentran mecanismos que permiten inspeccionar el código, descargarlo, comparar cambios, contactar con el desarrollador, gestionar tickets, etc.

Además, a misma plataforma se puede ver como un mecanismo de difusión para que otros desarrolladores interesados en la temática astronómica puedan interesarse por la aplicación y el proyecto en general (lo que podría atraer otros desarrolladores).

8.4. Difusión en la página oficial de la biblioteca INDI

Dado que la gran mayoría de los potenciales usuarios de la aplicación conocerán el protocolo INDI y es probable que visiten su sitio web oficial (<http://indilib.org>), nos hemos puesto en contacto con sus encargados para que añadieran esta aplicación como uno de los posibles clientes INDI existentes. Al poco actualizaron la página web del proyecto incluyéndolo (figura

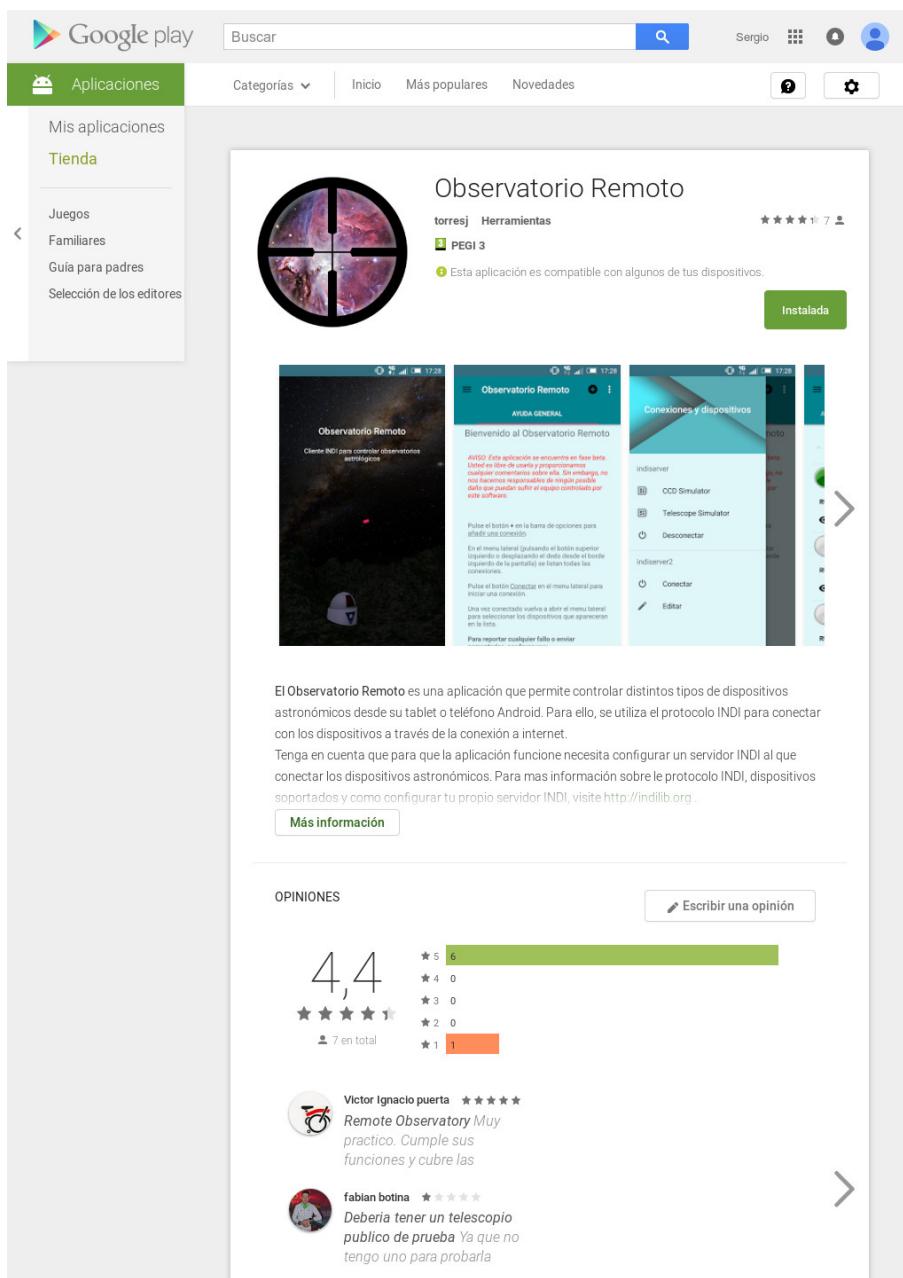


Figura 8.3: Captura de Remote Observatory en Google Play

8.4).

8.5. Difusión en el foro oficial de la biblioteca INDI

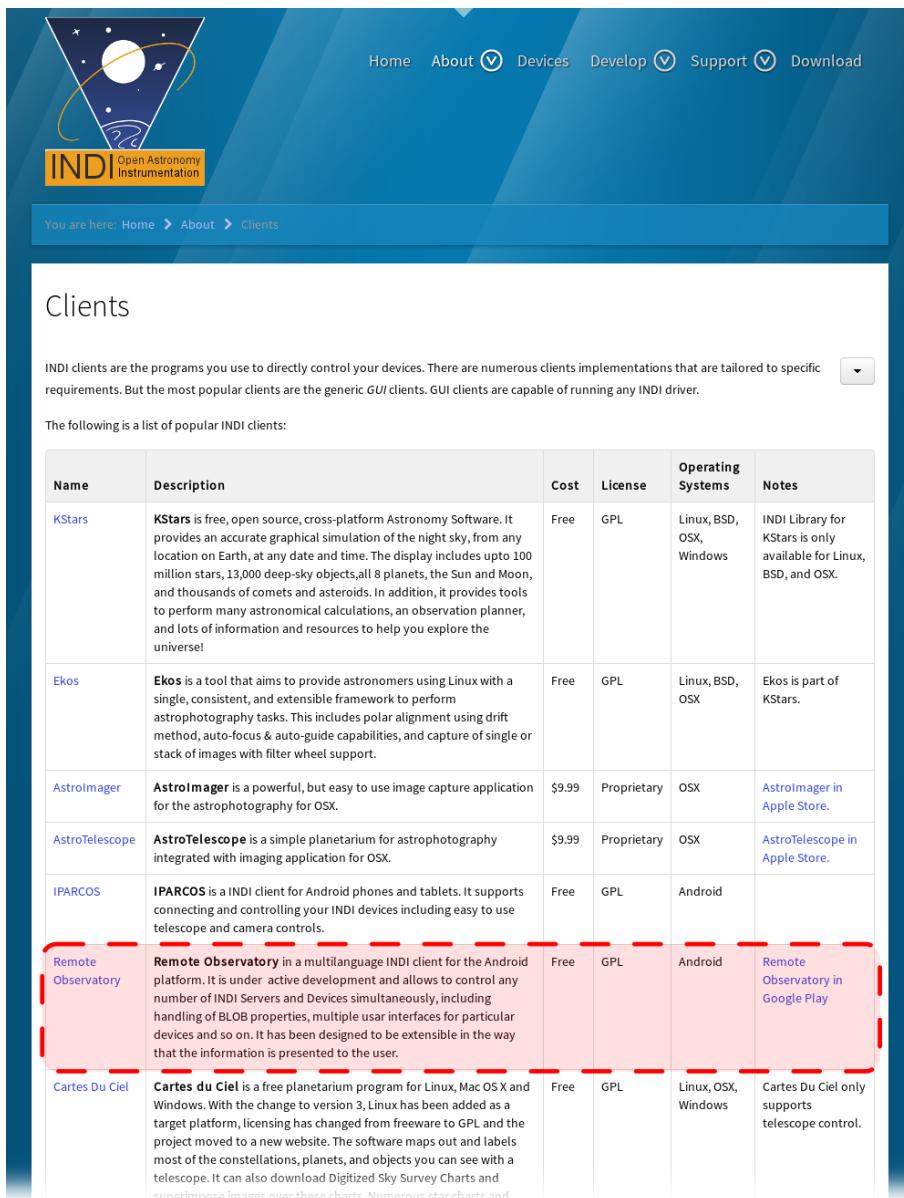
La página oficial de la biblioteca INDI tiene un foro donde se discuten temas relacionados: desde problemas con el uso de la biblioteca o alguna de sus herramientas relacionadas, se discuten temas de actualidad astronómica o detalles de implementación de la biblioteca, de los drivers, etc.

Se ha creado un hilo específico en el foro anunciando la aplicación (figura 8.5) lo que ha provocado que algunos usuarios se interesen y hayan empezado a mandar comentarios e incluso algún informe de errores.

