UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN

Escuela Técnica Superior de Ingenieros de Telecomunicación

Universidad Politécnica de Valencia

GeoRBAC: Dispositivo móvil en la arquitectura de control de acceso

Proyecto Final de Carrera

Ingeniería Superior de Telecomunicaciones

Autor: Ignacio Lacalle Úbeda

Director: Carlos Enrique Palau Salvador

12 de diciembre de 2014

Las primeras personas a las que quiero dar mi agradecimiento son mi madre Vicen, mi hermano Miguel y mi padre Rafa. Gracias a ellos este proyecto ha sido posible, así como todo lo que ha sucedido hasta este punto.

Me gustaría dirigir unas palabras al director de este proyecto, Carlos Palau. Ha contribuido de una manera significativa en la consecución de este trabajo, aportando luz durante todo su desarrollo, así como propuestas interesantes desde que tuvimos el placer de tenerle como profesor.

En la misma línea, quiero agradecer su colaboración a Alejandro y Eneko, con quien he trabajado codo con codo durante este tiempo. Gracias por su paciencia y optimismo.

Es complicado citar a toda la gente que ha hecho posible que llegue hasta aquí: amigos, familiares y profesores. Así que valgan estas palabras para que todos ellos se sientan parte de este documento.

Por último, gracias a mis compañeros de estudio más cercanos. Ellos lograron que una dura cuesta se convirtiera en un dulce descenso.

Resumen

Este trabajo está enmarcado en un entorno de acceso basado en roles y en geolocalización: GeoRBAC (Geolocalization Role-Based Access Control).

El control de acceso a infraestructuras es un tema de especial actualidad y que aplica a muy diferentes entornos. Es uno de los pilares en los que se basa la seguridad de los sistemas, tanto físicos (infraestructuras de todo tipo), como virtuales(redes, centros de recursos...).

Asimismo, hoy en día existe una tendencia al denominado Internet de las Cosas (Internet of Things o IoT). El proyecto actual trata sobre la creación de una de las partes intrínsecas del control de acceso vía telemática, la identificación del usuario y su interacción con el sistema, a la vez que ofrece una perspectiva de éste con respecto al IoT.

A pesar de existir soluciones comerciales de diferente índole que permiten realizar control de acceso, la unión de dos criterios como son la asignación de roles y la geolocalización para conseguir dicho sistema conforma un proyecto novedoso y con un abanico de opciones futuras realmente amplio.

El sistema global de GeoRBAC referido consta de tres partes claramente diferenciadas, y éste proyecto, como ya se ha comentado, está centrado en la implementación de una plataforma móvil para la identificación de acceso de usuario y obtención y distribución de posición geográfica empleando tecnología Android.

Se ha desarrollado, en el seno del sistema completo de control de acceso al medio, los mecanismos, sistemas y requisitos tecnológicos necesarios para la identificación del individuo que trata de acceder a la infraestructura.

La solución que se ha propuesto se encuentra desarrollada sobre el sistema operativo Android, pero el concepto, diseño y estructura sistemática (verdaderas claves del proyecto), son válidos para otros sistemas operativos o plataformas móviles.

Por otra parte, el sistema ha sido implementado en un entorno real. Esto ha conllevado un desarrollo completo de todas las partes del sistema de control de acceso, así como su implementación sobre máquinas físicas y las

consecuentes pruebas asociadas a la misma. La solución global, así como cada una de las partes y las interacciones entre ellas han sido implementadas sobre protocolos estándar ampliamente utilizados en la actualidad, como son las redes TCP/IP y la web, con los protocolos HTTP o HTTPS.

El núcleo de este proyecto consiste en el estudio de las capacidades del sistema operativo Android para con el sistema de GeoRBAC planteado, desarrollando asimismo una aplicación móvil sobre dicho sistema operativo, con la intención de ofrecer una plataforma real de identificación de usuario aplicada a un caso de uso.

