# ERELIS

## Una aplicació per a l'estalvi de bateria

## per a la plataforma Android

TFC - Arquitectura de Computadors i S.O.

Entrega Final

Juny del 2011

Alumne: Jesús Corrius Llavina

UOC. TFC. Curs 2011-2012

Consultor: Francesc Guim Bernat

# Índex de continguts

# 1. Motivació del projecte

En primer lloc, crec que és interessant veure quina és la meva experiència com a  informàtic, per tal d'entendre les meves motivacions a l'hora de realitzar aquest projecte.

Sóc llicenciat en Comunicació Audiovisual i programador informàtic amb més de 10 anys d'experiència en C, C++ i també en Python. He treballat en diferents entorns com són web, Windows, Linux, Mac, etc. Tinc molta experiència en entorns multiplataforma (sóc hacker d'un projecte de programari lliure anomenat LibreOffice, abans OpenOffice.org) i de programació de sistemes i multimèdia. Vaig treballar per l'empresa Fluendo en la creació de còdecs multimèdia lliures per al sistema operatiu Linux que ara mateix utilitzen diferents projectes com Gstreamer o VLC.

Últimament, he estat treballant com a freelance i m'he trobat amb una gran demanda d'aplicacions per a l'iPhone, que és un mercat molt interessant des d'un punt de vista professional. Com a resultat vaig aprendre a fer aplicacions per iPhone i vaig estar treballant per algunes empreses que necessitaven una aplicació per a aquest entorn. La majoria d'aplicacions eren molt simples, però algunes requerien uns coneixements a baix nivell de la plataforma que em van permetre conèixer bé com funciona internament.

Un cop he conegut com funciona l'iPhone, i com crear aplicacions en aquesta plataforma, he pensat que seria molt interessant també aprendre com funciona la plataforma Android. Es tracta principalment d'un sistema operatiu per a dispositius mòbils, abanderat per Google i la Open Handset Alliance (OHA), que és el darrer membre de l'escena de la telefonia cel·lular i també dels PDA/smartphones.  El seu continu creixement a ritme vertiginós trimestre rere trimestre ha superat totes les expectatives i, tot i ser un nouvingut, ja ha superat a tots els seus competidors directes: l'iPhone d'Apple, la plataforma Blackberry, Symbian o el Windows Mobile de la totpoderosa Microsoft.

El meu projecte serà una aplicació per a la plataforma Android, utilitzant el SDK sobre Eclipse. Aquesta aplicació serà un servei que correrà com a servei en el dispositiu i s'encarregarà de monitoritzat l'ús de memòria i la prioritat dels processos i s'encarregarà d'optimitzar els recursos com, per exemple, en el cas que hi ha hagi poca memòria, bateria o

temps de CPU disponibles, intentant assignar els recursos restants a les aplicacions més importants i tancant les aplicacions i serveis que siguin prescindibles. L'àrea en què em centraré especialment és en el control de la bateria, intentant optimitzar el dispositiu quan hi hagi la bateria baixa, ja que aquest és un dels grans problemes dels dispositius mòbils. És conegut que la majoria de fabricants de maquinari i programari dediquen una gran quantitat d'esforços a intentar allargar la vida de la bateria tan com sigui possible.

El projecte, en definitiva, em sembla molt interessant perquè toca dos temes que m'interessen especialment com és la programació de sistemes i la optimització de recursos.

# 2. Objectius del projecte

El projecte s'anomena erelis. La paraula erelis és àguila en lituà, i d'alguna manera és una metàfora del què fa l'aplicació que es dedica a anar observant el què passa en el dispositiu "des del cel" (en aquest cas, en segon pla) sense deixar-se notar ni pertorbar la vida a ras de terra i només passa a l'acció quan és necessari (en aquest cas, desactivant serveis quan hi ha la bateria baixa, o bé "caçant" les aplicacions que no es comporten correctament).

L'objectiu del projecte es crear una aplicació que s'executi com a servei en un dispositiu Android i s'encarregui de monitoritzar els recursos del sistema en temps real. Així doncs aquesta aplicació funcionarà en segon pla i no hi haurà cap interfície d'usuari visible a no ser que sigui necessària la intervenció de l'usuari o bé aquest vulgui personalitzar algun aspecte del funcionament del programa.

Aquesta aplicació es centrarà especialment en el control de l'estat de la bateria del dispositiu per tal d'intentar allargar al màxim la vida de la bateria. Hi ha diferents maneres de fer-ho, però la manera més efectiva és anar parant els diferents components de maquinari que no estan en ús (xarxa wifi, bluetooth, etc.) i limitant l'ús dels recursos del sistema. Alguns dispositius, per exemple, permeten reduir de manera programàtica la velocitat de la CPU, amb la qual cosa les aplicacions van més lentes, però la vida de la bateria és més llarga. En aquest cas, l'important és trobar un bon balanç entre els dos.

L'aplicació també monitoritza diferents característiques de l'estat del sistema, tan a nivell general com a nivell d'aplicació, i pren decisions sobre els processos que estan funcionant segons aquestes mesures. Aquests dos nivells de monitorització impliquen dos nivells bàsics de funcionament detallats a continuació:

1. En primer lloc tenim el nivell general del dispositiu. En aquest punt es monitoritza l'estat general del sistema i es prenen decisions sobre els processos que estan funcionant quan el nivell de memòria, bateria o altres recursos és baix. En aquest cas, un procediment habitual serà l'eliminació de processos que siguin prescindibles. Però es poden establir altres alternatives.

2. En segon lloc tenim el nivell específic de cada procés. La monitorització també es produirà per procés per tal de comprovar si els processos es comporten “correctament” o no. Si un procés es passa amb l'ús de certs recursos del sistema, podem canviar-li la prioritat o bé eliminar-lo. L'opció normal en cas que el procés utilitzi més recursos del normal serà tancar el procés de manera neta (sense que l'usuari perdi informació) i només en casos extrems es demanarà al sistema que forci l'eliminació del programa.

Si bé aquest és el funcionament bàsic de l'aplicació, l'usuari haurà de poder canviar o personalitzar certes coses. L'usuari ha de poder accedir als paràmetres de monitorització per poder observar quin és el comportament del seu dispositiu en tot moment i també fer els canvis que creguin convenients en el comportament per defecte de les aplicacions. Així per exemple, una aplicació que faci streaming de vídeo, ha de poder utilitzar molts més recursos que no pas una aplicació molt més simple que no faci un us tan intensiu de memòria o CPU.

Un altre aspecte interessant de l'aplicació és la possibilitat d'intentar controlar l'ús de la bateria en el dispositiu amb una gestió més eficient dels processos que s'estan executant. En aquest moment és impossible predir de quina manera la vida de la bateria pot veure's modificada, però serà interessant mesurar-ho un cop tingui l'aplicació en funcionament en un dispositiu real.

Aquests són els objectius principals del projecte. Tot i això, hi ha a la possibilitat d'incorporar algunes millores o fer petits canvis segons quin sigui el resultat de les proves fetes amb el dispositiu. I si hi ha alguna altra manera per tal d'optimitzar l'eficiència de les aplicacions, es poden implementar també.

# 3. Descripció de les tasques

El projecte es divideix en diferents tasques que, de manera lògica, s'hauran de realitzar una darrera l'altra. En el següent apartat veurem la planificació d'aquestes tasques i aquí ens centrarem a explicar amb cert detall en què consisteixen cada una d'elles. S'ha de tenir en compte que un programa informàtic és una peça d'enginyeria prou complicada i requereix una bona planificació i també una bona cerca de documentació i investigació sobre allò que es vol realitzar.

En aquest projecte les tasques a realitzar són les següents:

1. Cerca de treball relacionat: en aquesta tasca es buscaran treballs o aplicacions relacionades amb els objectius del nostre projecte. Es tindran especialment en compte les eines de desenvolupament que permeten mesurar l'estat del sistema i també la performance de les aplicacions. Al mateix temps es buscarà a l'Android Market aplicacions que facin operacions similars. També es buscaran idees per a l'interfície d'usuari.
2. Documentació: en aquesta tasca es buscarà i llegirà tota la informació necessària per tal de fer l'aplicació, especialment la del Android SDK que proporciona Google, tot i que també existeixen moltes altres pàgines web que ofereixen informació i recursos sobre la plataforma Android. Un cop s'haurà acomplert aquesta tasca, estarem en condicions de passar a la tasca següent.
3. Disseny: en aquesta tasca es farà el disseny de l'aplicació. La tasca anterior ens haurà donat tota la informació necessària per tal d'acomplir aquesta tasca satisfactòriament. Es farà el disseny de l'aplicació des del punt de vista d'enginyeria (classes, APIs que farem servir, etc.) i també la interfície d'usuari i les diferents pantalles que tindrà l'aplicació.
4. Instal·lació de l'entorn de treball: en aquesta tasca s'instal·larà l'entorn de treball i totes les eines per fer l'aplicació, en aquest cas serà l'entorn Eclipse i, evidentment, l'Android SDK.

5. Implementació: en aquesta tasca es durà a terme la programació pròpiament dita de l'aplicació, això és, la creació de les interfícies d'usuari de l'aplicació i també les classes i les crides a l'API del sistema que ens permetran tenir una aplicació perfectament funcional i llesta per poder-la provar.
6. Avaluació de la implementació: en aquesta tasca es provarà l'aplicació que s'haurà creat en la tasca anterior. Si no es disposa de cap dispositiu Android per fer la prova, serà provat principalment en l'emulador que ens ofereix l'Android SDK. Un cop tinguem totes les proves fetes, documentarem els problemes que hem trobat i les millores que s'han de fer a l'aplicació de cara a l'entrega final.
7. Millores finals sobre la implementació: en aquesta tasca s'utilitzarà tota la informació obtinguda en l'apartat anterior per tal de modificar el comportament de l'aplicació o bé resoldre els problemes trobats. Aquesta serà l'última tasca que, una vegada realitzada, permetrà tenir l'aplicació definitiva i realitzar l'entrega final.

# 4. Planificació de les tasques

En l'apartat anterior s'han vist les tasques i ara es farà una estimació temporal de les mateixes. Farem el càlculs tenint en compte que el curs són unes 15 setmanes i així doncs es distribuiran les tasques en aquest context. Es pot afinar una mica més, però s'ha decidit utilitzar les setmanes com a unitat ja que aquesta unitat de temps és suficientment granular i es representativa de la manera d'organitzar el temps a la UOC.

Tenint en compte els criteris anteriors, s'ha distribuït el temps de les tasques de la manera següent:

- Cerca de treball relacionat: 1 setmana
- Documentació: 1 setmana
- Disseny: 3 setmanes
- Instal·lació de l'entorn de treball: 0,5 setmanes
- Implementació: 4 setmanes
- Avaluació de la implementació: 2 setmanes
- Millores finals sobre la implementació: 3 setmanes

Si bé és cert que algunes d'aquestes tasques es poden fer en paral·lel, especialment les primeres de la llista, la gran majoria de tasques depenen les unes de les altres. O sigui que el diagrama de Gantt amb la planificació tindrà l'aspecte següent:

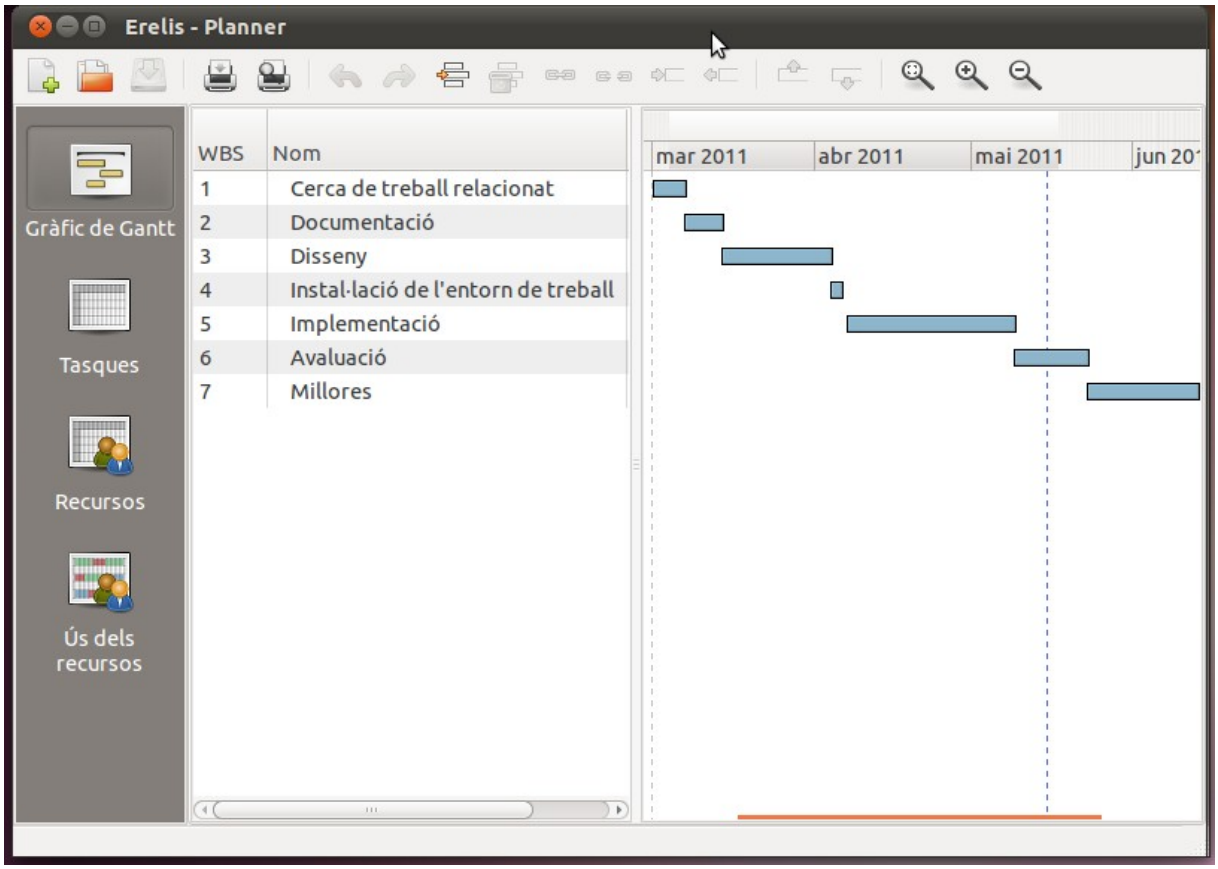



*Diagrama de Gantt amb la planificació de les tasques*

I a la taula següent podem trobar amb més detall les dates d'inici i de finalització de cada tasca:

| Tasca | Data inic | Data finalització |
|---|---|---|
| Cerca de treball relacionat | 01/03/11 | 06/03/11 |
| Documentació | 07/03/11 | 13/03/11 |
| Disseny | 14/03/11 | 03/04/11 |
| Instal·lació de l'entorn de treball | 04/04/11 | 06/04/11 |
| Implementació | 07/04/11 | 08/05/11 |
| Avaluació | 09/05/11 | 22/05/11 |
| Millores | 23/05/11 | 12/06/11 |
| Total projecte | 01/03/11 | 12/06/11 |

*Planificació setmanal de tasques*

En quant a l'estimació econòmica del projecte, estem parlant d'un desenvolupament de 15 setmanes de feina pensada per a un sol desenvolupador. Tot i això, hem de tenir en compte que la planificació és setmanal i no implica treballar vuit hores al dia en el projecte. Fent una estimació més realista, el projecte es pot fer perfectament en un mes de feina per un programador. Si posem un sou de 30.000 euros a l'any, el cost econòmic de l'aplicació seria:

30.000 euros / 12 = 2500 euros bruts.