8.6. Algunas ideas futuras para la difusión

Además de las actividades comentadas, se tienen planeadas algunas más para realizar en un futuro no muy lejano y aumentar la visibilidad del proyecto. Entre otras nos gustaría mencionar:

- Creación de un vídeo demostrativo de la aplicación y su uso con instrumental astronómico real. Además, este vídeo puede usarse para incrustarse tanto en la página web de la aplicación como en la página de Google Play.
- Poner a disposición del público un servidor INDI que implemente los simuladores de telescopio básicos para que los usuarios puedan hacer algunas pruebas fácilmente aunque no dispongan de hardware astronómico.



The screenshot shows the official INDI website's 'Clients' page. At the top, there is a navigation bar with links for Home, About, Devices, Develop, Support, and Download. Below the navigation bar, a breadcrumb trail indicates the current location: You are here: Home > About > Clients. The main content area is titled 'Clients'. It contains a brief introduction stating that INDI clients are programs used to control devices, mentioning KStars, Ekos, AstroImager, AstroTelescope, IPARCOS, Remote Observatory, and Cartes Du Ciel. A table lists these clients with their details: Name, Description, Cost, License, Operating Systems, and Notes. The 'Remote Observatory' row is highlighted with a red dashed border.

Name	Description	Cost	License	Operating Systems	Notes
KStars	KStars is free, open source, cross-platform Astronomy Software. It provides an accurate graphical simulation of the night sky, from any location on Earth, at any date and time. The display includes up to 100 million stars, 13,000 deep-sky objects, all 8 planets, the Sun and Moon, and thousands of comets and asteroids. In addition, it provides tools to perform many astronomical calculations, an observation planner, and lots of information and resources to help you explore the universe!	Free	GPL	Linux, BSD, OSX, Windows	INDI Library for KStars is only available for Linux, BSD, and OSX.
Ekos	Ekos is a tool that aims to provide astronomers using Linux with a single, consistent, and extensible framework to perform astrophotography tasks. This includes polar alignment using drift method, auto-focus & auto-guide capabilities, and capture of single or stack of images with filter wheel support.	Free	GPL	Linux, BSD, OSX	Ekos is part of KStars.
AstroImager	AstroImager is a powerful, but easy to use image capture application for the astrophotography for OSX.	\$9.99	Proprietary	OSX	AstroImager in Apple Store.
AstroTelescope	AstroTelescope is a simple planetarium for astrophotography integrated with imaging application for OSX.	\$9.99	Proprietary	OSX	AstroTelescope in Apple Store.
IPARCOS	IPARCOS is a INDI client for Android phones and tablets. It supports connecting and controlling your INDI devices including easy to use telescope and camera controls.	Free	GPL	Android	
Remote Observatory	Remote Observatory is a multilanguage INDI client for the Android platform. It is under active development and allows to control any number of INDI Servers and Devices simultaneously, including handling of BLOB properties, multiple user interfaces for particular devices and so on. It has been designed to be extensible in the way that the information is presented to the user.	Free	GPL	Android	Remote Observatory in Google Play
Cartes Du Ciel	Cartes du Ciel is a free planetarium program for Linux, Mac OS X and Windows. With the change to version 3, Linux has been added as a target platform, licensing has changed from freeware to GPL and the project moved to a new website. The software maps out and labels most of the constellations, planets, and objects you can see with a telescope. It can also download Digitized Sky Survey Charts and superimpose Images over these charts. Numerous star charts and	Free	GPL	Linux, OSX, Windows	Cartes Du Ciel only supports telescope control.

Figura 8.4: Captura de la página web oficial de la biblioteca INDI, donde se menciona al Observatorio Remoto <http://indilib.org/about/clients.html>

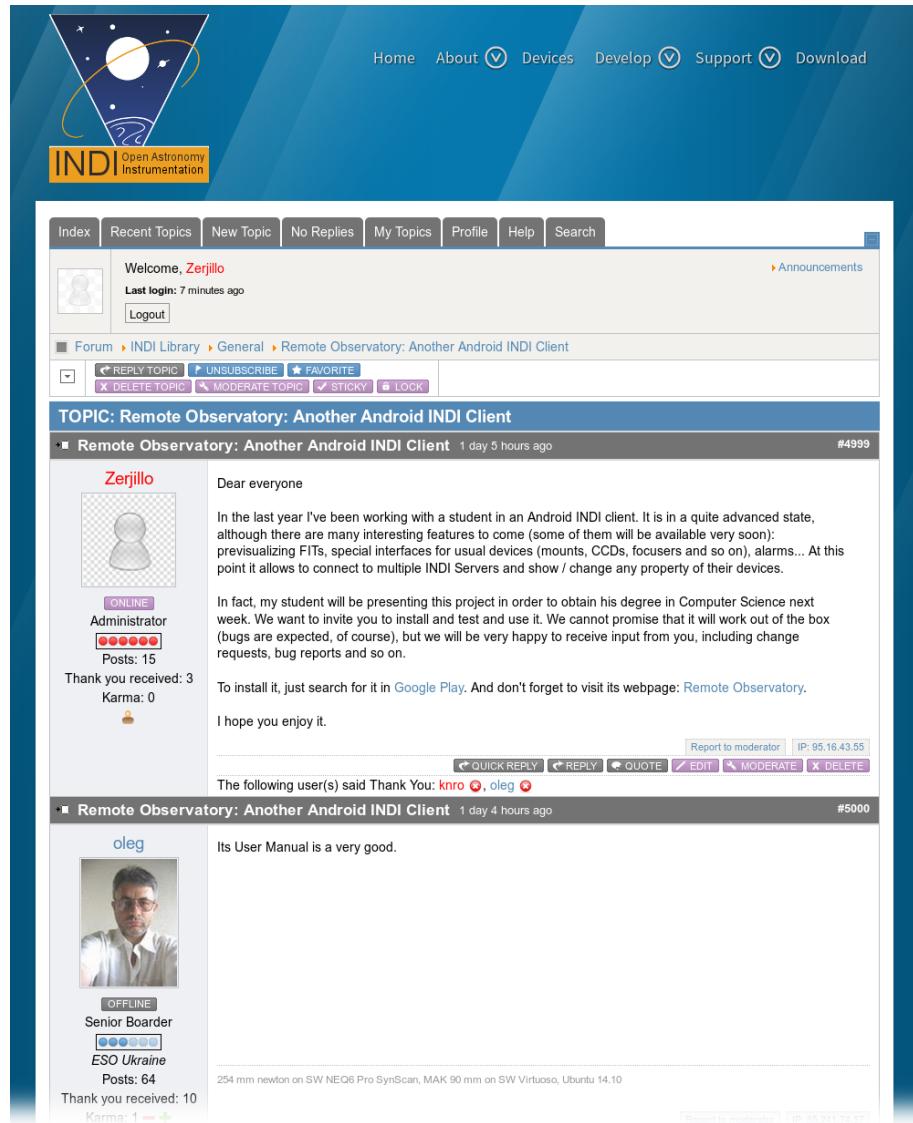


Figura 8.5: Captura del hilo del foro de la biblioteca INDI donde se presenta la aplicación <http://indilib.org/forum/general/823-remote-observatory-another-android-indi-client.html>

Capítulo 9

Conclusiones y trabajos futuros

Después de realizar este proyecto, he podido experimentar la importancia del orden en el desarrollo del software. En ocasiones los estudiantes podemos pecar de querer saltar directamente a la fase de implementación, sin darnos cuenta del gran error que conlleva esta decisión cuando hablamos de proyecto con una cierta envergadura.

Hay que tener en cuenta que debemos establecer unas fases claramente definidas y ser organizado en el trabajo. Intentar comprender lo que se quiere hacer extrayendo los objetivos y analizando correctamente lo que necesitamos. Si bien pienso que aún tengo que mejorar mucho al respecto, creo que el simple hecho de concienciarde de la importancia que tiene, es en sí una de las lecciones más importantes que se pueden sacar de la realización del proyecto.

Por otro lado, cabe destacar la importancia que tiene realizar un diseño correcto. La fase de implementación será más o menos llevadera y más o menos complicada en función de lo bueno o malo que sea el diseño previo. Aunque en todo momento se ha intentado ser riguroso en el diseño, malas decisiones o diseños ambiguos provocan tener que volver a rediseñar con el consiguiente tiempo perdido.

Si todas las fases previas a la implementación se realizan de forma rigurosa y correcta, la fase de implementación debería consistir simplemente en seguir unas instrucciones detalladas que no requieren pensar más allá del propio lenguaje de programación utilizado.

Por otro lado, tener que afrontar una planificación temporal, e intentar llevarla a cabo, también ha sido muy instructivo. Cuando se plantea un proyecto, se debe tener en cuenta una planificación inicial. En estos meses

he podido comprobar lo complicado que es ajustarse a ella, teniendo que modificarla y reajustarla de manera continua.

Por todas estas razones, creo que la principal conclusión del proyecto es asimilar la importancia y la necesidad de aplicar la ingeniería del software.

A nivel técnico, trabajar con tecnologías móviles y con equipos astronómicos ha sido una satisfacción personal ya que eran dos campos desconocidos para mí. Desarrollar mi primera aplicación en **Android** y publicarla da un valor añadido al propio valor académico del proyecto.