Se ha ahondado sobre diferentes características de las aplicaciones Android, como son los servicios, los hilos de ejecución, las conexiones web, el almacenamiento, la interfaz de usuario y la comunicación con diferentes sistemas. Tras el análisis exhaustivo de estos elementos, se ha implementado la citada aplicación obteniendo un soporte a una de las partes del sistema global.

En definitiva, el proyecto actual se enmarca en la creación de un sistema completo de control de acceso basado en roles y geolocalización, implementado con tecnologías disponibles en la actualidad, escalable y aprovechando elementos y conceptos que poseen un claro potencial a medio y largo plazo. Aún queda mucho por avanzar en el entorno GeoRBAC en la realidad, sin embargo en este proyecto se realiza una inmersión en las características básicas de estos sistemas, ofreciendo una solución que permite realizar dicho control, y sentando las bases para futuros desarrollos.

Índice general

1. Presentación **5**
1.1. Motivación y Objetivos 5
1.2. Estructura del documento 8

2. Introducción **9**
2.1. Estado del arte 9
2.1.1. Control de acceso 9
2.1.2. Roles en control de acceso: RBAC 19
2.1.3. Geolocalización en control de acceso: GeoRBAC 23
2.1.4. Location Based Services 27
2.2. Primera aproximación 28

3. Arquitectura e Implementación **33**
3.1. Análisis de Requisitos 33
3.1.1. Lista 33
3.1.2. Escenarios 36
3.2. Diseño de la solución 38
3.2.1. Componentes globales 38
3.2.2. Elección de la plataforma 40
3.3. Implementación 51
3.3.1. Elementos de Android utilizados 51
3.3.2. Procesos y capturas de pantalla 82

4. Conclusión **101**

5. Anexo 1 **105**

6. Anexo 2 **106**

7. Anexo 3 **108**

Índice de Figuras **111**

Bibliografía **114**

Capítulo 1

Presentación

1.1. Motivación y Objetivos

La seguridad, principalmente la entendida en los entornos de comunicaciones, es un conjunto de técnicas que tratan de minimizar la vulnerabilidad de los sistemas o de la información en ellos contenida.

Existen muchos tipos de seguridad, así como variedades de implementaciones de la misma en la actualidad. En la carrera de Ingeniería de Telecomunicaciones hemos estudiado ciertos protocolos, técnicas y sistemas enfocados a aportar seguridad a lo que para los profesionales del sector es relevante: las redes. Entre ellos se encuentran tales como IPSec, VPNs, métodos criptográficos, ACLs, Firewalls, IDS.. y una amplia lista de elementos que sirven para crear un entramado de opciones de seguridad disponibles hoy en día, todas ellas frecuentemente utilizadas.

Sin embargo, este proyecto surgió con la motivación de explotar el concepto de una de las premisas básicas de la seguridad: el control de acceso.

El hecho de controlar el acceso de un individuo a unas instalaciones o recursos proporciona un elemento clave a la hora del correcto funcionamiento y mantenimiento de los mismos.

En el entorno de las telecomunicaciones, el control de acceso cobra especial importancia, ya que con frecuencia poseen altos requerimientos de seguridad y confidencialidad, y este tipo de mecanismos resulta básico.

Hoy en día existe una corriente en crecimiento dentro de la cual se defien-

de y justifica el empleo de un escenario denominado Internet de las Cosas, o Internet of Things (IoT), que consiste en el consiste en la integración de sensores y dispositivos en objetos cotidianos que quedan conectados a Internet a través de redes fijas e inalámbricas.

Todo ello produjo que se planteara la opción de profundizar en un sistema de seguridad a partir del control de acceso basado en roles y geolocalización, tratando de emplear el concepto de Internet de las Cosas en su desarrollo.

La creación de roles permite a la organización responsable de controlar el acceso disponer de unos criterios para el manejo del mismo. En base a estos roles pueden tomarse decisiones de entrada, salida, estancia o cualquier combinación entre ellas.

El uso de la geolocalización en el sistema otorga un soporte fiable a la hora de proporcionar acceso. La obtención de los datos de localización de las diferentes unidades susceptibles de acceder a los recursos es un hándicap que se afronta en este proyecto.