# 5. La plataforma Android

En aquest apartat es veurà una introducció a la plataforma Android que ens portarà a través de la seva història i, molt especialment, la tecnologia sobre la que està construïda. S'aprofitarà per veure també la difusió de la plataforma que, com es veurà, és molt important i majoritària i això ens serveix com a bon justificant de la nostra elecció tecnològica.

## *5.1. Història*

La plataforma Android, desenvolupada actualment per Google i els membres de l'Open Handset Alliance, és un conjunt de solucions per a dispositius mòbils que ofereix un sistema operatiu, programari intermediari i les aplicacions més importants. El sistema operatiu està basat en el kernel de Linux i en un conjunt de llibreries estàndards de programari lliure, que són el nucli sobre les quals es desenvolupa la plataforma. El projecte utilitza la llicència Apache[Oha11] per a la major part del codi, la qual cosa fa que que també sigui un projecte de programari lliure o de codi font obert [Goo11a].

El desenvolupador original del projecte va ser l'empresa Android, Inc., que Google va comprar al 2005 [Bus05]. Dos anys després de l'adquisició, el 25 de novembre del 2007, Google va anunciar la creació de l'Android Open Source Project (AOSP) que és l'organització que s'encarrega del manteniment i el desenvolupament de la plataforma [Goo11b].

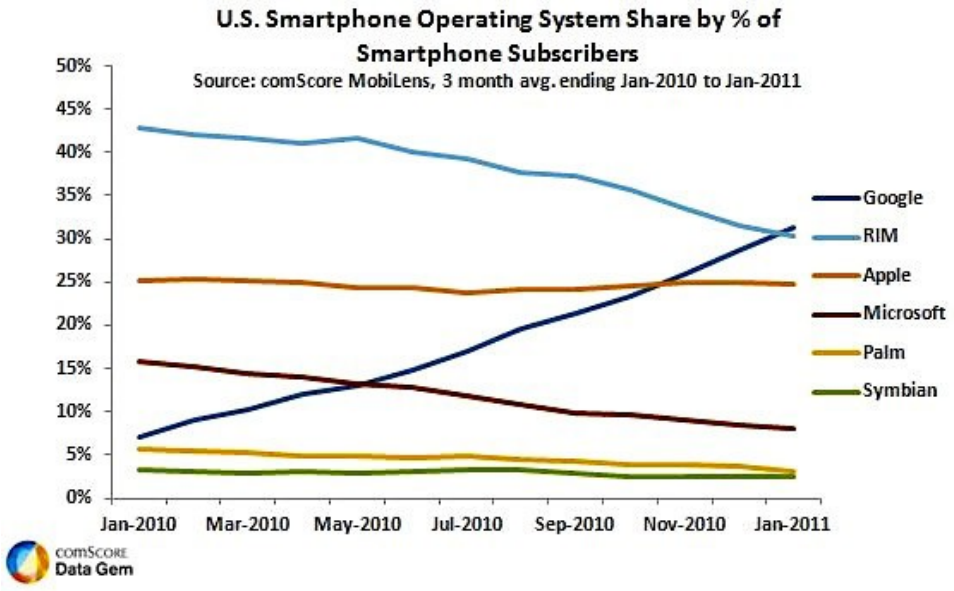
Al mateix temps també es va anunciar la creació de l'Open Handset Alliance [Oha07], que és un consorci de més 80 empreses de maquinari, programari i telefonia d'arreu del món que té com a objectiu la creació, el manteniment i la difusió d'un sistema de codi font obert per a telefonia mòbil. Els membres inicials de l'Open Handset Alliance van ser: Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm, Samsung Electronics, Sprint Nextel, T-Mobile i Texas Instruments [Oha07]. Més tard és van afegir altres empreses com ARM Holdings, Atheros Communications, Asustek Computer Inc, Garmin Ltd, PacketVideo, Softbank, Sony Ericsson, Toshiba Corp, and Vodafone Group Plc [Reu08].

La primera versió pública de la plataforma, la versió 1.0, es alliberar el 21 d'octubre del 2008. En el moment de fer aquest treball existeixen dues versions de la plataforma: la 2.3 (Gingerbread) i la 3.0 (Honeycomb). La primera versió està pensada per a telèfons mòbils[And10] i la segona per a tablets[Ana11].

## *5.2. Difusió*

El nombre d'usuaris de dispositius que utilitzen la plataforma Android ha tingut un creixement espectacular en molt poc temps. Durant el segon trimestre del 2009 la quota de mercat era del 2.8% [App09] i al 2010 havia crescut fins al 33% [Reu11].

El febrer del 2010 comScore va afirmar que Android tenia el 9% del mercat de telèfons intel·ligents dels Estats Units, partint del 5.2% del novembre del 2009. Al final del 2010, la quota de mercat havia pujat fins al 21.4% [Com10].



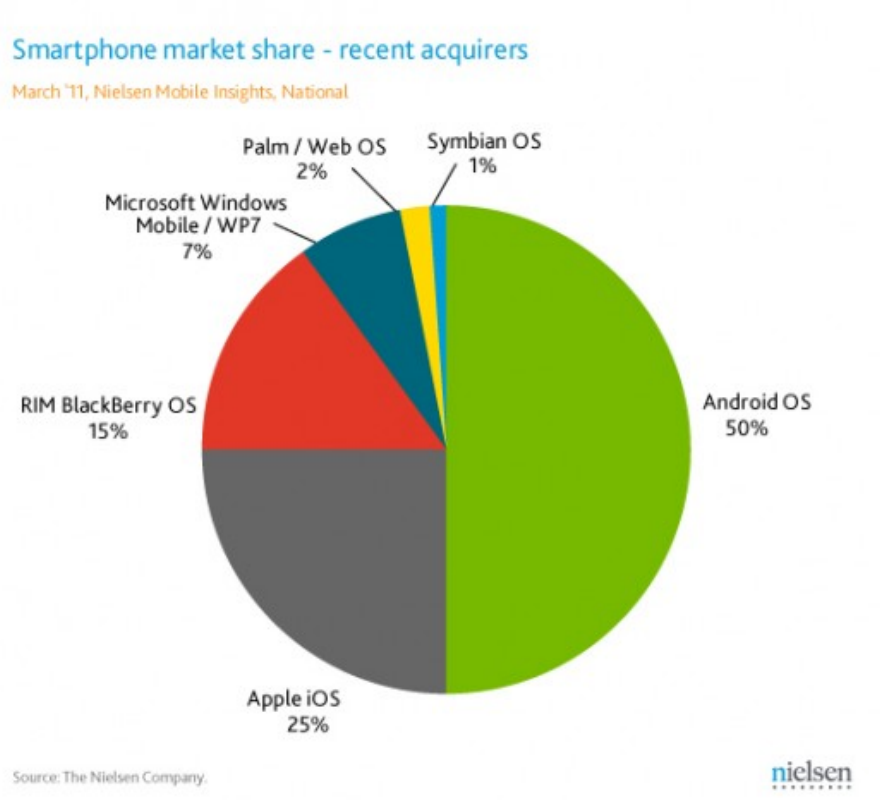

*Evolució de la quota de mercat d'Android. Font: comScore*

Segons el NPD Group, les unitats venudes de telèfons intel·ligents amb Android el situen en el primer lloc als Estats Units, en el segon i tercer trimestres del 2010 amb una quota de mercat del 43,6% en el tercer trimestre [Cne10].

Al febrer de 2011, durant el Mobile World Congress 2011 a Barcelona, Eric Schmidt, llavors

CEO de Google, va anunciar que la plataforma Android ha arribat a 350.000 activacions per dia [Pha11].

Per fer-nos a la idea de la seva gran popularitat, segons les dades publicades per l'empresa americana Nielsen a principis de març del 2011 [Der11], Android ha esdevingut el sistema operatiu per a smartphones amb més demanda per part dels usuaris dels Estats Units. En el gràfic següent podem veure quin és l'últim dispositiu mòbil adquirit pels usuaris que han participat en l'estudi:




*Quota de mercat dels smartphones, març del 2011. Font: Nielsen.*

Finalment, el 10 de maig del 2011, Google ha anunciat que 400.000 nous dispositius s'activen cada dia i que, en total, se n'han activat ja més de 100 milions [Goo11c].

## *5.3. Tecnologia*

En aquest apartat s'estudiarà la plataforma Android des d'un punt de vista estrictament tecnològic. En primer lloc, es veurà una introducció general a la seva arquitectura i després

ens centrarem en dos aspectes claus per a aquest projecte: la creació d'aplicacions i el sistema de gestió de processos i recursos de l'Android.

### *5.3.1. Arquitectura de la plataforma*

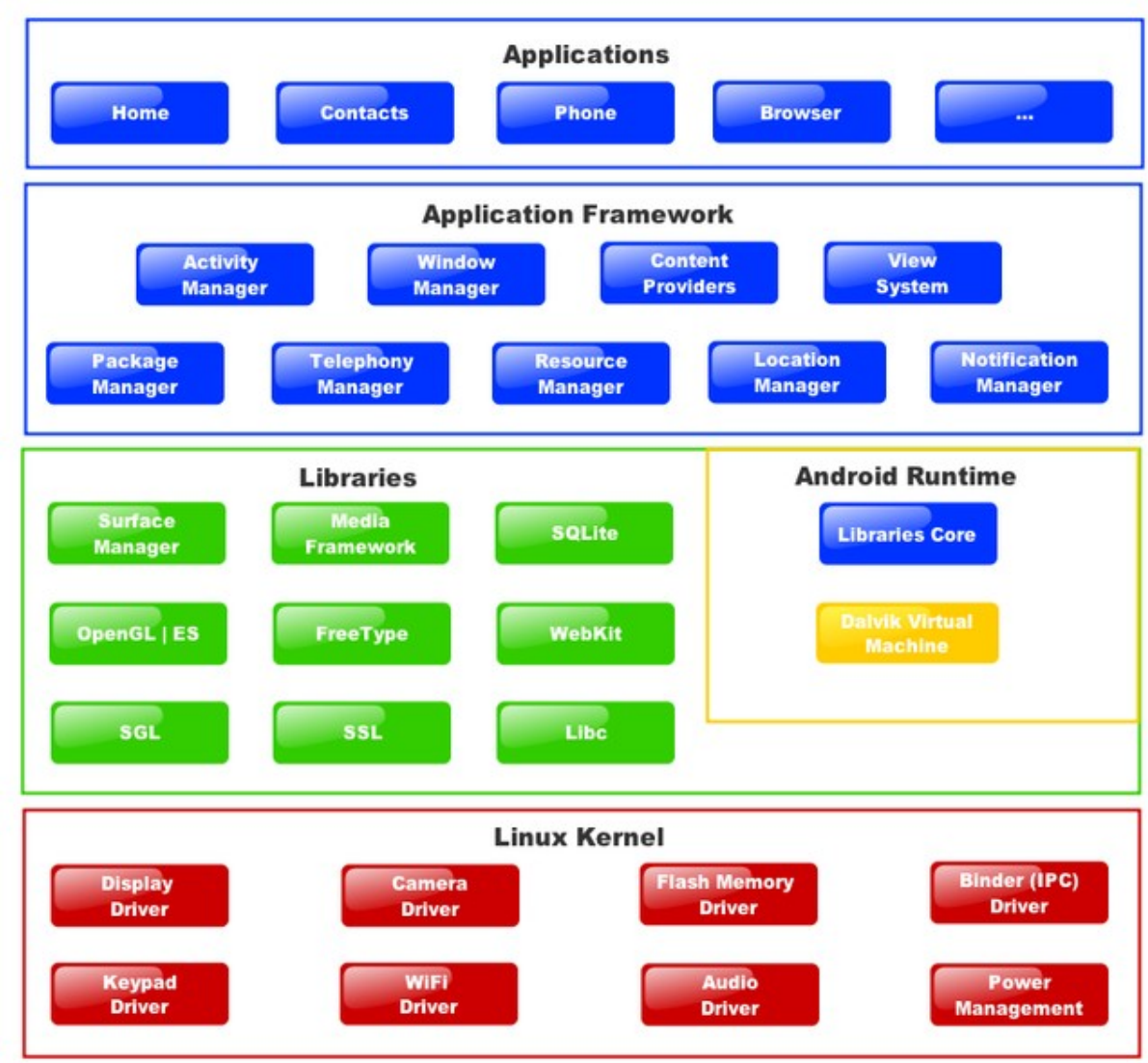
En el seu nivell més baix, el sistema operatiu està basat en un nucli de Linux de la versió 2.6 que actua com a capa d'abstracció entre el maquinari i les capes de programari superiors. Inclou els controladors bàsics per als dispositius disponibles al terminal: pantalla, àudio, xarxa sense fils, gestió d'energia, etc.

Per altra banda, a les capes superiors, el conjunt de solucions Android consisteix en una sèrie d'aplicacions escrites en el llenguatge de programació Java que, al mateix temps, s'executen sobre un framework de llibreries principals escrites també en Java. Tot aquest codi s'executa sobre una màquina virtual que s'anomena Dalvik [Goo11d].

Però no només hi ha Java en aquest conjunt de solucions, sinó que podem trobar també llibreries i programes escrits en altres llenguatges de programació.  Les llibreries escrites en C inclouen el gestor de superfícies, l'OpenGL ES 2.0 3D, el WebKit, el motor gràfic SGL, l'SSL, i la Bionic libc. El sistema operatiu Android, inclòs el nucli de Linux, es compon aproximadament de12 milions de línies de codi, incloent 3 milions de línies de XML, 2,8 milions de línies de C, 2.1 milions de línies de Java, i 1,75 milions de línies de C++ [Gub10].

Tot i que la majoria de desenvolupament sobre la plataforma es fa amb Java, podem utilitzar també altres llenguatges de programació. De fet, gràcies al Android Native Development Kit, podem crear codi en C i altres llenguatges de programació que es puguin compilar amb el gcc a codi ARM natiu i instal·lar-los al dispositiu [Ben07]. Després aquest codi es pot executar des del Java com si fos una llibreria nativa escrita en aquest llenguatge de programació.