Las aplicaciones para dispositivos móviles son el presente y el futuro. La aplicación desarrollada cubre una necesidad real para un sector de usuarios concreto. Este hecho hace que el proyecto se considere vivo, y que se hayan planteado ya muchas líneas nuevas de desarrollo más haya del trabajo fin de grado. Por ello, se ha dado una especial atención a la difusión ya que nuestro objetivo es intentar que más usuarios usen la aplicación, y que la información que nos aporten retroalimente el proceso de mejora. Además el hecho de estar publicado con una licencia de **Software Libre** es una clara invitación a cualquier desarrollador para que colabore y ayude a mejorar el software. Personalmente ha sido para mí una motivación extra saber que más allá de la evaluación, se estaba realizando una aplicación real que continuará en el futuro y en la que estaré orgulloso de seguir formando parte.

En un futuro esperamos poder añadir funcionalidades extras a la aplicación tales como:

- Permitir crear alarmas para notificar cambios en propiedades (muy útil para controlar posibles emergencias en el observatorio).
- Añadir más vistas para propiedades estándar de INDI.
- Añadir más vistas a dispositivos conocidos, mejorando la interacción entre el usuario y las posibilidades del dispositivo concreto.
- Añadir una base de datos con la posición de estrellas y constelaciones para poder orientar directamente los dispositivos sin tener que introducir coordenadas.
- Crear un visor específico de archivos de tipo fits, para poder aplicarle algunos filtros y mejorar su visión en un dispositivo móvil
- Agregar más opciones a los ajustes generales para permitir decidir que tipos de alertas queremos en función de quien las envíe (conexión, dispositivo o propiedad)

Apéndice A

User Manual

A.1. Preface: A user manual for a mobile app? Are you kidding me?

Although many app developers and phone manufacturers think that there is no need to provide a user manual for a mobile app (they think users “enjoy” discovering hidden features) we feel it is better for the user to be able to understand how any software works. Therefore, you don’t have to look if the app can do something or not. You just check the manual. So, if you are looking for the way to do something and it is not in this manual... it probably cannot be done :-).

A.2. Starting the Application

Once installed you can launch the application from your apps menu in your phone. The loading screen (figure A.1) will appear showing the name of the program and its version number.

A.3. Main Screen Description

The main screen is divided in three main regions (image A.2):

- **Top Button Bar:** From where you can access to the Connection / Device selection , the Add Connection  and Main Menu  buttons. These will be described in more detail in the following.
- **Tabs Bar:** In this bar you will be able to select the different interfaces for each of the INDI devices that you are connected to.
- **Main Control:** The main portion of the app is dedicated to show the different properties of the INDI devices. At this point, this part of the screen just shows some general help about the app.

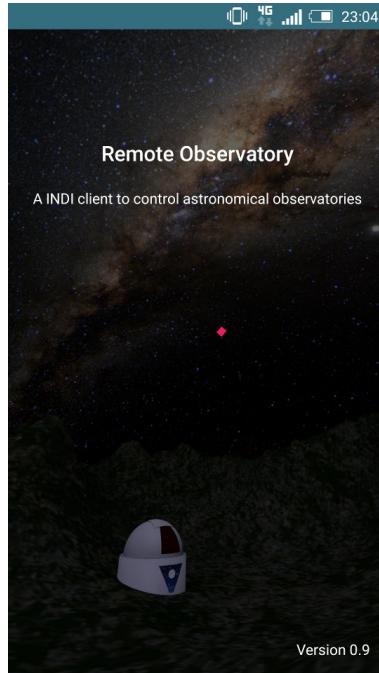


Figura A.1: Init Screen

A.4. Managing Connections

To control any INDI device you must connect to the server it is plugged in. To do so, **Remote Observatory** allows you to define one or more INDI servers (connections) to which you can connect. The first step to add a connection is to press the **Add Connection** button. A form will appear (figure A.3) in which several fields are required:

1. **Name:** A name for the connection (just to identify it) *Example: My Observatory.*
2. **Server:** The IP address or domain of your INDI Server. *Examples: 198.51.100.3 or indi.example.com.*
3. **Port:** The port of your INDI server. Typically 7624, but it depends on your server configuration. *Example: 7624.*
4. **Autoconnect:** Enabling it will make the app to connect to the server when starting.
5. **Enable BLOBs:** The app will ask the server to send BLOB data. This is for example important if you pretend to get images from a CCD. Please note that BLOBs often sends quite big amounts of data (not good if you have restrictions on your mobile data connection).

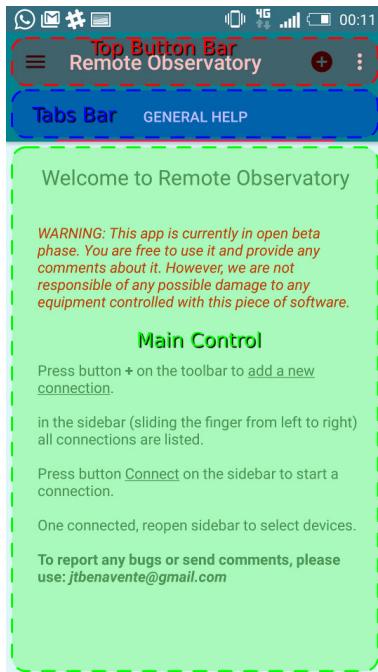


Figura A.2: Main Screen

Once you **Add** the connection it will appear in the **Connection / Device selection** panel (figure A.4), accesible from its button in the top button bar. From that panel you can **Edit** any of the fields of the connection or **Connect** to the server. Finally, if you want to delete a connection that will be no longer in use, you can use the **Remove Connections** option in the **Gereral Options** menu.

A.5. Adjusting Settings

There are several configuration options that control the general behaviour of the app. You can find those options under the **Settings** (figure A.5) option in the **Main Menu** (figure A.6):

- **Notifications:** The app can be configured to alert of some conditions (new devices found, connections that are lost, etc) using different mechanisms:
 - **System Notifications:** Using system notifications, usually visible in the upper menu in your android device.
 - **Dialog Notifications:** Using a dialog that shows the appropriate message.

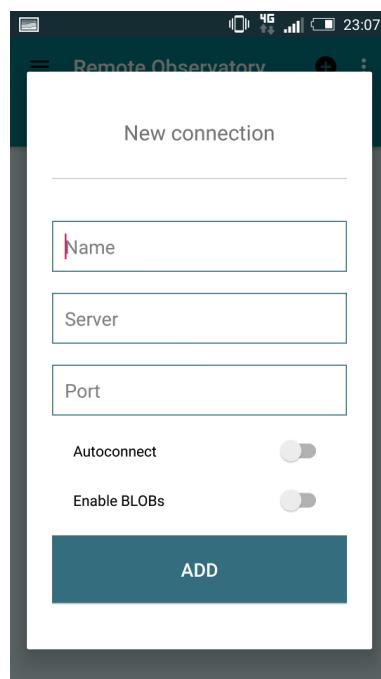


Figura A.3: New Connection

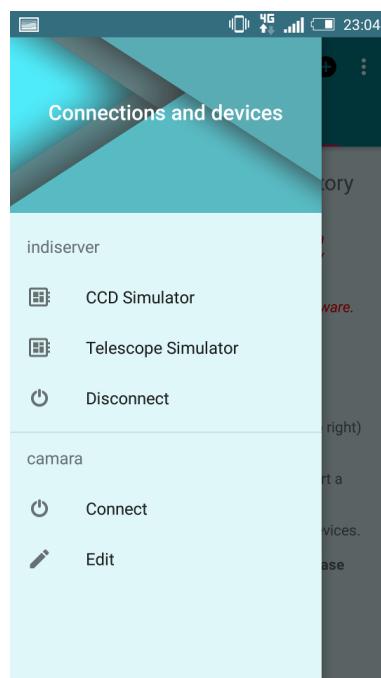


Figura A.4: Connection and Devices Panel

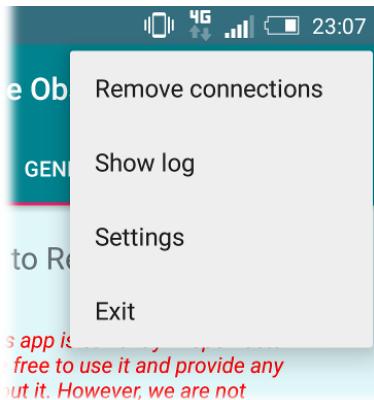


Figura A.5: Main Menu

- **Sound Notifications:** A sound will be played.
- **Vibrate Notifications:** The phone will vibrate.
- **General Settings:**
 - **Default Folder:** The default folder in which configuration files and BLOB data will be saved.