Por otra parte, el impacto de las aplicaciones móviles en los últimos años ha sido incuestionable. Una gran cantidad de tareas cotidianas como compras en tiendas, consulta y confeccionamiento de calendarios, intercambio de correos electrónicos, diversión y entretenimiento... así como funciones corporativas y complejas actividades empresariales como transacciones bancarias, acceso a intranets, etc. han pasado a ser el pan de cada día con un dispositivo móvil de última generación en la mano.

Observando la situación en que prácticamente la totalidad de los individuos de la sociedad tiene a su alcance un dispositivo móvil con capacidad de implementar las funcionalidades comentadas y muchas más, y aunando este concepto con el IoT, se planteó la posibilidad de involucrar un smartphone, tablet, PDA o similar en esta parte de la arquitectura de control de acceso.

Desde el comienzo, este proyecto debía tener un enfoque eminentemente práctico, centrando los esfuerzos didácticos y divulgativos principalmente en el estudio de tecnologías y conceptos de apoyo que irían surgiendo a lo largo de su realización.

Tras analizar las diversas opciones, se emprendió el desarrollo de un sistema de control de acceso basado en roles y geolocalización y, en este proyecto en particular, la parte de creación de una plataforma para la identificación

de acceso de usuario en el sistema.

En cuanto a los **objetivos del proyecto**:

El desarrollo del proyecto hace referencia de forma continua a la relación de éste con el sistema global de GeoRBAC, sin embargo la parte implementada en este caso es singular, dejando al margen la parte de procesamiento de eventos central y representación al usuario de la organización que controla el acceso final.

Igualmente, parece relevante comentar de forma breve los objetivos del sistema GeoRBAC:

- Creación de una arquitectura escalable de control de acceso
- Empleo de servicios web como núcleo de la comunicación
- Puesta en marcha en un entorno real del diseño del sistema

Y de forma particular en el proyecto al que se refiere este documento:

- **Desarrollo de una plataforma móvil** que haga uso de los conceptos de geolocalización. Esta herramienta deberá estar implementada de la forma más completa posible, tras analizar tecnologías disponibles para conseguir un control de acceso óptimo.
- **Inclusión de la plataforma de forma integral en el sistema de GeoRBAC global:** Deberá interactuar con el resto de partes de la arquitectura, ajustando su funcionalidad a los formatos y filosofías de trabajo. Tanto en el envío de los datos como en el almacenamiento, concepción como sensor del dispositivo y reconocimiento de requisitos comunes para una implementación real de todas las partes interconectadas.

Y como objetivos y capacidades obtenidas de menor relevancia implícitos en la consecución de los anteriores:

- Conocimiento completo del entorno de desarrollo de aplicaciones nativas sobre el sistema operativo Android
- Aprender a interconectar elementos modulares basados en diferentes tecnologías

- Obtención de un considerable *know-how* en la implementación de Servicios Basados en Localización (LBS)

1.2. Estructura del documento

Este documento presenta tres grandes partes claramente diferenciadas, que tratan de ofrecer una visión global, y a la vez sucinta del proyecto que se ha llevado a cabo.

En primer lugar, existe un apartado de Introducción, que pretende situar al lector en el contexto del proyecto, a la vez que se introducen conceptos utilizados en el desarrollo. Se explica el origen y las primeras fases del proyecto. Este capítulo se divide en dos subsecciones. La primera de ellas versa sobre el control de acceso, los sistemas analizados para el proyecto actual y los conceptos, definiciones y aspectos interesantes de los elementos clave del mismo. De la misma forma, se expone el estado del arte y la importancia de las aplicaciones móviles y los Servicios Basados en Localización, ambos conceptos primordiales en este proyecto. La segunda trata sobre una primera aproximación a la solución del sistema global, sin ofrecer el enfoque desde el punto de vista del dispositivo móvil del control de acceso, como sí sucede en el siguiente apartado.