En el gràfic següent podem veure d'una manera molt clara quina és l'arquitectura del sistema:




*Arquitectura dels sistemes Android. Font: Google.*

Com es pot veure en el gràfic, l'arquitectura té diferents nivells o capes, amb el nucli de Linux a sota de tot i les aplicacions en el punt més alt. Una de les maneres de separar les diferents capes és veure què s'executa amb codi natiu, i quines parts utilitzen Java.

Codi natiu:

A baix nivell tenim el nucli de Linux que s'encarrega de carregar els controladors del maquinari i controlar la gestió de processos, memòria, entrada i sortida, etc. Per sobre el nucli, tenim les llibreries del sistema, com la llibreria estàndard de C, i les altres llibreries que ens ofereixen el codi bàsic sobre el qual es construeix la resta del sistema. L'últim element que

utilitza codi natiu és la màquina virtual de Java Dalvik, que és la que executa tot el codi Java que trobem al nivell que veurem a continuació.

Codi Java:

La màquina virtual Dalvik, vista a l'apartat anterior, executa tot el codi Java de la plataforma, però no tot el codi Java es troba en el mateix nivell. En primer lloc tenim les llibreries bàsiques sobre les quals es construeix tota la resta, aquestes inclouen, per exemple, l'accés des de Java del codi natiu en C o C++ de les llibreries que hem vist en l'apartat anterior. Sobre aquestes, tenim el framework que ens proporciona l'Android SDK propiament dit, sobre el qual es construeixen les aplicacions, i que ens ofereix totes les abstraccions típiques d'un sistema per a construir aplicacions, com pot ser una "classe finestra" o bé l'accés d'alt nivell sobre les propietats de la càmera dels dispositius. És sobre aquesta API que es desenvolupen les aplicacions que interactuen amb l'usuari final, tant les que venen de sèrie amb el sistema com les que els desenvolupadors externs poden crear.

### *5.3.2. Desenvolupament d'aplicacions*

Les aplicacions per a la plataforma Android es desenvolupa en Java utilitzant l'Android Software Development Kit, la primera versió del qual es va publicar alguns mesos després de la presentació pública del projecte. L'Android SDK inclou un conjunt de llibreries, un depurador i un emulador que permet executar el codi sense necessitat d'un dispositiu real.

També, com ja hem comentat en l'apartat anterior, existeix un kit de desenvolupament en codi natiu conegut com a NDK (Native Development Kit) que permet escriure i compilar programari per a executar directament per a processadors de l'arquitectura ARM en comptes d'utilitzar la màquina virtual. Aquesta aproximació al desenvolupament proporciona beneficis en rendiment i versatilitat, però és només adequat per a aplicacions que tenen unes necessitats molt específiques en aquest sentit.

Tot i que hem de realitzar algunes activitats de baix nivell, per crear l'aplicació per a aquest projecte s'utilitzarà el mètode de desenvolupament estàndard per a la plataforma que és el llenguatge de programació Java conjuntament amb l'Android SDK. A més a més de la plataforma en sí, l'entorn de desenvolupament que farem servir serà un IDE (o sigui:

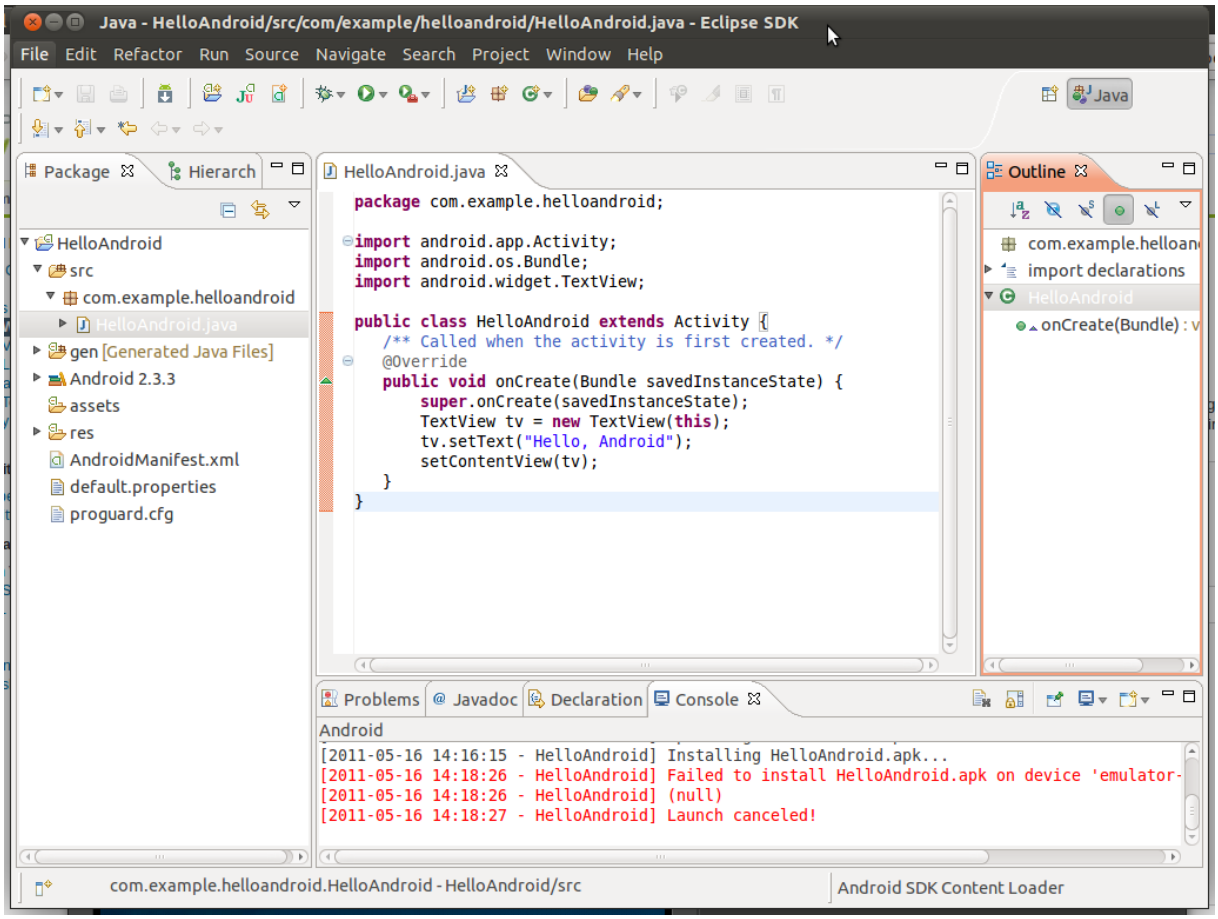
Integrated Development Environment o Entorn de Desenvolupament Integrat) que si bé no és en absolut necessari i seria possible utilitzar un editor de text i les eines de línia d'ordres a través d'un terminal per a escriure, compilar, depurar i executar un programa, la feina és molt més fàcil i ràpida de realitzar a través d'un entorn integrat.

L'entorn oficialment suportat i recomanat per al desenvolupament en Android és Eclipse (3.4 o superior) i el connector específic d'aquest per a Android, anomenat ADT plugin (Android Development Tools) [Goo10b].

Per tal de configurar l'entorn de desenvolupament es necessiten els següents elements:

- JDK (Java Development Kit) versió 5 o 6: Es pot descarregar des de la pàgina web següent: http://www.oracle.com/technetwork/java/javase/downloads/index.html . És important remarcar que es necessita el JDK, ja que amb el JRE (Java Runtime Environment o Entorn d'Execució de Java), que és el que es troba normalment instal·lat en els nostres sistemes operatius, no n'hi ha prou. La versió utilitzada en aquest projecte és la JDK versió 6 Update 25.
- Eclipse IDE: Es pot descarregar a la pàgina web: http://www.eclipse.org/downloads/ . Per a aquest projecte es va instal·lar la versió clàssica 3.6.2 (Helios) en la seva edició per a arquitectures de 64 bits.
- Android SDK: Aquest component essencial es troba disponible a l'adreça web següent: http://developer.android.com/sdk/index.html . Pel projecte s'ha utilitzat la versió 11, que era l'última versió disponible en aquell moment.
- ADT Plugin: Aquest és el component que integra l'Android SDK amb l'entorn integrat de desenvolupament Eclipse. Es pot descarregar des de la pàgina web següent: http://developer.android.com/sdk/eclipse-adt.html . La versió que s'ha utilitzat en el projecte és la 10.0.1.

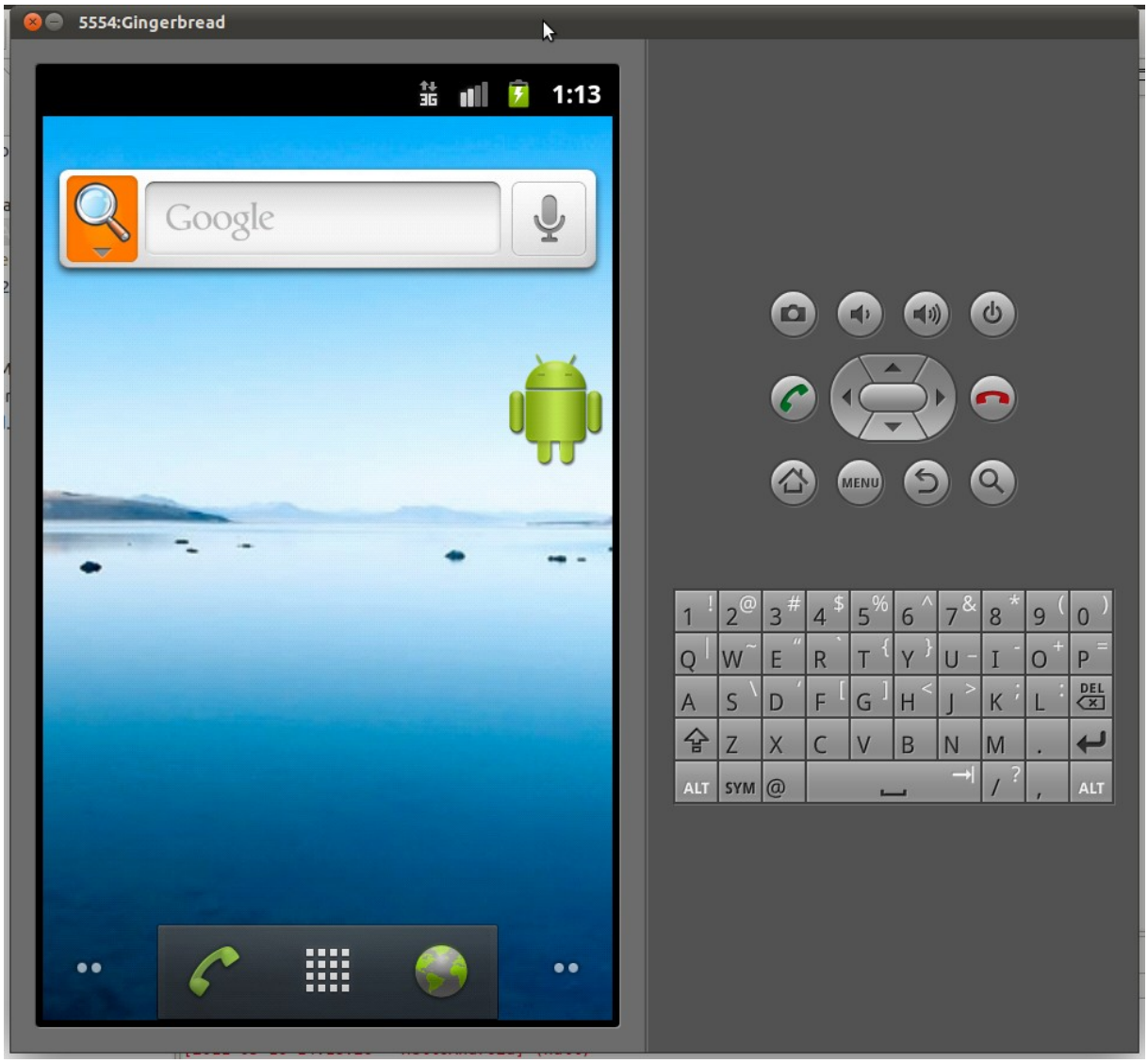
A la figura següent es pot veure l'entorn de desenvolupament integrat Eclipse configurat per a treballar amb la plataforma Android:




*Entorn de desenvolupament de l'Android SDK per a Eclipse. Font: elaboració pròpia.*

Per a depurar el programari s'utilitza l'emulador incorporat al SDK. Per tal de crear un nou emulador cal engegar el “Android SDK and AVD Manager” (per accedir a ell es pot utilitzar la icona de la barra superior o bé es pot utilitzar la icona corresponent del menú 'Window' que trobem a la part superior de la finestra de l'Eclipse) i crear-la amb el botó 'New' tot escollint els paràmetres que volem personalitzar. S'ha de tenir en compte que si el directori d'usuari no es troba en la seva ubicació per defecte, hem de modificar la variable d'entorn ANDROID_SDK_HOME perquè indiqui el el lloc corresponent en el nostre sistema de fitxers.

En la següent captura de pantalla, es pot veure l'emulador de l'Android en funcionament. La versió de l'emulador utilitzada és la 2.3.3 (Gingerbread):




*Emulador de dispositiu Android. Font: elaboració pròpia.*

# 6. Gestió de recursos i processos a la plataforma Android

Després de fer una introducció a la plataforma Android i a la seva tecnologia, és important veure els detalls tècnics que estan més relacionats amb l'aplicació que volem crear. Com que erelis és fonamentalment un programa d'estalvi d'energia relacionat íntimament amb la gestió de recursos del sistema i l'optimització dels processos en execució, aquest apartat es dedicarà íntegrament a parlar  sobre aquests temes.

Tal i com s'ha vist en els apartats anteriors, la plataforma Android es construeix sobre la versió 2.6 del nucli de Linux [Cle07]. Això vol dir que, internament, la gestió de recursos i processos és molt similar a qualsevol altre sistema que es basi en aquest sistema operatiu o fins i tot a qualsevol sistema tipus UNIX que implementi l'estàndard POSIX.

En aquest punt de l'exposició és important, però, tenir en compte dues consideracions molt importants. La primera és que es tracta d'un sistema Linux modificat, la qual cosa implica que si bé no hi ha diferències importants en la filosofia general del sistema, si que trobem alguns punts on difereixen. Aquestes diferencies han estat una font de conflicte permanent entre els desenvolupadors del nucli de Linux i Google i això ha fet que algunes millores que ha fet Google en la seva plataforma no s'hagin contribuït al nucli de Linux [May10]. Les parts que han generat més polèmica han estat alguns controladors de maquinari, que en aquest treball no ens interessen especialment, i les modificacions en l'estalvi d'energia, sobre les quals parlarem més endavant.

La segona consideració important a tenir en compte és quin tipus d'accés ens dóna a aquestes característiques l'API (Application Programming Interface o Interfície de Programació d'Aplicacions) de l'Android SDK. La nostra aplicació utilitzarà una API d'alt nivell en el llenguatge de programació Java per tal d'accedir a totes les característiques i funcionalitat del sistema i, com és natural, l'accés a algunes funcionalitat de molt baix nivell, o bé no serà possible, o bé s'hauran d'implementar seguint els condicionants i les restriccions que ofereix aquesta API.