A.6. Logging

All the activity from / to de different devices are logged in text files which are accesible to the user. This logs may help the user to detect any problem or understand what has happened during an abbservation session. Each connection has an associated log file:

- **To view the logs** just use the **Show log** option in the **Main Menu** (figure A.7).
- **To Clear a file log** use the **Remove log** option in the **Edit panel** for the connection (figure A.8).

A.7. Connecting and Disconnecting from INDI Servers

Once you have set up at least one connection (section A.3), you can connect to it by pressing the **Connect** button in the **Connection / Device**

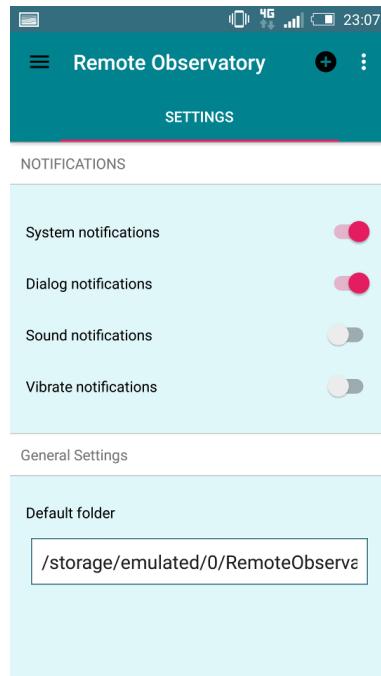


Figura A.6: Settings



Figura A.7: View Log

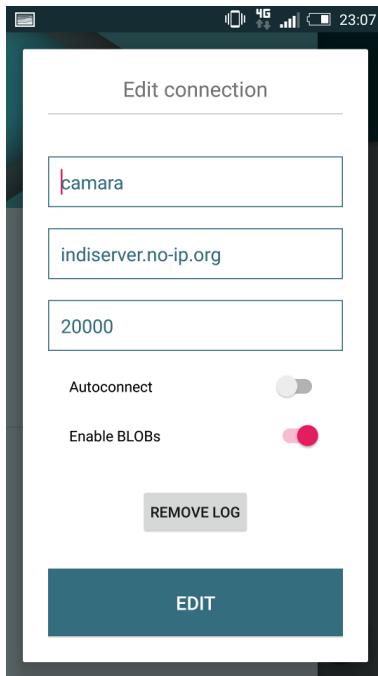


Figura A.8: Edit Connection

selection panel (figure A.4). Once the connection is established the app may receive information about all the devices in that INDI server.

To disconnect from a particular INDI server, just use the **Disconnect** button in the **Connection / Device selection** panel. Please note that if for any reason the connection is lost (the mobile phone loses its connection, the server is restarted, etc.), the app will notify us about this situation by means of any activated controls in the **Settings** (section A.3) (vibration, sound, system notification and/or dialog).

A.8. Device Views

Once a connection is established the server will usually send the available devices. To check the properties of any device, you can again click on the **Device Name** in the **Connection / Device selection** panel. This action will add at least a new tab to the **Tabs Bar** with the name of the device. Please take into account that to ease the management of different devices, more than one view for a particular device might be added to the **Tabs Bar**.

The **Default Device View** A.10) consists on a list of INDI Properties that are grouped according to their nature in different dropdown lists. Each property value, status and information is updated whenever the server sends any change about it. Each property has some common icons on its left side:

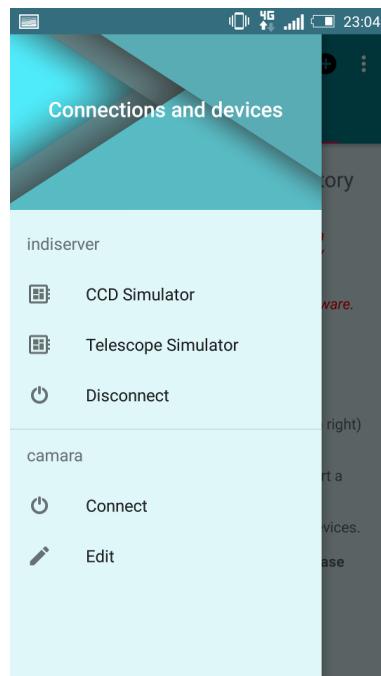


Figura A.9: Connect Button



Figura A.10: Default Device View

- **Property State Icon:** INDI properties must have one state. Each state is represented by a icon with the following colors:

- **Idle:** Gray  (the property has not yet been set).
- **Ok:** Green  (the value of the property is correct).
- **Busy:** Yellow  (the property is adjusting its value).
- **Alert:** Red  (there was some problem with the property value).

- **Property Permission:** INDI properties must show any of the following permissions:

- **Read / Write:** Denoted by a RW icon. The property values can be read and written.
- **Read Only:** Denoted by a RO icon. The property values can be read, but not written.
- **Write Only:** Denoted by a WO icon. The property values can be written, but not read.

- **Visibility Icon  **: Which allow to show / hide particular properties. Please check hidding / showing properties (section A.9).

On the right of those icons the app shows the current values for each one of the elements of the property. Depending on the nature of the property a slightly different interface is used to show that information:

- **Text properties:** It just shows the text value of each element of the property.

- **Number properties:** It shows the number value of each element of the property, formatted in its appropriate way (integer, float, sexagesimal...).

- **Light properties:** It shows an icon representing the light state:

- **Idle:** Gray .
- **Ok:** Green .
- **Busy:** Yellow .
- **Alert:** Red .

- **Switch properties:** It will show which of the options on the switch are ON or OFF.

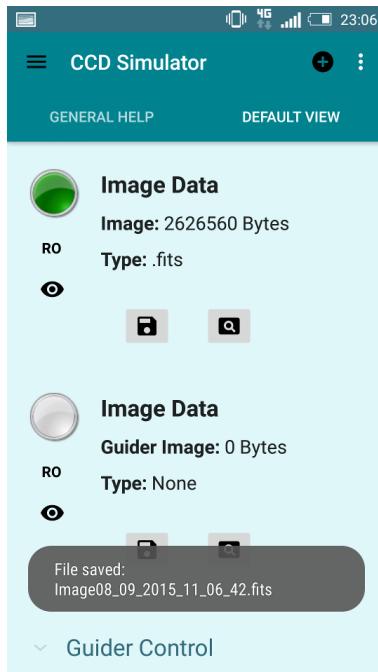


Figura A.11: Saving Blob Data

- **BLOB properties:** It will show the size of the BLOB packet (in bytes), its type (usually a file extension) and a pair of buttons:
 - **Save button** : It will save the BLOB data to the appropriate folder (see section A.4).
 - **Preview button** : It will launch another application on the phone that is able to manage the BLOB. Please note that if you want to visualize .fits images an appropriate app that handles that image type should be installed in your device.

A.9. Updating Properties

Every property that has a RW or WO permission can be updated with new values. To do so, you just click on the property to update. If a RO is clicked an alert will be shown (figure A.12). Depending on the nature of the property a different form will be shown to update the property values:

- **Text Property:** A form with the text values to change (figure A.13).
- **Number Property:** A form with the number values to change. The number **format** and **min** and **max** wil be shown and incorrect values will be filtered (figure A.14).

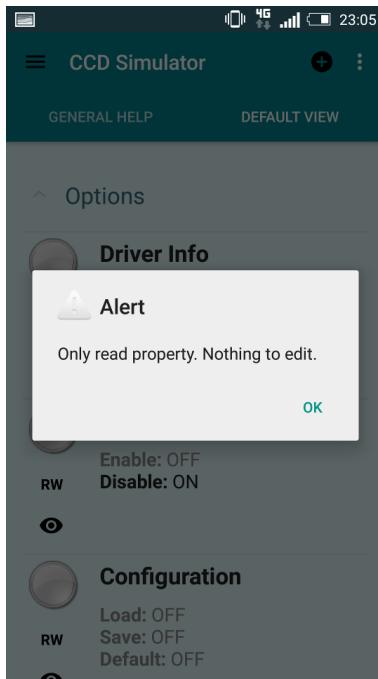


Figura A.12: Read Only Property Alert

- **Switch Property:** An array of the possible selection elements will be shown. The **Switch Rule** will prevent the user to turn on more elements that are allowed (figure A.15).
- **BLOB Property:** An interface that allows to pick files from your device to send its contents to the property.

A.10. Hidding / Showing Properties

As devices usually have a large amount of properties and some of them are not used very often by the user (configuration ones, for example) the **Default Device View** allows to hide them in order to simplify the interface.

To do so, we can toggle the **Visibility** button on the property.

However, to be able to show a hidden property, a **Global Visibility** toggle button (figure A.16) has been incorporated (visible on the lower right part of the app when scrolling). When that button is all properties will be shown. When it is only the properties with its **Visibility** button will be shown.

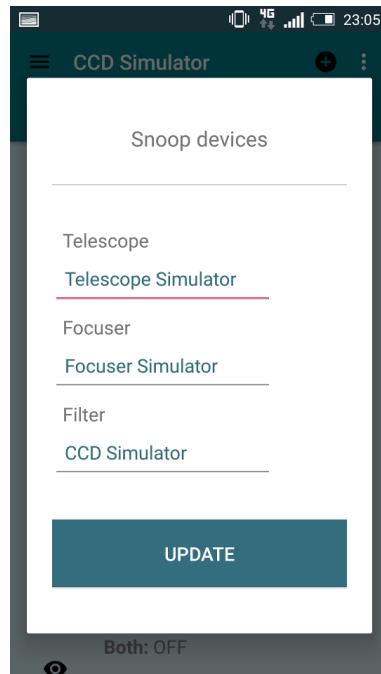


Figura A.13: Updating a Text Property

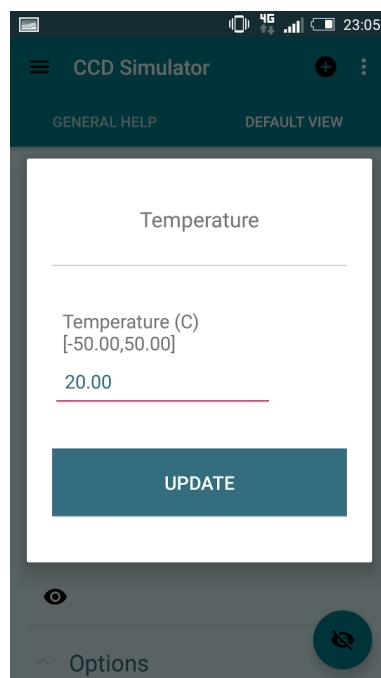


Figura A.14: Updating a Number Property

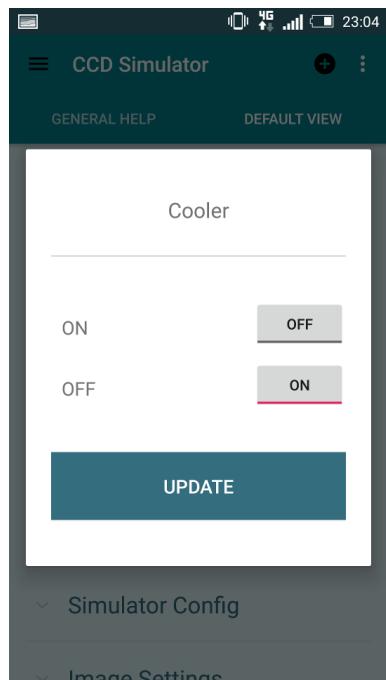


Figura A.15: Updating a Switch Property

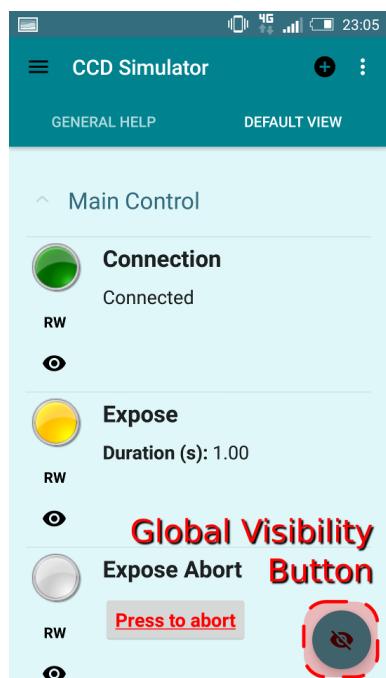


Figura A.16: Global Visibility Button

Apéndice B

Manual del desarrollador

B.1. Mecanismo de adición de vistas

La aplicación ha sido diseñada para permitir añadir nuevas vistas para propiedades y dispositivos sin la necesidad de conocer el código de la aplicación. Cualquier clase que se añada debe ir dentro del paquete principal con el resto de clases. Igualmente, todos los layaout de android (vistas) deben ir en la carpeta de `resources` y dentro de esta en `layouts`.

B.1.1. Propiedades

Para poder añadir una nueva vista debemos seguir tres pasos:

- Crear una clase que implemente la interfaz **UIPropertyManager**.
- Crear una interfaz de usuario para android en XML tanto para la vista de elemento de lista como para la vista para editar una propiedad.
- Declarar un objeto de la clase creada y añadirlo al vector de manejadores.

Interfaz UIPropertyManager

A continuación se muestra la interfaz de Java que debemos implementar en la nueva clase manejadora que deseamos crear.

Fragmento de código B.1: Interfaz UIPropertyManager

```
1 /**
2  * Interface to handle views and indi propertys
3  */
4 public interface UIPropertyManager {
5     /**
6      * Check if this class can represent p
7 }
```

```
8  *
9  * @param p Indi property
10 * @return True/false if class can represent p
11 */
12 boolean handlesProperty (INDIProperty p);
13
14 /**
15 * Create and return a view sets with p elements
16 *
17 * @param p Indi property
18 * @param inflater Layout inflater to inflate view
19 * @param parent ViewGroup to inflate view
20 * @param context Context to allow use Activity methods
21 * @return View
22 */
23 View getPropertyView (INDIProperty p, LayoutInflator inflater,
24     ViewGroup parent, Context context);
25
26 /**
27 * update view v with Indi property p elements
28 *
29 * @param p Indi property
30 * @param v View
31 */
32 void updateView (INDIProperty p, View v);
33
34 /**
35 * Create a new view dialog to allow set Indi property p elements
36 *
37 * @param p Indi property
38 * @param inflater Layout inflater to inflate view
39 * @return View
40 */
41 View getUpdateView(INDIProperty p, LayoutInflater inflater,
42     DialogFragment fragment);
43
44 /**
45 * Update property with change saved at view v
46 *
47 * @param p Indi property
48 * @param v View
49 */
50 void updateProperty(INDIProperty p, View v);
51
52 /**
53 * Get priority
54 *
55 * @return priority
56 */
57 int getPriority();
58
59 /**
60 * Get update button reference
61 *
62 * @return update button
63 */
64 Button getUpdateButton();
```

Como podemos ver en el fragmento de código B.1, las funciones que debemos implementar son:

- **handlesProperty**: Esta función recibe una propiedad y debe comprobar si puede manejarla o no devolviendo un valor booleano.
- **getPropertyView**: Una vez que comprobamos que podemos manejar la propiedad, esta función debe devolvernos la vista “inflada” que se mostrará en la lista de propiedades. Los parámetros inflater, parent y context son necesarios para poder inflar la vista.
- **updateView**: Esta función es la que debe coger la vista anteriormente creada, y debe rellenarla a partir de la propiedad p.
- **getUpdateView**: Esta función debe crear la vista para el modo de edición de la propiedad. Esta vista se pintará en un diálogo de tipo **Fragment**. Al igual que antes, necesitamos las variables inflater y fragment para “inflar” la vista.
- **updateProperty**: La vista para editar propiedades permite al usuario editar una propiedad. Una vez que el usuario ha terminado la edición, debemos actualizar la propiedad con los valores de la vista y enviar la información al servidor usando la función de la propiedad de **INDI**, **sendChangesToDriver**.
- **getPriority**: Esta función debe devolver una valor numérico para establecer la prioridad que tendrá la vista respecto a otras vistas o a la vista por defecto de la propiedad. Cuanto mayor sea el valor, más prioridad tendrá
- **getUpdateButton**: Esta función debe devolver un objeto “Button” si la vista tendrá el botón para actualizar, o *null* en caso de que no haya botón.

Para ilustrar la creación de la clase y la implementación de la interfaz, en el segmento de código B.2 podemos ver la implementación de la clase **UIConnecPropertyManager**. Este tipo de propiedad es una *Switch* al que se va a cambiar la vista por defecto. Vemos que es muy importante implementar la función **handlesProperty** ya que en ella debemos comprobar si la propiedad que nos mandan es la que podemos manejar.

Otro ejemplo de clase manejadora es la del segmento de código B.3. En este caso buscamos cualquier propiedad de tipo *switch* que tenga un solo elemento y que su nombre sea “Abort”.