A més a més, s'ha de tenir en compte que també hi ha nivells dins de les APIs en Java. Algunes d'aquestes simplement fan de pont entre la nostra aplicació i les crides al sistema

operatiu i d'altres implementen diferents nivells d'abstracció. Es pot il·lustrar aquest fet tot avançant-nos una mica a la propera secció. En els cas de la gestió de processos tenim la classe Process i la classe ActivityManager. La primera és molt més propera al sistema operatiu, ja que, per exemple, ens permet enviar senyals als processos, mentre que la segona classe és molt més abstracta i només ens ofereix funcionalitat i estadístiques de molt més alt nivell.

## *6.1. Gestió de processos*

La plataforma Android utilitza, en general, el mateix sistema de processos que qualsevol altre sistema operatiu que segueix l'estàndard POSIX (com pot ser UNIX o Linux) però amb una diferència fonamental: els processos continuen en memòria després d'acabar la seva activitat i només s'eliminen quan el sistema necessita més memòria. La idea que hi ha darrere d'això és que els processos carregats en memòria no suposen cap impacte en el rendiment del dispositiu mòbil, però s'executen molt més ràpidament si l'usuari torna a fer córrer l'aplicació [Bal10].

Cada aplicació Android s'executa en un procés separat dins de la seva pròpia instància de la màquina virtual de Java Dalvik i és aquesta màquina virtual la que s'encarrega de gestionar els recursos i d'eliminar els processos segons sigui necessari.

El procediment que se segueix  per a eliminar els processos en memòria es pot resumir dient que el sistema té en memòria una llista dels processos en memòria ordenats pel criteri d'antiguitat i que elimina en primer lloc els processos més antics dels que es troben carregats en memòria. Però això no és del tot acurat perquè també hi influeix un criteri de prioritat que s'assigna a cada un dels processos.

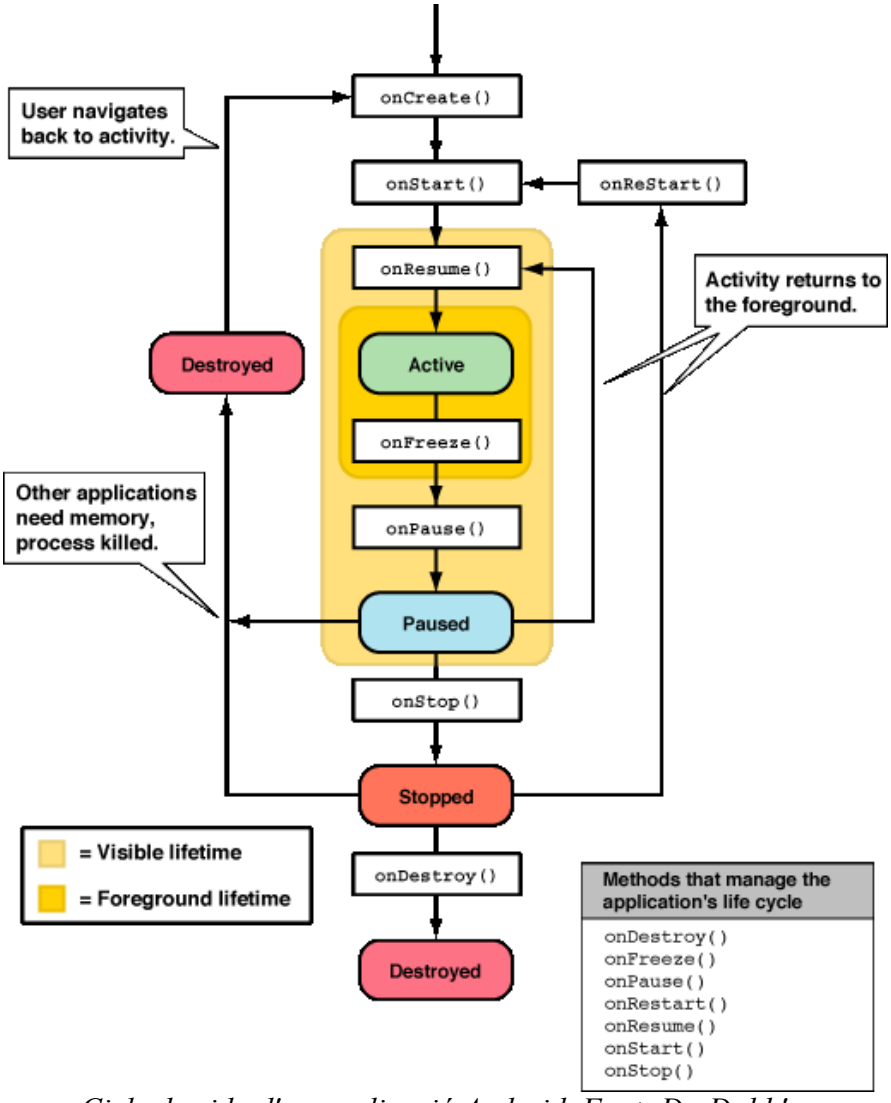
Aquesta prioritat ve representada per valors del 0 al 5 i, segons el codi font de l'Android [Xda10], té els valors següents, ordenats de major a menor prioritat:

- FOREGROUND_APP
- VISIBLE_APP
- SECONDARY_SERVER
- HIDDEN_APP

- CONTENT_PROVIDER
- EMPTY_APP

Això vol dir que els processos que tenen assignada la prioritat EMPTY_APP són els primers a ser eliminats quan el dispositiu necessita més memòria disponible.

En la il·lustració que es pot veure a continuació [Dob08] es veu quin és el cicle de vida d'una aplicació Android:




*Cicle de vida d'una aplicació Android. Font: Dr. Dobb's.*

En l'API de l'Android SDK tenim la classe Process, que és la que ens dóna accés d'alt nivell a

la gestió bàsica de processos des de la nostra aplicació. Aquesta classe té tres funcions principals:

1. Permet identificar els processos que estan en execució a través del seu nom o el seu GUID. Un cop tenim l'identificador únic del procés es pot donar ordres a aquest per tal de modificar el seu comportament.
2. Permet enviar senyals a un procés. No tenim tots els senyals definits a l'estàndard POSIX disponibles, sinó que només en tenim tres: SIGNAL_KILL, SIGNAL_QUIT i SIGNAL_USR1. O sigui que la finalitat és la de permetre'ns només tancar o matar el procés o bé enviar-hi un senyal personalitzat en cas que l'implementem. En aquest aspecte, la classe és bastant limitada.
3. Permet definir la prioritat del procés. Aquesta tercera funcionalitat és la més interessant de totes tres ja que existeixen diferents prioritats segons si el procés es troba en primer o segon pla i també depenent als recursos del sistema als quals accedeix. La llista de prioritats és la següent:
    - THREAD_PRIORITY_AUDIO
    - THREAD_PRIORITY_BACKGROUND
    - THREAD_PRIORITY_DEFAULT
    - THREAD_PRIORITY_DISPLAY
    - THREAD_PRIORITY_FOREGROUND
    - THREAD_PRIORITY_LESS_FAVORABLE
    - THREAD_PRIORITY_LOWEST
    - THREAD_PRIORITY_MORE_FAVORABLE
    - THREAD_PRIORITY_URGENT_AUDIO
    - THREAD_PRIORITY_URGENT_DISPLAY

Encara a més alt nivell, afegint una altra capa d'abstracció, l'Android SDK disposa de la classe anomenada ActivityManager que ens permet accedir a tota mena d'informació sobre l'activitat

(o sigui: l'aplicació) que s'està executant. I més concretament la classe anomenada ActivityManager.RunningAppProcessInfo que també ens ofereix informació sobre la prioritat del procés. Però la informació que trobem en aquesta classe és molt menys interessant per a l'aplicació que no pas la que ofereix la classe Process.

## *6.2. Gestió de memòria*

Com s'ha pogut veure a l'apartat anterior, la gestió de processos en els dispositius Android està molt lligada a la gestió de la memòria, ja que el cicle de vida dels processos està íntimament relacionat amb la memòria que es troba disponible en el dispositiu. Però la gestió de memòria en sí mateixa també té les seves peculiaritats que descriurem en aquesta secció.

Tal i com fan els altres sistemes que utilitzen una màquina virtual com Java o .NET, el sistema Android utilitza la seva màquina virtual en temps d'execució per a gestionar la memòria de les aplicacions. Però la diferencia important, com ja hem vist en l'apartat anterior, és que la màquina virtual també s'encarrega de gestionar el cicle de vida de les aplicacions. Les aplicacions amb menys prioritat s'aturen o s'eliminen per a donar recursos a les aplicacions que tenen una prioritat més elevada. I, de tots, el recurs més important és la memòria.

Tot i aquesta gestió automàtica de la memòria no tots els usuaris estan satisfets amb el seus resultats i n'hi ha molts que utilitzen aplicacions externes per a eliminar aplicacions que no utilitzen en comptes d'esperar que la màquina virtual ho faci automàticament ja que, en alguns casos, el rendiment del dispositiu en general i també la vida de la bateria es redueix de manera considerable [Mob10].

A l'Android SDK disposem de la classe Debug.MemoryInfo que ens permet accedir a la quantitat de memòria utilitzada per un procés determinat. Podem saber, per exemple, la quantitat de memòria nativa que utilitza i també la memòria assignada dins de la màquina virtual de Java. La quantitat de memòria està sempre mesurada en kB.

A més alt nivell, la classe ActivityManager.MemoryInfo ens ofereix la possibilitat d'accedir a la memòria total del sistema (però no de manera tan exacte, ja que no té en compte la memòria que utilitza el nucli) i també saber si el sistema es troba en situació de memòria baixa i a partir de quin llindar es considera que el sistema es troba en estat de memòria baixa

segons la quantitat de memòria que hi hagi disponible.

## *6.3. Ús de temps de CPU*

Per a l'aplicació que volem construir en aquest projecte, monitoritzar l'ús de temps de la CPU en tot moment és essencial. Malauradament, no hi ha cap manera directa a través de l'API de l'Android SDK de saber l'ús de CPU de cada procés o inclús la carrega d'ús de CPU del dispositiu en general.

La classe ActivityManager.getRunningAppProcesses que és l'encarregada de llistar els processos que s'estan executant en la màquina virtual de Java, no proporciona cap mena d'informació sobre l'ús de memòria o l'ús de CPU del procés. Si llegim el codi font de la plataforma Android, veurem que hi ha una manera de fer-ho, però aquesta funcionalitat és interna i no s'exporta a través de l'SDK perquè les aplicacions creades pels usuaris en puguin fer ús.

Una de les maneres d'aconseguir aquest tipus d'informació és recordar que sota la capa de l'Android tenim tota una plataforma Linux al nostra servei a la qual podem accedir a través de crides al sistema. Per exemple, podem parsejar el contingut del fitxer “proc/stat” per obtenir aquests valors. Una manera de fer-ho a través de l'API de Java és la següent:

```
RandomAccessFile reader = new RandomAccessFile("/proc/stat", "r");
String load = reader.readLine();
String[] toks = load.split(" ");
long idle1 = Long.parseLong(toks[5]);
 long cpu1 = Long.parseLong(toks[2]) + Long.parseLong(toks[3]) + Long.parseLong(toks[4])
          + Long.parseLong(toks[6]) + Long.parseLong(toks[7]) + Long.parseLong(toks[8]);
```

També ho podem fer tot parsejant la sortida de text d'algunes eines estàndard de Linux que es troben disponibles en el sistema com són “top” i “ps”. És d'aquesta manera com es farà en l'aplicació tal i com veurem a l'apartat dedicat a la implementació.

En tot cas, és una llàstima que la plataforma Android no disposi actualment d'una manera més elegant d'obtenir aquesta informació a través de l'Android SDK.

## *6.4. Estalvi d'energia*

Aquest és un apartat especialment important perquè l'estalvi d'energia i, en conseqüència, l'allargament de la durada de la bateria el màxim possible, és l'objectiu fonamental de l'aplicació. És evident que una de les funcions més importants és la de monitoritzar l'estat de la bateria. Si bé és veritat que conèixer l'estat general de la bateria és molt fàcil, es pot fer bastant complicat d'obtenir percentatges ben acurats de la càrrega del dispositiu. Per fer-ho, s'ha de recórrer, com en el cas anterior, directament al sistema Linux que és la base de la plataforma.

La classe BatteryManager del SDK de l'Android ens permet saber l'estat de la bateria. Podem obtenir informació sobre si el dispositiu s'està carregant o no, de l'estat físic de la bateria (per exemple: si la bateria està en bon estat o bé necessita ser substituïda) i també del nivell de càrrega de la mateixa. La classe disposa de la propietat EXTRA_LEVEL que és un enter que va del número 0 fins al valor màxim representat per la propietat EXTRA_SCALE.

A més baix nivell, i de manera més acurada, es pot parsejar el contingut dels fitxers que es troben en el directori del dispositiu anomenat "/sys/class/power_supply/battery" [Gre11]. En el fitxers "batt_vol" i "batt_temp" podem trobar els volts i la temperatura de la bateria, respectivament. I en el fitxer anomenat "uevent" podem trobar les constants amb els valors següents, que serveixen per fer una lectura encara més acurada de l'estat de la bateria:

- POWER_SUPPLY_NAME
- POWER_SUPPLY_TYPE
- POWER_SUPPLY_STATUS
- POWER_SUPPLY_HEALTH
- POWER_SUPPLY_PRESENT
- POWER_SUPPLY_TECHNOLOGY
- POWER_SUPPLY_CAPACITY

El següent aspecte a tenir en compte és la possibilitat de desactivar alguns serveis no

essencials per tal de mantenir el dispositiu en funcionament per intentar allargar l'estalvi de la bateria. Exemples de serveis que es poden desactivar programàticament i que ens interessen per a la nostra aplicacio són el gprs, la wifi, el Bluetooth.

El cas de la wifi és especialment interessant per a la nostra aplicació. En primer lloc, no té sentit tenir la wifi activada buscant xarxes quan ens trobem en una zona on no n'hi ha. Això l'únic que fa és disminuir considerablement el nivell de la bateria en una cerca inútil i continua d'aquestes xarxes. I fins i tot en el cas que tinguem xarxes wifi a la nostra disposició és interessant que l'usuari pugui decidir no utilitzar-les i accedir a Internet, com a alternativa, a través de GPRS o 3G. Alternativament, el servei wifi es pot voler desactivar només en el cas que l'estat de la bateria sigui molt baix per intentar manteniment el dispositiu funcionant el màxim de temps possible.

La desactivació del servei de wifi es pot fer a través de la classe WifiManager i el mètode setWifiEnabled de l'API de l'Android SDK. Aquest mètode està disponible des de la versió 1 de l'API amb la qual cosa ha de funcionar en tots els dispositius sigui quina sigui la versió d'Android que tinguin instal·lada.