Fragmento de código B.2: Clase manejadora de la propiedad connection

```

1 public class UIConnecPropertyManager implements UIPROPERTYMANAGER, View
2     .ONCLICKLISTENER {
3     //Atributes
4     int layout;
5     int layout_dialog;
6     Button button;
7
8     public UIConnecPropertyManager(){
9         layout=R.layout.connec_property_view_list_item;
10        layout_dialog=R.layout.connec_property_edit_view;
11    }
12
13    @Override
14    public boolean handlesProperty(INDIProperty p) {
15
16        if(p.getName().equals("CONNECTION"))
17            return true;
18        else
19            return false;
20    }
21
22    @Override
23    public View getPropertyView(INDIProperty p, LayoutInflator
24        inflater, ViewGroup parent, Context context) {
25        View v=inflater.inflate(layout, parent, false);
26        return v;
27    }
28
29    @Override
30    public void updateView(INDIProperty p, View v) {
31        setView(v, p);
32    }
33
34    @Override
35    public View getUpdateView(INDIProperty p, LayoutInflator inflater,
36        DialogFragment fragment) {
37        View v = inflater.inflate(layout_dialog,null);
38        TextView name=(TextView)v.findViewById(R.id.property_name);
39        Switch s=(Switch)v.findViewById(R.id.conn_switch);
40        button=(Button)v.findViewById(R.id.update_button);
41
42        ArrayList<INDIELEMENT> list =(ArrayList) p.getElementsAsList()
43        ;
44        INDISwitchElement elem=(INDISwitchElement)list.get(0);
45
46        if (elem.getValue().equals(Constants.SwitchStatus.ON))
47            s.setChecked(true);
48        else
49            s.setChecked(false);
50
51        name.setText(p.getLabel());
52        return v;
53    }
54
55    @Override
56    public void updateProperty(INDIProperty p, View v) {
57        Switch s=(Switch)v.findViewById(R.id.conn_switch);

```

```
57     ArrayList<INDIElement> list =(ArrayList) p.getElementsAsList()
58     ;
59     INDISwitchElement conect=(INDISwitchElement)list.get(0);
60     INDISwitchElement disconnect=(INDISwitchElement)list.get(1);
61
62     try {
63         if(s.isChecked()){
64             disconnect.setDesiredValue(Constants.SwitchStatus.OFF);
65             conect.setDesiredValue(Constants.SwitchStatus.ON);
66         }else{
67             disconnect.setDesiredValue(Constants.SwitchStatus.ON);
68             conect.setDesiredValue(Constants.SwitchStatus.OFF);
69         }
70
71         p.sendChangesToDriver();
72     } catch (INDIValueException e) {
73         e.printStackTrace();
74     } catch (IOException e) {
75         e.printStackTrace();
76     }
77 }
78
79 @Override
80 public int getPriority() {
81     return 5;
82 }
83
84 @Override
85 public Button getUpdateButton() {
86     return button;
87 }
88
89 void setView(View v, INDIProperty p){
90     //Views
91     TextView name = (TextView)v.findViewById(R.id.name);
92     ImageView idle = (ImageView)v.findViewById(R.id.idle);
93     TextView perm = (TextView)v.findViewById(R.id.perm);
94     ImageView visibility = (ImageView)v.findViewById(R.id.
95         visibility);
96     TextView element = (TextView)v.findViewById(R.id.element);
97
98     visibility.setTag(p);
99     visibility.setFocusable(false);
100    visibility.setOnClickListener(this);
101
102    //others
103    int light_res=0;
104    String perm_res="";
105    int visibility_res=0;
106
107    ArrayList<INDIElement> list =(ArrayList) p.getElementsAsList()
108        ;
109
110    String text="";
111
112    INDISwitchElement elem=(INDISwitchElement)list.get(0);
113
114    if (elem.getValue().equals(Constants.SwitchStatus.ON))
115        text="Connected";
116    else
117        text="Disconnected";
```

```

116     element.setText(text);
117
118
119
120     //State
121     if(p.getState().name().equals("IDLE")){
122         light_res=R.drawable.grey_light_48;
123     }else if(p.getState().name().equals("OK")){
124         light_res=R.drawable.green_light_48;
125     }else if(p.getState().name().equals("BUSY")){
126         light_res=R.drawable.yellow_light_48;
127     }else{
128         light_res=R.drawable.red_light_48;
129     }
130
131     //Permission
132     if(p.getPermission().equals(Constants.PropertyPermissions.R0))
133     {
134         perm_res="R0";
135     }else if(p.getPermission().equals(Constants.
136         PropertyPermissions.W0)){
137         perm_res="W0";
138     }else{
139         perm_res="RW";
140     }
141
142     if(DefaultDeviceView.conn.isPropertyHide(p))
143         visibility_res=R.drawable.ic_visibility_off_black_24dp;
144     else
145         visibility_res=R.drawable.ic_visibility_black_24dp;
146
147     name.setText(p.getLabel());
148     idle.setImageResource(light_res);
149     perm.setText(perm_res);
150     visibility.setImageResource(visibility_res);
151 }
152
153     @Override
154     public void onClick(View v) {
155         INDIProperty p=(INDIProperty)v.getTag();
156         Connection conn=DefaultDeviceView.conn;
157         if(conn.isPropertyHide(p)){
158             conn.showProperty(p);
159         }else{
160             conn.hideProperty(p);
161         }
162     }

```

Fragmento de código B.3: Clase manejadora de propiedades Abort

```

1 public abstract class DeviceView extends Fragment {
2
3     public class UIAbortPropertyManager implements UIPROPERTYManager, View.
4         OnClickListerner {
5
6         //Atributes
7         int layout;
    int layout_dialog;

```

```
8     Button button;
9     Context context;
10
11    public UIAbortPropertyManager(){
12        layout=R.layout.abort_property_view_list_item;
13        layout_dialog=R.layout.abort_property_edit_view;
14    }
15
16    @Override
17    public boolean handlesProperty(INDIProperty p) {
18        if(p instanceof INDISwitchProperty){
19            ArrayList<INDIElement> list =(ArrayList) p.
20                getElementsAsList();
21            if(list.size()==1) {
22                INDISwitchElement elem = (INDISwitchElement) list.get
23                    (0);
24                if(elem.getLabel().equals("Abort")){
25                    return true;
26                }
27                else{
28                    return false;
29                }
30            }
31            else{
32                return false;
33            }
34        }
35
36        @Override
37        public View getPropertyView(INDIProperty p, LayoutInflater
38            inflater, ViewGroup parent, Context context) {
39            this.context=context;
40            View v=inflater.inflate(layout, parent, false);
41            return v;
42        }
43
44        @Override
45        public void updateView(INDIProperty p, View v) {
46            setView(v, p);
47        }
48
49        @Override
50        public View getUpdateView(INDIProperty p, LayoutInflater inflater,
51            DialogFragment fragment) {
52            View v = inflater.inflate(layout_dialog,null);
53            TextView name=(TextView)v.findViewById(R.id.property_name);
54            button=(Button)v.findViewById(R.id.abort_button);
55            name.setText(p.getLabel());
56            return v;
57        }
58
59        @Override
60        public void updateProperty(INDIProperty p, View v) {
61            ArrayList<INDIElement> list =(ArrayList) p.getElementsAsList()
62            ;
63            INDISwitchElement abort=(INDISwitchElement)list.get(0);
64
65            try {
66                abort.setDesiredValue(Constants.SwitchStatus.ON);
67                p.sendChangesToDriver();
68            }
```

```

65     } catch (INDIValueException e) {
66         e.printStackTrace();
67     } catch (IOException e) {
68         e.printStackTrace();
69     }
70 }
71
72 @Override
73 public int getPriority() {
74     return 9;
75 }
76
77 @Override
78 public Button getUpdateButton() {
79     return button;
80 }
81
82 void setView(View v, INDIProperty p){
83     //Views
84     TextView name = (TextView)v.findViewById(R.id.name);
85     ImageView idle = (ImageView)v.findViewById(R.id.idle);
86     TextView perm = (TextView)v.findViewById(R.id.perm);
87     ImageView visibility = (ImageView)v.findViewById(R.id.
88         visibility);
89     TextView element = (TextView)v.findViewById(R.id.element);
90
91     visibility.setTag(p);
92     visibility.setFocusable(false);
93     visibility.setOnClickListener(this);
94
95     //others
96     int light_res=0;
97     String perm_res="";
98     int visibility_res=0;
99
100    String text=context.getResources().getString(R.string.
101        abort_text);
102
103
104    //State
105    if(p.getState().name().equals("IDLE")){
106        light_res=R.drawable.grey_light_48;
107    }else if(p.getState().name().equals("OK")){
108        light_res=R.drawable.green_light_48;
109    }else if(p.getState().name().equals("BUSY")){
110        light_res=R.drawable.yellow_light_48;
111    }else{
112        light_res=R.drawable.red_light_48;
113    }
114
115    //Permission
116    if(p.getPermission().equals(Constants.PropertyPermissions.R0))
117    {
118        perm_res="R0";
119    }else if(p.getPermission().equals(Constants.
120        PropertyPermissions.W0)){
121        perm_res="W0";
122    }else{
123        perm_res="RW";
124    }

```

```
122     }
123
124     if(DefaultDeviceView.conn.isPropertyHide(p))
125         visibility_res=R.drawable.ic_visibility_off_black_24dp;
126     else
127         visibility_res=R.drawable.ic_visibility_black_24dp;
128
129     name.setText(p.getLabel());
130     idle.setImageResource(light_res);
131     perm.setText(perm_res);
132     visibility.setImageResource(visibility_res);
133 }
134
135
136     @Override
137     public void onClick(View v) {
138         INDIProperty p=(INDIProperty)v.getTag();
139         Connection conn=DefaultDeviceView.conn;
140         if(conn.isPropertyHide(p)){
141             conn.showProperty(p);
142         }else{
143             conn.hideProperty(p);
144         }
145     }
146 }
```

Una vez que tenemos la clase creada debemos diseñar la interfaz de usuario en un archivo *XML* como el del archivo del fragmento de código B.4 que representa la vista de elemento de lista de la propiedad *Connection*. Además de este, debemos crear otra vista para mostrar en el diálogo de edición de propiedades (fragmento de código B.5). No hay ningún tipo de restricción para estas vistas. El desarrollador es libre de poner lo que necesite en ambas interfaces de usuario.

Fragmento de código B.4: Vista item XML de una propiedad

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    android"
    android:layout_width="match_parent" android:layout_height="
        wrap_content"
    android:padding="5dp">

    <ImageView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/idle"
        android:layout_marginRight="10dp"
        android:layout_alignParentLeft="true" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Text"
        android:id="@+id/name"
        android:layout_alignParentTop="true"
```

```
20      android:layout_toRightOf="@+id/idle"
21      android:layout_marginLeft="10dp"
22      android:textAppearance="@android:style/TextAppearance.Holo.
23          Large"
24      android:textStyle="bold" />
25  <TextView
26      android:layout_width="wrap_content"
27      android:layout_height="wrap_content"
28      android:text=""
29      android:id="@+id/label"
30      android:layout_below="@+id/name"
31      android:layout_toRightOf="@+id/idle"
32      android:layout_marginTop="10dp"
33      android:layout_marginLeft="10dp"
34      android:textAppearance="@android:style/TextAppearance.Holo.
35          Medium" />
36
37  <TextView
38      android:layout_width="wrap_content"
39      android:layout_height="wrap_content"
40      android:text=""
41      android:id="@+id/type"
42      android:layout_below="@+id/label"
43      android:layout_toRightOf="@+id/idle"
44      android:layout_marginTop="10dp"
45      android:layout_marginLeft="10dp"
46      android:textAppearance="@android:style/TextAppearance.Holo.
47          Medium" />
48
49  <ImageButton
50      android:layout_width="wrap_content"
51      android:layout_height="wrap_content"
52      android:layout_margin="30dp"
53      android:src="@drawable/ic_save_black_24dp"
54      android:id="@+id/save_button"
55      android:layout_below="@+id/type"
56      android:layout_toRightOf="@+id/idle"/>
57
58  <ImageButton
59      android:layout_width="wrap_content"
60      android:layout_height="wrap_content"
61      android:src="@drawable/ic_pageview_black_24dp"
62      android:id="@+id/view_button"
63      android:layout_alignTop="@+id/save_button"
64      android:layout_toRightOf="@+id/save_button"
65      android:layout_toEndOf="@+id/save_button" />
66
67  <TextView
68      android:layout_width="wrap_content"
69      android:layout_height="wrap_content"
70      android:id="@+id/perm"
71      android:textStyle="bold"
72      android:text=""
73      android:layout_margin="8dp"
74      android:paddingTop="10dp"
75      android:paddingLeft="4dp"
76      android:layout_alignParentLeft="true"
77      android:layout_below="@+id/idle"
78      android:textColor="#ff000000" />
79
80  <ImageView
81      android:layout_width="wrap_content"
```

```
79     android:layout_height="wrap_content"
80     android:id="@+id/visibility"
81     android:layout_margin="10dp"
82     android:layout_alignParentLeft="true"
83     android:layout_below="@+id/perm"/>
84
85 </RelativeLayout>
```

Fragmento de código B.5: Vista de edición XML de una propiedad

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
3      android:id="@+id/ScrollView01"
4      android:layout_width="fill_parent"
5      android:layout_height="fill_parent">
6      <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
7          android"
8              android:layout_width="match_parent"
9              android:layout_height="match_parent"
10             android:background="@android:color/white"
11             android:padding="24dp">
12
13
14     <TextView
15         android:id="@+id/info_text"
16         android:layout_width="wrap_content"
17         android:layout_height="wrap_content"
18         android:layout_alignParentTop="true"
19         android:layout_centerHorizontal="true"
20         android:gravity="center"
21         android:text="@string/action_edit"
22         android:textSize="20dp" />
23
24
25     <View
26         android:id="@+id/divider"
27         android:layout_width="match_parent"
28         android:layout_height="1dp"
29         android:layout_below="@+id/info_text"
30         android:layout_marginBottom="20dp"
31         android:layout_marginTop="10dp"
32         android:background="#C8C9CB" />
33
34     <EditText
35         android:id="@+id/name"
36         android:layout_width="match_parent"
37         android:layout_height="wrap_content"
38         android:layout_below="@+id/divider"
39         android:layout_centerHorizontal="true"
40         android:layout_marginTop="20dp"
41         android:background="@drawable/text_border"
42         android:ems="10"
43         android:hint="@string/hint_name"
44         android:inputType="textUri"
45         android:padding="12dp"
46         android:textAppearance="?android:attr/textAppearanceSmall"
47         android:textSize="18dp"
48         android:textColor="#346D7D"
49         android:text="indiserver" />
```

```
50    <EditText
51        android:id="@+id/host"
52        android:layout_width="match_parent"
53        android:layout_height="wrap_content"
54        android:layout_below="@+id/name"
55        android:layout_centerHorizontal="true"
56        android:layout_marginTop="20dp"
57        android:background="@drawable/text_border"
58        android:ems="10"
59        android:hint="@string/hint_host"
60        android:inputType="textUri"
61        android:padding="12dp"
62        android:textAppearance="?android:attr/textAppearanceSmall"
63        android:textSize="18dp"
64        android:textColor="#346D7D"
65        android:text="indiserver.no-ip.org" />
66
67    <EditText
68        android:id="@+id/port"
69        android:layout_width="match_parent"
70        android:layout_height="wrap_content"
71        android:layout_below="@+id/host"
72        android:layout_centerHorizontal="true"
73        android:layout_marginBottom="20dp"
74        android:layout_marginTop="20dp"
75        android:background="@drawable/text_border"
76        android:ems="10"
77        android:hint="@string/hint_port"
78        android:inputType="number"
79        android:padding="12dp"
80        android:textAppearance="?android:attr/textAppearanceSmall"
81        android:textSize="18dp"
82        android:textColor="#346D7D"
83        android:text="7624" />
84
85    <Switch
86        android:layout_width="match_parent"
87        android:layout_height="wrap_content"
88        android:layout_centerHorizontal="true"
89        android:text="@string/autoconnect"
90        android:layout_marginRight="20dp"
91        android:layout_marginLeft="20dp"
92        android:layout_marginBottom="20dp"
93        android:layout_below="@+id/port"
94        android:id="@+id/autconnect" />
95
96    <Switch
97        android:layout_width="match_parent"
98        android:layout_height="wrap_content"
99        android:layout_centerHorizontal="true"
100       android:text="@string/blob_enable"
101       android:layout_marginRight="20dp"
102       android:layout_marginLeft="20dp"
103       android:layout_marginBottom="20dp"
104       android:layout_below="@+id/autconnect"
105       android:id="@+id/blob_receive" />
106
107
108
109    <Button
110        android:id="@+id/remove_log_button"
111        android:layout_width="wrap_content"
```

```
112     android:layout_height="wrap_content"
113     android:padding="10dp"
114     android:layout_marginBottom="30dp"
115     android:layout_marginTop="15dp"
116     android:text="@string/remove_log"
117     android:textSize="14dp"
118     android:layout_below="@+id/blob_receive"
119     android:layout_centerHorizontal="true"
120     android:enabled="false" />
121
122
123 <Button
124     android:id="@+id/edit_button"
125     android:layout_width="match_parent"
126     android:layout_height="wrap_content"
127     android:paddingBottom="20dp"
128     android:paddingTop="20dp"
129     android:layout_marginBottom="20dp"
130     android:text="@string/edit_connect"
131     android:textColor="@android:color/white"
132     android:background="#346D7D"
133     android:layout_alignParentLeft="true"
134     android:layout_alignParentStart="true"
135     android:textSize="18dp"
136     android:layout_below="@+id/remove_log_button" />
137
138
139 </RelativeLayout>
140 </ScrollView>
```

Una vez creada la clase y las dos vistas en archivos XML, debemos añadir un objeto de la nueva clase manejadora, a la lista con el resto de objetos para manejar propiedades. Según la prioridad establecida, el nuevo manejador de propiedades se colocarán en una posición concreta. En el segmento de código B.6 podemos ver esta lista. La función que se observa es llamada al inicio del programa para establecer todas las clases que pueden manejar propiedades. Para cada objeto, llamamos al método de la clase Config addUiPropertyManager(nuevo manejador)

Fragmento de código B.6: Lista de objetos manejadores de propiedades

```
1
2 private void setUiProperties() {
3     //add UI object
4     Config.init();
5     Config.addUiPropertyManager(new UIBlobPropertyManager());
6     Config.addUiPropertyManager(new UITextPropertyManager());
7     Config.addUiPropertyManager(new UISwitchPropertyManager());
8     Config.addUiPropertyManager(new UINumberPropertyManager());
9     Config.addUiPropertyManager(new UILightPropertyManager());
10    Config.addUiPropertyManager(new UIConnecPropertyManager());
11    Config.addUiPropertyManager(new UIAbortPropertyManager());
12
13 }
```

Este método se encuentra definido en la actividad principal que se encuentra en el archivo “MainActivity.java”. Debemos buscarla y añadir ahí el objeto de nuestra clase.