El Bluetooth no és una característica que estigui disponible en tots els dispositius i, en molts casos, no està activat per defecte. Això fa que sigui un molt bon candidat per desactivar-lo en el cas que l'estat de la bateria sigui baixa o simplement es vulgui intentar allargar la durada de la bateria el màxim possible. Per tal de desactivar-lo, l'API de l'Android SDK disposa de la classe BluetoothAdapter que té un mètode que es diu disable que ens permet fer aquesta operació.

El cas del servei de telefonia és una mica més complicat que els altres, ja que en definitiva estem parlant d'un telèfon intel·ligent i, en teoria, la seva funció principal és permetre la realització i la recepció de trucades. Tot i això l'Android SDK disposa de la possibilitat de deshabilitar aquest servei. La classe ServiceState dins de android.telephony exporta el mètode setStateOff que ens permet desactivar la telefonia del dispositiu.

# 7. Disseny de l'aplicació

En aquesta secció es parlaran de moltes coses, totes elles relacionades amb el disseny de l'aplicació. En primer lloc es farà una enumeració de les funcionalitats i després veurem la seva especificació i, finalment ens centrarem en un punt tan important com és l'arquitectura del sistema.

## *7.1. Descripció de les funcionalitats*

En aquest punt farem una descripció funcional de l'aplicació des del punt de vista de l'usuari, sense entre en detalls tècnics sobre la mateixa. Aquesta secció servirà, doncs, d'introducció a la resta de seccions.

1. Permet estalviar bateria del dispositiu, utilitzant diferents tècniques. En primer lloc tindrem un sistema de mesura el més precís possible que ens indicarà en tot moment quin és l'estat de la bateria. El servei s'encarregarà d'anar desactivant funcionalitats del dispositiu quan aquests no s'utilitzin o no quedi molta bateria, per intentar que la durada de la bateria del telèfon mòbil sigui com més gran millor. En alguns casos, fins i tot baixant la freqüència del rellotge de la CPU, tot i que això es depenent del dispositiu. Si el dispositiu ho permet, serà possible implementar-ho.
2. Mostra i permet ordenar en una llista els processos que s'estan executant al sistema. Es mostrarà el nom del procés, temps de CPU, temps total, ús de memòria i altres valors addicionals. En la finestra on mostrarem els processos també es podran buscar per nom i filtrar segons unes característiques determinades. Són especialment interessants les possibilitats d'ordenar els processos per ús de CPU o memòria, per exemple.
3. A part de la informació de cada procés, l'aplicació ens ha de mostrar també les dades globals del sistema a títol informatiu com són: ús de CPU, memòria, etc. La informació sobre l'estat de la bateria es pot mostrar també aquí o també es pot fer només en una finestra a part especialitzada.
4. El servei, que funciona en segon pla, monitoritza l'estat de la bateria i els recursos generals del sistema i també que no hi hagi cap aplicació que es quedi "penjada" o utilitzi la majoria de recursos del dispositiu. Si és així, es matarà el procés. En altres

casos com que hi hagi bateria baixa, el procés serà també l'encarregat d'efectuar les gestions necessàries per tal d'intentar maximitzar la durada de la bateria.

Un aspecte especialment destacable de la nostra aplicació serà les seves grans possibilitats de configuració i personalització. Si bé, hi haurà uns valors per defecte raonables i que haurien de ser vàlids per a la majoria dels usuaris, els paràmetres es podran ajustar segons les necessitats de cadascú. Serà possible, fins i tot, crear perfils que no seran més que una agrupació de paràmetres de diferents categories. Aquests perfils podrien ser, per exemple: “maximitzar bateria” o bé “maximitzar rendiment” amb diferents valors intermedis.

L'aplicació ens permetrà configurar els elements següents:

5. El primer és configurar les opcions d'estalvi de bateria. En aquest cas podem decidir de quins serveis prescindim segons l'estat de càrrega de la bateria. La pantalla de configuració ens ofereix un llistat de serveis i un control que ens permet seleccionar-ho.
6. Es poden configurar uns límits dels usos dels recursos per procés. Si el procés utilitza més recursos dels assignats, s'enviarà una Es pot també voler que el servei ignori uns processos determinats encara que utilitzin molts recursos.
7. També es pot configurar aquests mateixos paràmetres a nivell global, però d'una manera molt més genèrica i que no depenguin de cap servei en concret, sinó de l'estat general del dispositiu.
8. En últim lloc, tenim la finestra per iniciar i parar el servei que s'executa en segon pla. Aquesta finestra ens permetrà fer-ho de forma manual. En el moment d'instal·lar la nostra aplicació, el servei es registrarà automàticament per iniciar-se cada vegada que s'engegui el dispositiu. Cal recordar que el servei és la part que realitza la monitorització dels recursos i que si no es troba funcionant, la nostra aplicació perd la major part de la seva funcionalitat.

## *7.2. Requeriments*

El programa està dividit entre un client i un servidor, tot seguint l'arquitectura més adequada a les necessitats de l'aplicació i, al mateix temps a les restriccions que ofereix la plataforma i l'SDK de l'Android, ja que estem treballant a un nivell més baix que la majoria de les altres

aplicacions d'usuari i, per tant, hem de seguir les guies i directrius que ens proporciona el sistema. Les premisses generals de funcionament de l'aplicació són les següents:

- Control sobre les accions del servidor.
- El client ha de poder rebre les notificacions del servidor.
- Control de seguretat per permetre que només el client pugui accedir al servidor i a les seves notificacions.
- Control d'accés a la base de dades tant del client com del servidor.

Tal i com ja hem comentat en la descripció de les funcionalitats, el servidor ha de ser el més lleuger possible perquè estarà funcionant en el dispositiu en tot moment, mentre que el client serà l'encarregat de realitzar totes les tasques importants a més a més de proporcionar la interfície d'usuari perquè els actors del sistema es puguin relacionar amb l'aplicació. S'ha dividit els requeriments en aquests dos programes i els tractarem per separat:

Requeriments del client

- Autenticació: el client s'ha d'autenticar amb el servidor per tal de poder accedir-hi.
- La comunicació amb el servidor es fa a través de d'alertes i missatges del sistema que ja ens proporciona la plataforma Android.
- El client també ha de poder accedir a la base de dades on tant el servidor i el client guarden informació de configuració sense que hi hagi conflictes entre tots dos programes accedint a les mateixes dades.

Requeriments del servidor

- Autenticació: el client s'ha d'autenticar amb el servidor per tal de poder accedir-hi.
- La comunicació amb el client es fa a través de d'alertes i missatges del sistema.
- El servidor llegeix i escriu valors en una base de dades que només ha de ser accessible al client i al servidor.

## 7.3. Especificació

L'especificació es farà en UML amb la creació d'actors i de casos d'ús que veurem a continuació. El cas dels actors es podria resoldre amb un sol actor, que és l'usuari que instal·la i fa servir l'aplicació en el seu telèfon intel·ligent. Però és també interessant dividir les tasques entre un usuari normal i un usuari més avançat ja que són realment perfils d'usuaris diferents tot i que una mateixa persona pot intercanviar-se els rols segons les seves necessitats en moments determinats.

### 7.3.1. Actors

A l'hora de definir els actors, s'ha especialitzat l'usuari en dos rols diferents. No es tracta d'un problema de permisos, ja que els dos tipus d'usuaris poden fer de tot, sinó que ens serveix per fer una classificació de les tasques que poden fer els usuaris segons el seu nivell de coneixements o necessitats d'ús en un moment determinat.

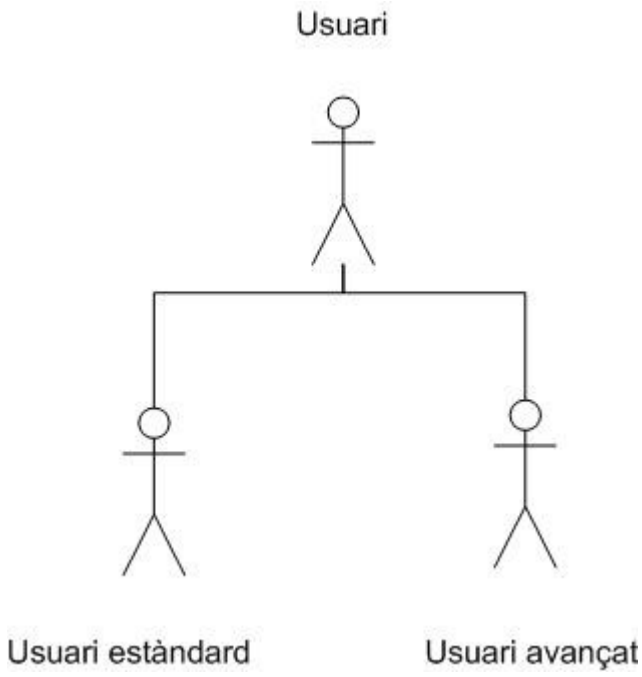
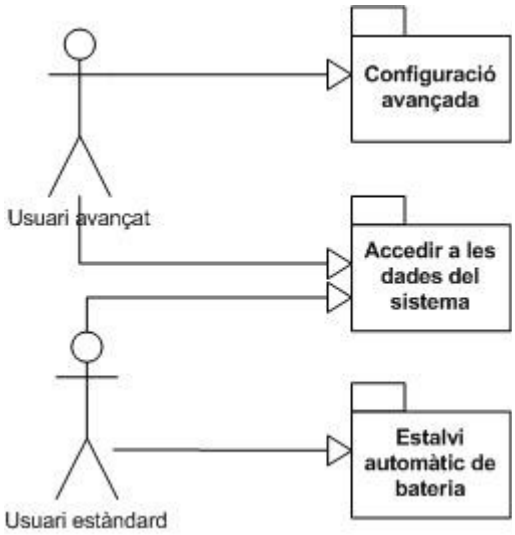



*Diagrama d'actors*

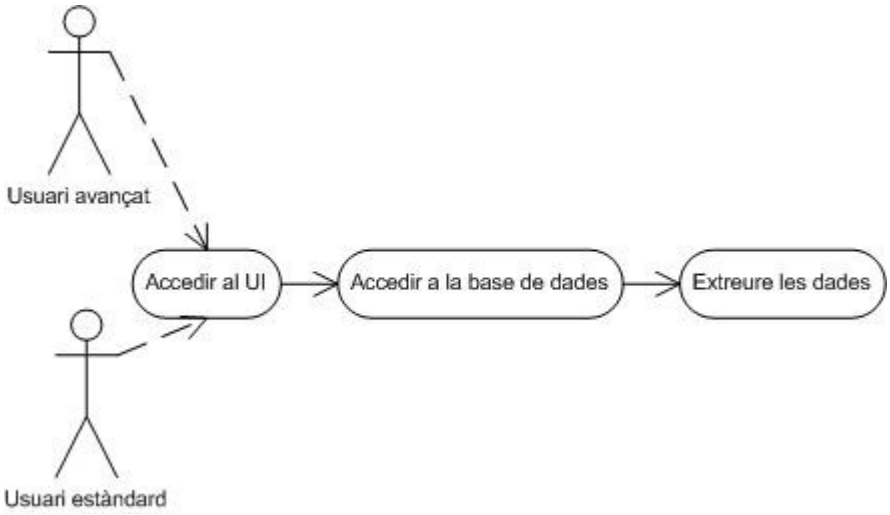
Les tasques que faria cada un d'aquests usuaris serien les següents:




*Classificació dels casos d'ús*

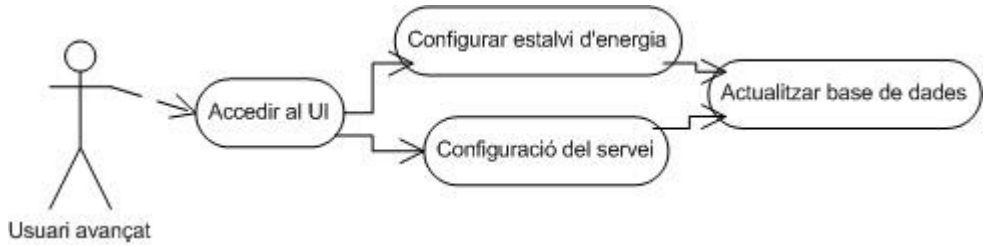
### *7.3.2. Diagrama de casos d'ús*

En els següents diagrames representarem els casos d'ús de la nostra aplicació:




*Accedir a les dades del sistema*


*Estalvi automàtic de bateria*




*Configuració avançada*

## *7.3.3. Especificació dels casos d'ús*

En aquest apartat especificarem els casos d'ús de l'aplicació dividits entre els dos tipus d'usuaris que hem mencionat en els apartats anteriors.

Usuari estàndard

**Accedir al UI**

| | |
|---|---|
| Resum de la funcionalitat: | Inici de l'accés al sistema |
| Actors: | Usuari estàndard, usuari avançat |
| Casos d'us relacionats: | Accedir a la base de dades |
| Precondició: | L'aplicació ha d'estar instal·lada |
| Resum: | Inicia l'interacció de l'usuari amb l'aplicació |

**Accedir a la base de dades**

| | |
|---|---|
| Resum de la funcionalitat: | L'aplicació accedeix a la base de dades |
| Actors: | Usuari estàndard, usuari avançat |
| Casos d'us relacionats: | Accedir al UI |
| Precondició: | El servei ha d'estar funcionant per tal de recollir dades |
| Resum: | L'accés a la base de dades ens permet accedir a la informació |

**Extreure les dades de la base de dades**

| | |
|---|---|
| Resum de la funcionalitat: | Obtenir les dades de la base de dades de manera visual |
| Actors: | Usuari estàndard, usuari avançat |
| Casos d'us relacionats: | Accedir a la base de dades |
| Precondició: | El servei ha d'estar funcionant i s'ha d'haver accedit a la base de dades |
| Resum: | S'obtenen les dades de funcionament del sistema |

**Activar el servei**

| | |
|---|---|
| Resum de la funcionalitat: | Activa el servidor |
| Actors: | Usuari estàndard, usuari avançat |
| Casos d'us relacionats: | Accedir al UI |
| Precondició: | L'aplicació i la base de dades han d'estar funcionant |
| Resum: | Activa el servidor que recull les dades |

**Escollir la configuració per defecte**

| | |
|---|---|
| Resum de la funcionalitat: | L'usuari selecciona la configuració per defecte |
| Actors: | Usuari estàndard, usuari avançat |
| Casos d'us relacionats: | Accedir al UI, Activar el servei |
| Precondició: | L'aplicació i el servei han d'estar funcionant |
| Resum: | El sistema s'executa amb els paràmetres preconfigurats. |

Usuari avançat

**Accedir al UI**

| | |
|---|---|
| Resum de la funcionalitat: | Inici de l'accés al sistema en mode avançat |
| Actors: | Usuari avançat |
| Casos d'us relacionats: | Accedir a la base de dades |
| Precondició: | L'aplicació ha d'estar instal·lada |
| Resum: | Inicia l'interacció de l'usuari amb l'aplicació |

**Configurar el servei**

| | |
|---|---|
| Resum de la funcionalitat: | L'usuari pot configurar el servei |
| Actors: | Usuari avançat |
| Casos d'us relacionats: | Accedir al UI |
| Precondició: | Servei funcionant |
| Resum: | Configuració del servei a nivell avançat |

**Configurar l'estalvi d'energia**

| | |
|---|---|
| Resum de la funcionalitat: | L'usuari pot configurar l'estalvi d'energia |
| Actors: | Usuari avançat |
| Casos d'us relacionats: | Accedir al UI, Configurar el servei |
| Precondició: | Servei funcionant |
| Resum: | Configurar l'estalvi d'energia a nivell avançat |

**Actualitzar la base de dades**

| | |
|---|---|
| Resum de la funcionalitat: | Actualització de la base de dades |
| Actors: | Usuari avançat |
| Casos d'us relacionats: | Accedir al UI, Configurar el servei, Accedir a la base de dades |
| Precondició: | Servei executant-se i configurat |
| Resum: | S'actualitza la base de dades amb les noves dades de configuració |

## *7.4. Arquitectura*

L'aplicació utilitza, a nivell general, dos components perfectament diferenciats: un servei i un client. Aquesta decisió està clarament influenciada també per les possibilitats i les restriccions de la plataforma Android a l'hora de realitzar aplicacions d'aquesta mena.

En primer lloc, el servei s'executa en segon pla tota l'estona i s'encarrega de monitoritzar l'estat del dispositiu. Un requeriment d'aquest servei és que, per les seves característiques, ha de consumir el menor nombre de recursos possible. Això implica que el servei ha de realitzar una funció molt bàsica i delegar la resta de funcionalitats al client, que només s'executa quan l'usuari ho demana directament o bé quan rep una notificació del servei. El servei, doncs, no té interfície d'usuari ni cap element que pugui interactuar amb l'usuari. L'usuari només pot decidir iniciar o aturar el servei des del client.

El servei és un bucle que s'encarrega d'obtenir cada segon (tot i que aquest paràmetre és configurable) l'estat del sistema a través de crides directes al nucli de Linux i altres components de baix nivell del sistema operatiu. Aquestes dades es poden enviar en temps real al client, si aquest s'està executant. Per fer-ho, i utilitzant la terminologia pròpia de l'Android, el client s'ha de subscriure com a Listener. També, en el cas que es produeixin unes condicions determinades prèviament configurades (per exemple, un canvi d'estat en la bateria), el servei envia una notificació d'aquest canvi d'estat al client.

Aquests paràmetres es llegeixen d'una base de dades i es carreguen en memòria. El servei ha de rebre una crida especial del client que li indica que ha de rellegir aquestes dades. Si aquesta crida no es produeix, el servei continua amb les dades anteriors ja que no les actualitza de manera automàtica. El servei només realitza algunes funcions determinades quan el client li demana explícitament, per estalviar recursos del dispositiu i contribuir el menys possible a

l'exhauriment de la bateria.

El segon component de l'arquitectura és el client. El client té la doble funció de ser la interfície gràfica sobre la que actua l'usuari (tot mostrant dades i permetent la configuració de certs paràmetres) i també, al mateix temps, el responsable de dur a terme les accions corresponents quan el servei li comunica algun canvi d'estat.

Un tercer component del sistema és la base de dades. Tant el client i el servidor hi accedeixen per tal de llegir (el servidor) o escriure (el client) les dades configuració. Android té suport per a bases de dades SQLite i aquestes serà la que farem servei per a la nostra aplicació.

### *7.4.1. Diagrames de seqüència*

En aquest apartat especificarem alguns diagrames de seqüència de les funcionalitats que hem detectat en els apartats anteriors, per tal d'il·lustrar encara amb més detall el funcionament de l'aplicació.

**Accedir al UI**

En aquest primer diagrama podem veure com es produeix l'accés al UI. Tot i que l'usuari no s'ha d'identificar en el UI per tal d'accedir al servei, es produeix una identificació interna a través del sistema que ens proporciona la plataforma Android per defecte per tal d'assegurar que només el client es pot connectar en el servidor.

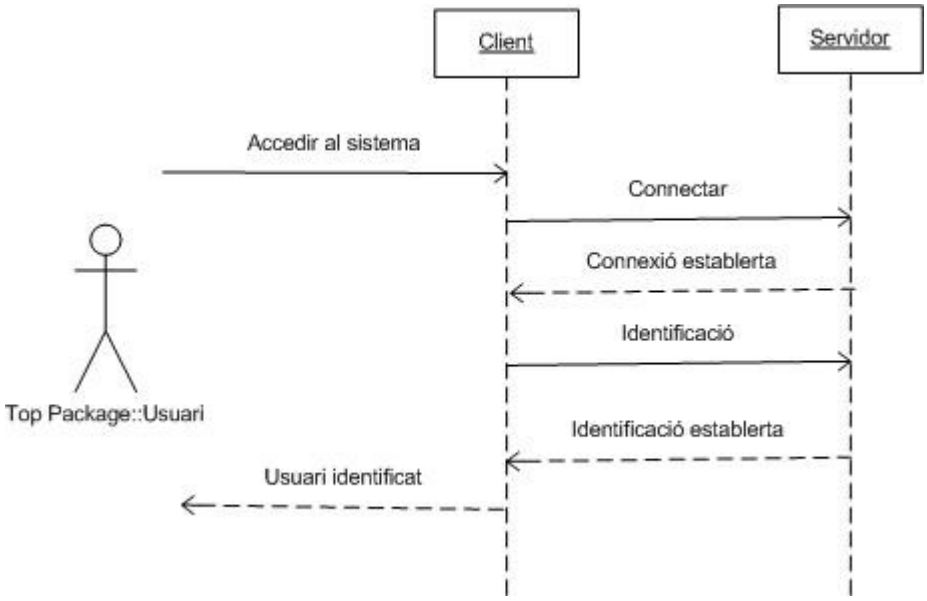



*Diagrama de seqüència d'accés al UI*

**Extreure les dades de la base de dades**

El cas d'ús més comú de l'aplicació és comprovar quin és l'estat del sistema, per saber si tot està funcionant correctament. Per fer-ho, l'usuari només ha d'accedir al UI per obtenir aquesta informació.

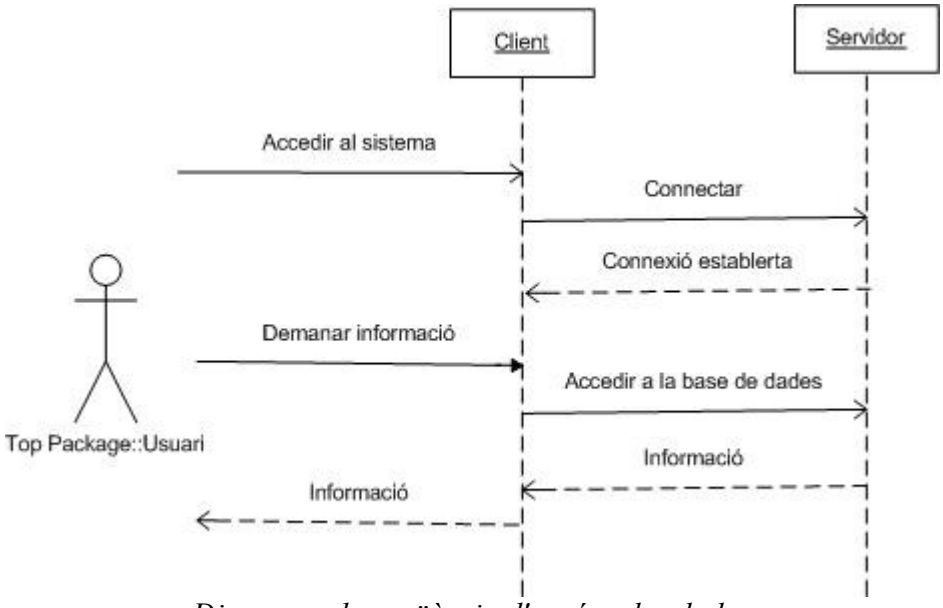



*Diagrama de seqüència d'accés a les dades*

**Activar / Desactivar el servei**

Activar o desactivar el servei permet a l'usuari configurar l'aplicació perquè simplement sigui una eina de monitorització del sistema o bé també s'encarregui de realitzar accions en el cas que es donin unes característiques determinades, com per exemple, que la bateria del sistema es trobi en un estat molt baix.

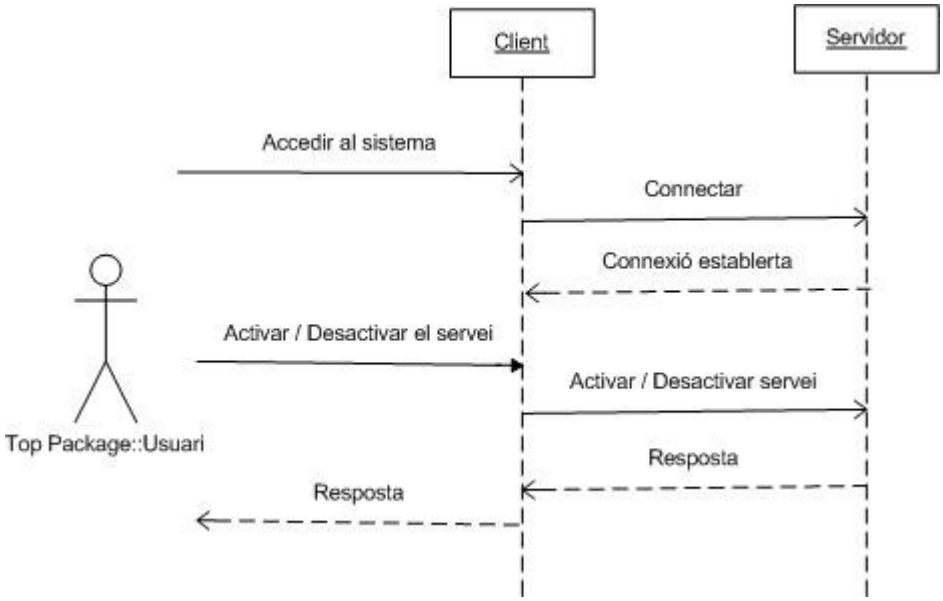



*Diagrama de seqüència d'activació del servei*

**Configurar el servei**

La configuració del servei permet establir quins són els paràmetres sobre els quals l'aplicació establirà les seves decisions i actuarà en conseqüència en el cas que les condicions especificades es compleixin.

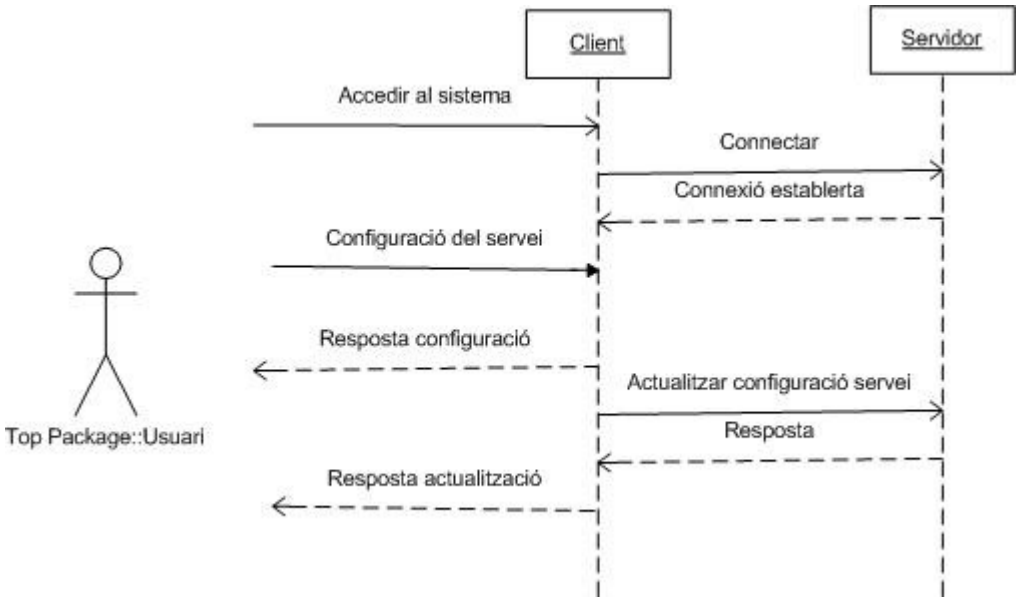



*Diagrama de la configuració del servei*

### *7.4.2. Classes*

En aquest apartat veurem quines són les classes que formen el projecte, com estan relacionades entre elles i també una breu descripció funcional dels seus mètodes. Això permetrà fer-se una idea molt clara de quin és el funcionament de l'aplicació i com es relacionen les classes entre elles per tal de construir el programa i de quina manera s'han implementat.