En la figura B.1 podemos ver la vista de las propiedades **Connection** y **Abort** en la lista, y en las figuras B.2 y B.3, sus vistas de edición



Figura B.1: Vista de las propiedades connection y abort

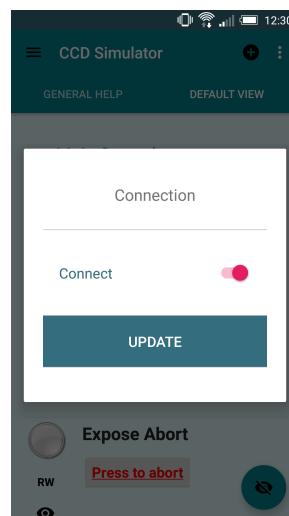


Figura B.2: Vista de edición para la propiedad Connection

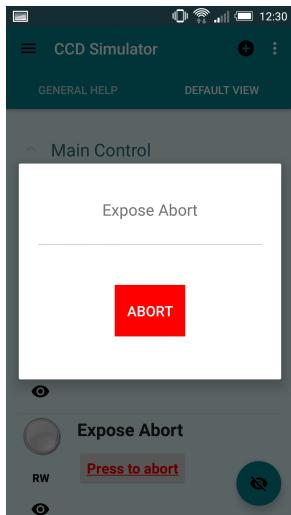


Figura B.3: Vista de edición para la propiedad Abort

B.1.2. Dispositivos

Al igual que con las propiedades, el software está diseñado para facilitar el diseño de nuevas clases para tener vistas especiales para algún dispositivo.

En el caso de las propiedades, se establecían prioridades para poder elegir de entre todas las posibilidades, cual iba a ser la interfaz de la propiedad concreta. En el caso de los dispositivos tenemos una interfaz de usuario por defecto, que lista las propiedades en una lista expandible. Pero se pueden añadir tantas como se deseen, pudiendo pasar de una a otra ya que se irían añadiendo al panel tabulado de la interfaz de usuario principal de la aplicación.

Para poder crear nuevas vistas para un dispositivo se ha diseñado la clase abstracta `DeviceView`.

Fragmento de código B.7: Clase abstracta DeviceView

```

1 public abstract class DeviceView extends Fragment {
2
3     static Device device;
4     protected int layout;
5
6     /**
7      * Check if this class can represent p
8      *
9      * @param dev Indi property
10     * @return True/false if class can represent p
11     */
12    public abstract boolean handlesDevice (Device dev);
13
14    public abstract String getName();
```

```

15
16     @Override
17     public abstract View onCreateView(LayoutInflater inflater,
18         ViewGroup container, Bundle savedInstanceState);

```

Para poder crear una nueva vista de dispositivo debemos:

- Crear una clase que herede de `DeviceView`.
- Crear un archivo *XML* con la definición de la interfaz de usuario.
- Añadir un objeto de la clase a la lista de clases manejadoras de dispositivos.

Como podemos ver en el fragmento de código B.7, tenemos dos atributos que son el dispositivo que vamos a manejar, y la referencia de **Android** al recurso donde se encuentra el archivo *XML* con la definición de la vista. Además debemos implementar las siguientes funciones:

- `handlesDevice`: Esta función recibe un dispositivo y debe comprobar si puede manejarlo o no devolviendo un valor booleano.
- `getName`: Esta función debe devolver el nombre que se desea que aparezca en la parte superior de la vista (vista tabulada).
- `onCreateView`: La clase abstracta hereda de `Fragment` y por ello esta función debe ser implementada, ya que se llamará cuando se llame a la clase para pintar la interfaz de usuario [16]. En ella deberemos crear la vista y realizar todas las inicializaciones que queramos.

En el fragmento de código B.8 podemos ver una clase para mostrar una vista alternativa de una estación meteorológica concreta. Es una vista simple que muestra una imagen de fondo, pero es ilustrativa del proceso que debemos seguir para crear nuevas clases para manejar dispositivos.

Fragmento de código B.8: Clase DeviceMeteoView

```

1  public class DeviceMeteoView extends DeviceView {
2      @Override
3      public boolean handlesDevice(Device dev) {
4          if(dev.getName().equals("Meteo")){
5              return true;
6          }else
7              return false;
8      }
9
10     @Override
11     public String getName() {

```

```
12     return "Meteo";
13 }
14
15 @Override
16 public View onCreateView(LayoutInflater inflater, ViewGroup
17     container, Bundle savedInstanceState) {
18     layout=R.layout.meteo_view;
19     View view = inflater.inflate(layout, container, false);
20     FrameLayout frame=(FrameLayout)view.findViewById(R.id.layout);
21     frame.setBackground(getResources().getDrawable(R.drawable.
22         aagcloudwatcher));
23     return view;
24 }
```

Al igual que ocurría con las propiedades, debemos crear una vista definiéndola en un archivo *XML* con los elementos visuales de **Android** que necesitemos.

Por último debemos agregar un objeto de nuestra clase a la lista de manejadores de dispositivos, como se ve en el fragmento de código B.9 llamando al método de la clase `Config addDeviceView(nuevo manejador)`

Fragmento de código B.9: Lista de objetos manejadores de dispositivos

```
1 private void setDeviceViews() {
2     Config.addDeviceView(new DeviceMeteoView());
3 }
```


Bibliografía

- [1] “Standards for Astronomy”, ASCOM, Fecha de último acceso: 15/11/2014.
<http://ascom-standards.org/>
- [2] “INDI”, Fecha de último acceso: 1/9/2015.
<http://indilib.org/>
- [3] “La Historia de la astronomía”, Fecha de último acceso: 29/11/2014.
<http://www.conocimientosweb.net/dcmt/ficha3960.html>
- [4] “Astronomía Sur”, Fecha de último acceso: 15/6/2015.
<http://www.astrosurf.com/astrosur/telescopios.htm>
- [5] “El telescopio: la historia del invento que revolucionó la ciencia”, Elena Sanz , Fecha de último acceso: 15/6/2015.
<http://www.muyinteresante.es/ciencia/articulo/el-telescopio-la-historia-del-invento-que-revoluciono-la-ciencia>
- [6] “CCD”,Centro Nacional de Biotecnología , Fecha de último acceso: 15/6/2015.
http://www.cnb.csic.es/~fotonica/Photonic_en/Review/ccd1.htm
- [7] “Ópticas adaptativas”, Instituto de astrofísica de Canarias , Fecha de último acceso: 15/6/2015.
<http://www.iac.es/gabinete/iacnoticias/2-99/gtc/gtc3.htm>
- [8] “IDC analyze the future”, IDC, Fecha de último acceso: 15/2/2015.
<http://www.idc.com/getdoc.jsp?containerId=prUS25450615>
- [9] “Proyectos Ágiles”, Fecha de último acceso: 10/10/2014.
<http://www.proyectosagiles.org/que-es-scrum>
- [10] “Android Developer”, Fecha de último acceso: 25/7/2015.
<http://developer.android.com/develop/index.html>
- [11] “Java API”, Fecha de último acceso: 20/7/2015.
<http://docs.oracle.com/javase/7/docs/api/>

- [12] “How to make Material Desing Navigation Drawer”, Fecha de último acceso: 17/3/2015.
<http://www.android4devs.com/2014/12/how-to-make-material-design-navigation-drawer.html>
- [13] “Android Tab Layout with Swipable Views”, Ravi Tamada, Fecha de último acceso: 2/4/2015.
<http://www.androidhive.info/2013/10/android-tab-layout-with-swipeable-views-1/>
- [14] “Android Expandible List View Tutorial”, Ravi Tamada, Fecha de último acceso: 20/5/2015.
<http://www.androidhive.info/2013/07/android-expandable-list-view-tutorial/>
- [15] “How to Make Floating Action Button”, Fecha de último acceso: 15/4/2015.
<http://www.android4devs.com/2015/03/how-to-make-floating-action-button-fab.html>
- [16] “Activity Life Cycle Android”, Fecha de último acceso: 5/2/2015.
<http://www.codelearn.org/android-tutorial/android-activity>
- [17] “Android Studio”, Fecha de último acceso: 10/10/2014.
<https://developer.android.com/sdk/index.html>
- [18] “GitHub”, Fecha de último acceso: 11/9/2015.
<https://github.com/>
- [19] “Sublime Text 2”, Fecha de último acceso: 1/9/2014.
<http://www.sublimetext.com/>