En primer lloc, es veurà una descripció a molt alt nivell de les classes de l'aplicació, i les seves relacions, representades en el diagrama següent:

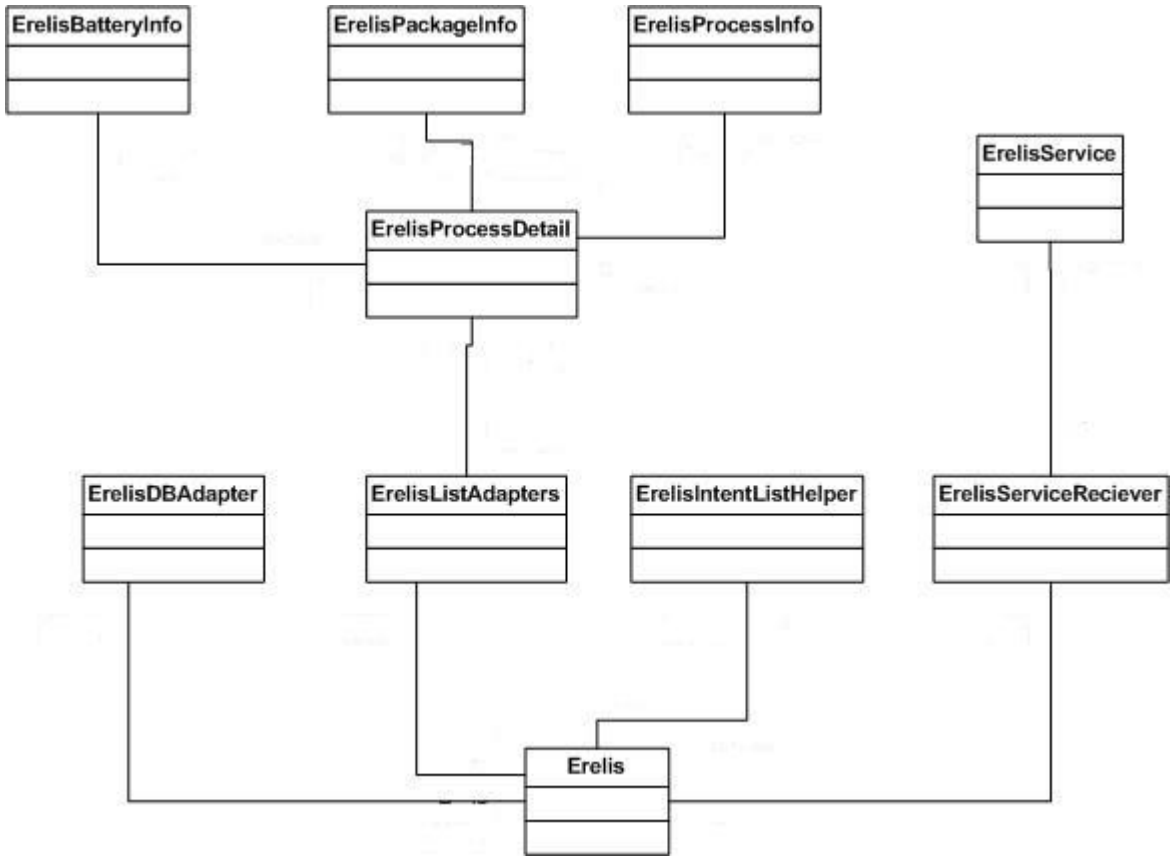
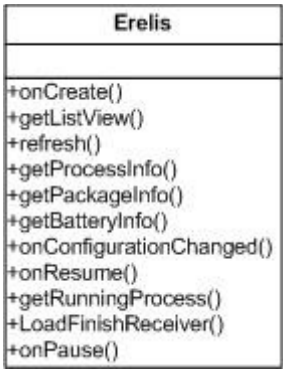



*Diagrama de les classes de l'aplicació*

Un cop s'han vist les classes que formen l'aplicació, es veurà quins són els membres que implementen cada una d'elles:

**Erelis**

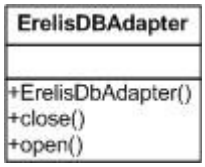
Aquesta és la classe principal i el punt d'entrada de l'aplicació. S'encarrega d'inicialitzar el UI i els diferents serveis.



| | |
|---|---|
| onCreateBundle() | Punt d'entrada de l'aplicació |
| getListView() | Obtenir els ListViews del programa |
| refresh() | Es crida quan l'aplicació ha d'actualitzar el UI |
| getProcessInfo() | Obtenir informació dels processos |
| getPackageInfo() | Obtenir informació dels paquets |
| getBatteryInfo() | Obtenir informació de l'estat de la bateria |
| onConfigurationChange(Configuration) | Callback que es crida quan hi ha canvis en la configuració del programa |
| onResume() | El sistema posa l'aplicació en mode continuar |
| getRunningProcess() | Obtenir els processos que s'estan executant |
| LoadFinishReciever() | Callback que es crida després de rebre un Intent. |
| onPause() | El sistema posa l'aplicació en mode pausa |

**ErelisDBAdapter**

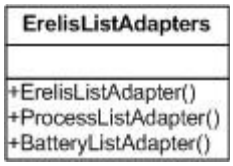
Aquesta classe gestiona l'accés a la base de dades.




| | |
|---|---|
| ErelisDbAdapter | Inicialitza l'accés a les dades de la base de dades |
| close() | Obre la connexió amb la base de dades |
| open() | Tanca la conneixió amb la base de dades |

**ErelisListAdapters**

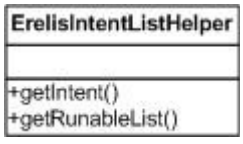
Aquesta classe implementa els diferents adaptadors de llista que ens serveixen per implementar el UI de l'aplicació.




| | |
|---|---|
| ErelisListAdapter(ErelisManager, List<RunningTaskInfo>) | Constructor de la classe |
| getCount() | Retorna el nombre d'elements |
| getItem(int) | Retorna l'element |
| getItemId(int) | Retorna l'identificador de l'element |
| getView(int, View, ViewGroup) | Retorna la vista |
| ProcessListAdapter(ErelisManager, ArrayList<DetailProcess>) | Constructor de la classe |
| getCount() | Retorna el nombre d'elements |
| getItem(int) | Retorna l'element |
| getItemId(int) | Retorna l'identificador de l'element |
| getView(int, View, ViewGroup) | Retorna la vista |
| BatteryListAdapter(ErelisManager, List<BatteryStatsInfo>) | Constructor de la classe |
| getCount() | Retorna el nombre d'elements |
| getItem(int) | Retorna l'element |
| getItemId(int) | Retorna l'identificador de l'element |
| getView(int, View, ViewGroup) | Retorna la vista |

**ErelisIntentListHelper**

Aquesta classe implementa el menú de context quan es fa clic sobre algun dels elements de les llistes del UI.

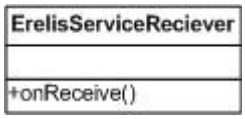



| | |
|---|---|
| getIntent(String, PackageManager) | Obté la classe Intent de l'opció determinada |

| | |
|---|---|
| getRunableList(PackageManager, boolean) | Obté la llista de UI |

**ErelisServiceReciever**

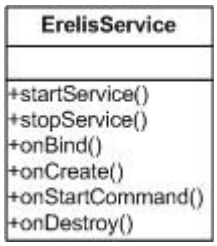
Aquesta classe rep les notificacions que envia el servei a l'aplicació.




| | |
|---|---|
| onReceive(Context, Intent) | Callback que obté les crides que envia el servidor |

**ErelisService**

La classe que implementa el servei que s'executa en segon pla i que monitoritza l'estat del sistema.




| | |
|---|---|
| startService() | Mètode que inicia el servei |
| stopService() | Mètode que atura el servei |
| onBind(Intent) | Callback que es crida con el servei està incialitzat |
| onCreate() | Callback que es el punt d'entrada del servei |
| onStartCommand(Intent, int, int) | Callback que es crida quan el servei rep una petició d'actualització |
| onDestroy() | Callback que es crida quan el servei està a punt de ser eliminat |

**ErelisProcessDetail**

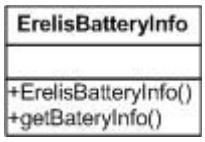
Aquesta classe recull tota la informació que obtenim d'un procés a través de les altres classes.


| | |
|---|---|
| ErelisProcessDetail(Context, RunningAppProcessInfo) | Constructor de la classe |
| compareTo(Object) | Mètode usat per a comparar dos processos |
| fetchApplicationInfo(ErelisPackageInfo) | Demanar a la classe ErelisPackageInfo que actualitzi la informació de les aplicacions |
| fetchPackageInfo() | Demanar informació actualitzada dels paquets |
| fetchPsRow(ErelisProcessInfo) | Demanar informació actualitzada sobre els processos en funcionament |
| getAppinfo() | Obtenir informació sobre l'aplicació |
| getBaseActivity() | Obtenir informació sobre l'activitat |
| getIntent() | Obtenir la classe Intent |
| getPackageName() | Obtenir informació sobre el nom del paquet |
| getPkginfo() | Obtenir informació sobre els paquets |
| getPsrow() | Obtenir informació sobre els processos que s'estan executant en el dispostiu |
| getRuninfo() | Obtenir informació sobre els programes en funcionament |
| getTitle() | Obtenir el títol de l'aplicació |
| isGoodProcess() | Mètode que ens informa si el procés està considerat que té un bon comportament |

| | |
|---|---|
| setAppinfo(ApplicationInfo) | Establir la informació de l'aplicació |
| setPkginfo(PackageInfo) | Establir la informació del paquet |
| setPsrow(ProcessInfoRow) | Establir la informació sobre la llista de processos |
| setRuninfo(RunningAppProcessInfo) | Establir la informació sobre les applicacions en funcionament |

**ErelisBatteryInfo**

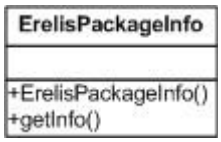
Aquesta classe s'utilitza per a obtenir informació sobre l'estat de la bateria.




| | |
|---|---|
| ErelisBatteryInfo() | Constructor de la classe |
| getBateryInfo() | Retorna la informació sobre l'estat de la bateria |

**ErelisPackageInfo**

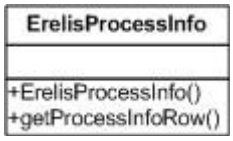
Aquesta classe obté informació sobre les aplicacions instal·lades.




| | |
|---|---|
| ErelisPackageInfo(Context) | Constructor de la classe |
| getInfo(String) | Retorna la informació sobre un paquet determinat |

**ErelisProcessInfo**

La classe s'encarrega d'obtenir informació dels processos que s'executen al sistema.




| | |
|---|---|
| ErelisProcessInfo() | Constructor de la classe |
| getProcessInfoRow(String) | Retorna la informació sobre un procés determinat |

# 8. Implementació

En aquest apartat es veurà com s'han implementat les funcionalitats descrites a l'apartat anterior a nivell de codi de l'aplicació. El codi s'ha escrit en el llenguatge de programació Java i per sobre de la plataforma l'Android SDK, que ens ofereix una sèrie d'ajudes molt importants, però també algunes restriccions, en la manera de fer les coses.

En aquesta implementació s'ha intentat seguir en tot moment les bones pràctiques de disseny i les recomanacions de la documentació, i dels exemples, de la plataforma. D'aquesta manera ens podem assegurar, per una banda, un bon funcionament de l'aplicació i també que aquesta serà compatible en versions posteriors a la versió 11.

## *8.1. Accés a les dades dels processos*

Per accedir a les dades dels processos, s'utilitza la classe anomenada ErelisProcessInfo. El funcionament de la classe és molt senzill. En el moment d'instanciar-la llença la comanda "ps" del sistema Linux i llavors captura la sortida del programa en una classe anomenada ProcessInfoRow, que conté la informació de cada procés. Llavors existeix un mètode public anomenat getProcessInfoRow que ens permet accedir a aquesta informació des d'una altra classe.

El codi font de la classe és el següent:

```
public class ErelisProcessInfo {
        public static class ProcessInfoRow {
                String pid = null;
                String cmd;
                String ppid;
                String user;
                int mem;

                public ProcessInfoRow(String line) {
                        if (line == null)
                                return;
                        String[] p = line.split("[\\s]+");
```

```
                if (p.length != 9)
                        return;
                user = p[0];
                pid = p[1];
                ppid = p[2];
                cmd = p[8];
                mem = Integer.parseInt(p[4]);
                if (isRoot()) {
                        rootpid = pid;
                }
        }

        public boolean isRoot() {
                return "zygote".equals(cmd);
        }

}
private ArrayList<ProcessInfoRow> pslist;

@SuppressWarnings("unused")
private static String rootpid = null;

public ErelisProcessInfo() {
        ps();
}

public ProcessInfoRow getProcessInfoRow(String cmd) {
        for (ProcessInfoRow row : pslist) {
                if (cmd.equals(row.cmd)) {
                        return row;
                }
        }
        return null;
}
```

```
	private void ps() {
		String ps = null;
		BufferedReader in = null;
		try {
			Process process = null;
			process = Runtime.getRuntime().exec("ps");
			in = new BufferedReader(new InputStreamReader(
					process.getInputStream()));
			ps = in.toString();

		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} finally {
			if (in != null)
				try {
					in.close();
				} catch (IOException e) {
					// TODO Auto-generated catch block
					e.printStackTrace();
				}
		}

		String[] lines = ps.split("\n");
		pslist = new ArrayList<ProcessInfoRow>();
		for (String line : lines) {
			ProcessInfoRow row = new ProcessInfoRow(line);
			if (row.pid != null)
				pslist.add(row);
		}
	}
}
```

## 8.2. Accés a la informació dels paquets

L'aplicació he d'accedir a la informació dels paquets que ens ofereix la plataforma Android SDK per tal de mostrar informació sobre el programa que s'està executant. Per tal de fer-ho, s'accedeix a la classe ApplicationInfo que proporciona informació sobre el nom del procés que correspon a una aplicació determinada, que la plataforma Android anomena paquet.

Per tal d'accedir a la informació de la mida dels paquets, que no forma part de l'API pública de l'Android SDK, hem d'utilitzar l'AIDL que és l'Android IDL per a la comunicació entre processos (IPC). Si l'entorn de desenvolupament és l'Eclipse, com és en el cas d'aquest projecte, instal·lar els fitxers AIDL és molt simple i es pot fer seguint els passos següents:

1. Has de descarregar els fitxers PackageStats i IpackageStatsObserver del codi font de l'Android.
2. A través de l'Eclipse, s'ha de crear un nou paquet Java i anomenar-lo: android.content.pm.
3. Ara s'han de copiar o bé importar els fitxers en el paquet que s'acaba de crear en el pas anterior.
4. El plugin ADT de Google genera automàticament la interfície IpackageStatsObserver en el directori gen.

La classe que implementa la funcionalitat d'obtenir la mida dels paquets és la següent:

```
public class ErelisPackageInfo {
        public static PackageInfo getPackageInfo(PackageManager pm, String name) {
                PackageInfo ret = null;
                try {
                        ret = pm.getPackageInfo(name, PackageManager.GET_ACTIVITIES);
                } catch (NameNotFoundException e) {
                }
                return ret;
        }

        private List<ApplicationInfo> appList;
```

```
    public ErelisPackageInfo(Context ctx) {
        PackageManager pm = ctx.getApplicationContext().getPackageManager();
        appList = pm.getInstalledApplications(
                PackageManager.GET_UNINSTALLED_PACKAGES);
    }
    public ApplicationInfo getInfo(String name) {
        if (name == null) {
            return null;
        }
        for (ApplicationInfo appinfo : appList) {
            if (name.equals(appinfo.processName)) {
                return appinfo;
            }
        }
        return null;
    }
}
```

## 8.3. Informació guardada dels processos i paquets

L'obtenció de les dades dels processos i paquets s'obté a través de les classes que s'han mencionat anteriorment, però aquestes dades s'han de combinar per tal d'oferir la informació adient a l'usuari. La classe ErelisProcessDetail és la classe que s'encarrega de fer-ho possible. Les altres classes de l'aplicació accedeixen a aquesta classe per tal d'obtenir la informació sobre els processos.

El codi de la classe que ho implementa és el següent:

```
public class ErelisProcessDetail implements Comparable<Object> {
    static class PkgSizeObserver extends IPackageStatsObserver.Stub {
        public int idx;

        public void onGetStatsCompleted(PackageStats pStats, boolean succeeded) {
            // pStats.cacheSize;
        }
    }

    private ErelisProcessInfo.ProcessInfoRow psrow = null;
    private ApplicationInfo appinfo = null;
```

```
private PackageInfo pkginfo = null;
private ActivityManager.RunningAppProcessInfo runinfo = null;
private String title = null;
private PackageManager pm;
private Intent intent = null;

public ErelisProcessDetail(Context ctx, ActivityManager.RunningAppProcessInfo runinfo) {
        this.runinfo = runinfo;
        pm = ctx.getApplicationContext().getPackageManager();
}

public int compareTo(Object another) {
        if (another instanceof ErelisProcessDetail && another != null) {
                return this.getTitle().compareTo(
                                ((ErelisProcessDetail) another).getTitle());
        }
        return -1;
}

public void fetchApplicationInfo(ErelisPackageInfo pkg) {
        if (appinfo == null)
                appinfo = pkg.getInfo(runinfo.processName);
}

public void fetchPackageInfo() {
        if (pkginfo == null && appinfo != null)
                pkginfo = ErelisPackageInfo.getPackageInfo(pm, appinfo.packageName);
}

public void fetchPsRow(ErelisProcessInfo pi) {
        if (psrow == null)
                psrow = pi.getProcessInfoRow(runinfo.processName);
}

public ApplicationInfo getAppinfo() {
        return appinfo;
}

public String getBaseActivity() {
        return pkginfo.activities[0].name;
}

public Intent getIntent() {
        if (intent != null)
                return intent;
        intent = null;
        intent = pm.getLaunchIntentForPackage(pkginfo.packageName);
        if (intent != null) {
                intent = intent.cloneFilter();
                intent.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
```

```
                return intent;
        }
        if (pkginfo.activities.length == 1) {
                intent = new Intent(Intent.ACTION_MAIN);
                intent.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
                intent.setClassName(pkginfo.packageName, pkginfo.activities[0].name);
                return intent;
        }
        intent = ErelisIntentListHelper.getIntent(pkginfo.packageName, pm);
        if (intent != null) {
                intent.addFlags(Intent.FLAG_ACTIVITY_BROUGHT_TO_FRONT);
                return intent;
        }
        return null;
}

public String getPackageName() {
        return appinfo.packageName;
}

public PackageInfo getPkginfo() {
        return pkginfo;
}

public ErelisProcessInfo.ProcessInfoRow getPsrow() {
        return psrow;
}

public ActivityManager.RunningAppProcessInfo getRuninfo() {
        return runinfo;
}

public String getTitle() {
        if (title == null)
                title = appinfo.loadLabel(pm).toString();
        return title;
}

public boolean isGoodProcess() {
        return runinfo != null && appinfo != null && pkginfo != null
        && pkginfo.activities != null
        && (pkginfo.activities.length > 0);
}

public void setAppinfo(ApplicationInfo appinfo) {
        this.appinfo = appinfo;
}

public void setPkginfo(PackageInfo pkginfo) {
        this.pkginfo = pkginfo;
```

```
	}

	public void setPsrow(ErelisProcessInfo.ProcessInfoRow psrow) {
		this.psrow = psrow;
	}

	public void setRuninfo(ActivityManager.RunningAppProcessInfo runinfo) {
		this.runinfo = runinfo;
	}
}
```

## 8.4. Accés a l'estat de la bateria

L'altre element imprescindible que l'aplicació ha de controlar és l'estat de la bateria en tot moment. Això es fa a través de la classe ErelisBatteryInfo que està implementada de manera molt similar a la classe ErelisProcessInfo. El servidor fa ús d'aquesta classe per tal de controlar l'status de la bateria en tot moment i enviar les notificacions adients a la resta de l'aplicació.

El codi de la classe que implementa l'accés a l'estat de la bateria és el següent:

```
public class ErelisBatteryInfo {

	private String batterLevel;
	private Context _ctx;

	public ErelisBatteryInfo(Context ctx) {
		_ctx = ctx;
	}

	private void batteryLevel() {
		BroadcastReceiver batteryLevelReceiver = new BroadcastReceiver() {
			public void onReceive(Context context, Intent intent) {
				context.unregisterReceiver(this);
				int rawlevel = intent.getIntExtra(BatteryManager.EXTRA_LEVEL,
						-1);
				int scale = intent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
				int level = -1;
				if (rawlevel >= 0 && scale > 0) {
					level = (rawlevel * 100) / scale;
				}
				batterLevel = "Battery Level Remaining: " + level + "%";
			}
		};
		IntentFilter batteryLevelFilter = new IntentFilter(
				Intent.ACTION_BATTERY_CHANGED);
```

```
                _ctx.registerReceiver(batteryLevelReceiver, batteryLevelFilter);
        }

        public String getBatteryInfo() {
                batteryLevel();
                return batterLevel;
        }
}
```

## 8.5. Implementació del servidor

El servidor funciona en segon pla durant tota la vida de l'aplicació i s'encarrega bàsicament monitoritzar l'estat de la bateria. S'ha de tenir en compte que el servidor està funcionant en tot moment i que, per tant, és important que faci el mínim de coses possibles i que tota la funcionalitat estigui implementada en el client.

El codi de la implementació del servidor és el següent:

```
public class ErelisService extends Service
{
        private Timer timer = new Timer();
        private ErelisBatteryInfo batteryInfo = new ErelisBatteryInfo(this);
        private int batteryCharge;

        private void startService()
        {
                timer.scheduleAtFixedRate( new TimerTask()
                {
                        public void run()
                        {
                                batteryCharge = batteryInfo.getBatteryNumber();
                                Log.v("Erelis", "ErelisService: " + batteryInfo.getBatteryInfo());
                        }

                }, 0, 100);
        }

        private void stopService()
        {
                if (timer != null)
                        timer.cancel();
        }

        public IBinder onBind(Intent intent)
        {
                return null;
        }
```

```
	@Override
	public void onCreate()
	{
		super.onCreate();
		startService();
		Log.v("Erelis", "ErelisService Created");
	}

	@Override
	public int onStartCommand(Intent intent, int flags, int startId)
	{
		Log.v("Erelis", "ErelisService onStartCommand");
		return START_STICKY;
	}

	@Override
	public void onDestroy()
	{
		stopService();
		super.onDestroy();
		Log.v("Erelis", "ErelisService Destroyed");
	}

	public int getBatteryCharge()
	{
		return batteryCharge;
	}
}
```

## 8.6. Accés a la base de dades

L'aplicació desa les seves dades de configuració en una base de dades SQLite. La descripció de la taula de configuració és la següent:

| | |
|---|---|
| _id | integer primary key autoincrement |
| battery_level | text not null |
| action | text not null |

Aquestes dades són accessibles tan per part del client com del servidor a través de la classe ErelisDbAdapter que és la que s'encarrega d'encapsular l'accés a més baix nivell i proporcionar uns mètodes públics per tal d'accedir a les dades (per a llegir-es i, si s'escau, per a modificar-les). Així doncs, totes les classes que han d'accedir a la base dades, necessiten una instància d'ErelisDbAdapter.

Internament la classe té una implementació de la classe de l'Android SDK anomenada SQLiteOpenHelper, que és la que s'encarrega de gestionar la creació i l'accés a la base de dades. Per tant, la creació i l'accés a la base de dades (exportats públicament a través dels mètodes “open” i “close) es fan a través de la mateixa.

Aquest és el fragment de codi que ho implementa:

```
public class ErelisDbAdapter {

    private static class DatabaseHelper extends SQLiteOpenHelper {

        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }

        @Override
        public void onCreate(SQLiteDatabase db) {

            db.execSQL(DATABASE_CREATE);
        }

        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            Log.w(TAG, "Upgrading database from version " + oldVersion + " to "
                    + newVersion + ", which will destroy all old data");
            db.execSQL("DROP TABLE IF EXISTS configuration");
            onCreate(db);
        }
    }
    public static final String KEY_BATTERY_LEVEL = "battery_level";
    public static final String KEY_ACTION = "action";
    public static final String KEY_ROWID = "_id";

    private static final String TAG = "ErelisDbAdapter";
    private DatabaseHelper mDbHelper;

    private SQLiteDatabase mDb;

    private static final String DATABASE_CREATE =
    "create table configuration (_id integer primary key autoincrement, "
    + "battery_level text not null, action text not null);";
    private static final String DATABASE_NAME = "erelis";
    private static final String DATABASE_TABLE = "configuration";

    private static final int DATABASE_VERSION = 2;

    private final Context mCtx;
```

```
public ErelisDbAdapter(Context ctx) {
        this.mCtx = ctx;
}

public void close() {
        mDbHelper.close();
}

public long createConfiguration(String battery_level, String action) {
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_BATTERY_LEVEL, battery_level);
        initialValues.put(KEY_ACTION, action);

        return mDb.insert(DATABASE_TABLE, null, initialValues);
}

public boolean deleteConfiguration(long rowId) {

        return mDb.delete(DATABASE_TABLE, KEY_ROWID + "=" + rowId, null) > 0;
}

public Cursor fetchAllConfigurations() {

        return mDb.query(DATABASE_TABLE,
        new String[] { KEY_ROWID, KEY_BATTERY_LEVEL, KEY_ACTION },
        null, null, null, null, null);
}

public Cursor fetchConfiguration(long rowId) throws SQLException {

        Cursor mCursor =
                mDb.query(true, DATABASE_TABLE,
                                new String[] { KEY_ROWID, KEY_BATTERY_LEVEL,
                                KEY_ACTION }, KEY_ROWID + "=" +
                                 rowId,null, null, null, null, null);

        if (mCursor != null) {
                mCursor.moveToFirst();
        }
        return mCursor;

}

public ErelisDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
}

public boolean updateConfiguration(long rowId, String battery_level, String action) {
```

```
                ContentValues args = new ContentValues();
                args.put(KEY_BATTERY_LEVEL, battery_level);
                args.put(KEY_ACTION, action);

                return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
        }
}
```

Com podem veure en el codi, es reimplementen dues funcions de la classe SQLiteOpenHelper per tal d'adaptar-la a la base de dades que són “OnCreate” i “OnUpgrade”, això es fa només perquè necessitem crear la nostra taula anomenada “configuration”, si aquesta no existeix. Al OnCreate li passem una constant amb la cadena de texta de creació dels camps de la base de dades. La resta de mètodes són utilitzats en la seva implementació per defecte.

# 9. Bibliografia

En aquest apartat hi ha un llistat de la bibliografia utilitzada per fer aquest treball. Les entrades estan ordenades a partir del nom l'autor o bé, en molts casos, del nom de l'editor:

[Ana11]: “Google's Android Event Analysis” [en línia], Mithun Chandrasekhar, 2 febrer del 2011, <http://www.anandtech.com/show/4150/googles-android-event-analysis/2>

[And10]: "Android 2.3 Platform Highlights" [en línia], Android Developers. 6 de desembre del 2010, <http://developer.android.com/sdk/android-2.3-highlights.html>

[App09]: “Canalys: iPhone outsold all Windows Mobile phones in Q2 2009” [en línia], Prince McLean, 21 d'agost del 2009, <http://www.appleinsider.com/articles/09/08/21/canalys_iphone_outsold_all_windows_mobile_phones_in_q2_2009.html>

[Bal10]: “Speed up Android by reducing application memory usage” [en línia], Hardward Forums, 20 de març del 2010, <http://www.howardforums.com/showthread.php/1629892-Speed-up-Android-by-reducing-application-memory-usage>

[Ben07]: “Native C applications for Android” [en línia], Ben Leslie, 13 de novembre del 2007, <http://benno.id.au/blog/2007/11/13/android-native-apps>

[Bus05]: “Google Buys Android for Its Mobile Arsenal” [en línia], Ben Eglin, 17 d'agost del 2005, <http://www.businessweek.com/technology/content/aug2005/tc20050817_0949_tc024.htm>

[Cle07]: “Androidology - Part 1 of 3 - Architecture Overview (VIDEO)” [en línia], Mike Cleron, 12 de novembre del 2007, <http://www.youtube.com/watch?v=QBGfUs9mQYY>

[Cne10]: “Android hits top spot in U.S. smartphone market” [en línia], Lance Whitney, 4 d'agost del 2010, <http://news.cnet.com/8301-1035_3-20012627-94.html>

[Com10]: “comScore Reports September 2010 U.S. Mobile Subscriber Market Share” [en línia], comScore, 3 de novembre del 2010, <http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Reports_Septe

mber_2010_U.S._Mobile_Subscriber_Market_Share>

[Der11]: "Nielsen – Users want more Android" [en línia], James Deruvo, 20 d'abril del 2011, <http://androidcommunity.com/nielsen-users-want-more-android-20110426/>

[Dob08]: "The Android Mobile Phone Platform" [en línia], Tom Thompson, Dr. Dobb's Magazine, 4 de setembre del 2008, <http://drdobbs.com/mobility/210300551?pgno=2>

[Goo11a]: "Philosophy and Goals" [en línia], Android Open Source Project, 2011, <http://source.android.com/about/philosophy.html>

[Goo11b]: "About the Android Open Source Project" [en línia], Android Open Source Project, 2011, <http://source.android.com/about/index.html>

[Goo11c]: "Android: momentum, mobile and more at Google I/O" [en línia], The Official Google Blog, 10 de maig del 2011, <http://googleblog.blogspot.com/2011/05/android-momentum-mobile-and-more-at.html>

[Goo11d]: "What is Android?" [en línia], Android Open Source Project, 2011, <http://developer.android.com/guide/basics/what-is-android.html>

[Gre11]: "Android Battery Information From File System" [en línia], GreenGar Studios, 25 d'abril del 2011, <http://www.greengar.com/2011/04/androidbattery/>

[Gub10]: "How many lines of code does it take to create the Android OS?" [en línia], Gubatron.com, 23 de maig del 2010, <http://www.gubatron.com/blog/2010/05/23/how-many-lines-of-code-does-it-take-to-create-the-android-os/>

[Mey10]: "Linux developer explains Android kernel code removal" [en línia], David Meyer, zdnet.com, 3 de febrer del 2010, <http://www.zdnet.com/news/linux-developer-explains-android-kernel-code-removal/389733>

[Mob10]: "Memory Management in Android" [en línia], Mobworld, 5 de juliol del 2010, <http://mobworld.wordpress.com/2010/07/05/memory-management-in-android/>

[Oha07]: "Industry Leaders Announce Open Platform for Mobile Devices" [en línia], Open Handset Alliance, 5 de novembre del 2007, <http://www.openhandsetalliance.com/press_110507.html>

[Oha11]: “Android Overview” [en línia], Open Handset Alliance, 2011, <http://www.openhandsetalliance.com/android_overview.html>

[Pha11]: “350,000 Activations Per Day, Says Schmidt” [en línia], Quentyn Kennemer, 15 de febrer del 2011, <http://phandroid.com/2011/02/15/350000-activations-per-day-says-schmidt/>

[Reu08]: “More mobile phone makers back Google's Android” [en línia], Jennifer Martinez, 12 d'octubre del 2008, <http://www.reuters.com/article/2008/12/10/openhandset-idUSN0928595620081210>

[Reu11]: “Google topples Nokia from smartphones top spot” [en línia], Tarmo Virki i Sinead Carew, 31 de gener del 2001, <http://uk.reuters.com/article/2011/01/31/oukin-uk-google-nokia-idUKTRE70U1YT20110131>

[Xda10]: “How to configure Android's *internal* taskkiller” [en línia], xda-developers forum, 24 de gener del 2010, <http://forum.xda-developers.com/showthread.php?t=622666>