A continuación, se ahonda en el desarrollo del proyecto, en el apartado de Arquitectura e Implementación. Este capítulo se divide a su vez en tres subsecciones: la primera muestra los requisitos del sistema a desarrollar y en particular en la plataforma móvil que permitirá identificar al usuario que trate de acceder a los recursos. La segunda ilustra el proceso iterativo de diseño de la solución y se culmina en la tercera parte con los detalles específicos de la codificación e implementación de la herramienta completa.

Por último, se realiza un apartado de Conclusión. En dicho apartado se reflexiona sobre la funcionalidad del proyecto, se proponen líneas futuras de desarrollo y se finaliza con una valoración global del proyecto.

Igualmente, el documento dispone de un índice al inicio del mismo que sirve para ilustrar su estructura, así como una lista de figuras, una serie de anexos y la referencia bibliográfica en las últimas páginas.

Capítulo 2

Introducción

2.1. Estado del arte

2.1.1. Control de acceso

El control de acceso es el proceso por el cual, dada una petición de recursos, se permite o niega el acceso a los mismos en base a la aplicación de unas políticas de acceso. Evitar el acceso no autorizado y permitir la libre circulación del personal autorizado es la principal característica de un control de acceso en un sistema de seguridad.

Los sistemas de control de accesos son un elemento clave en un sistema de seguridad, que cada día cuentan con mayor implantación, tanto si se trata de un acceso a infraestructuras físicas (hospital, hotel, fábrica...) como virtuales o de red.

Cada día se hace más necesario controlar, gestionar y monitorizar los movimientos de las personas que entran y salen de las zonas catalogadas como relevantes para la seguridad. Con mayor información de los individuos que acceden, se dispondrá de un entorno más seguro y protegido.

Los sistemas de Control de Accesos permiten configurar los derechos de acceso para todos los usuarios, generar grupos con accesos a las zonas autorizadas, franjas de horario habilitadas, etc.

El control de acceso comprende mecanismos de autenticación, autorización y auditoría. Sus principales objetivos son proteger datos y recursos frente al acceso no autorizado (proteger el secreto) y frente a una modificación no autorizada (proteger la integridad) a la vez que garantizar el acceso de los

usuarios legítimos a los recursos (no denegación de servicio). Con el fin de conseguir estos objetivos, se controlan todos los accesos al sistema y sus recursos, y solo se permite que tengan lugar aquellos autorizados.

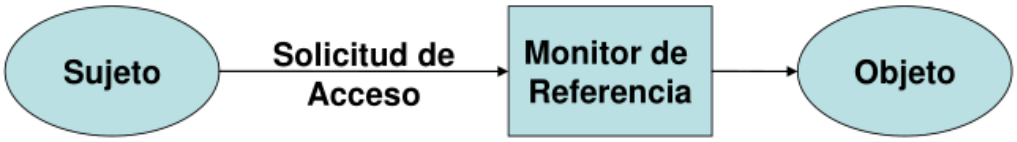

Figura 2.1: Concepto de control de acceso

Por otra parte, el proceso de administración de identidades y de privilegios de los usuarios en los sistemas de información nos permite reducir el riesgo de acceso o modificación no autorizada de la información.

El mantener el acceso a los sistemas a los usuarios válidos con los privilegios correctos requiere la definición de políticas de control de acceso, así como una administración de los mismos.

Las metas que persigue el control de acceso pueden resumirse en las siguientes:

- Evitar la modificación de datos o penetración en infraestructuras críticas por usuarios no autorizados.
- Evitar la modificación no autorizada o no intencional por parte de usuarios autorizados
- Preservación de la consistencia interna y externa de las instalaciones y recursos
- Mayor garantía de confidencialidad de la información, gracias a un acceso controlado a los servicios.
- Registro y revisión de eventos y actividades críticas llevadas a cabo por los usuarios
- Mayor efectividad de los empleados, al minimizarse los conflictos y problemas derivados de la asignación de permisos.

- Menor probabilidad de errores en servicios críticos relacionados con la actividad de usuarios no cualificados.
- Capacidad de monitorizar el uso de los servicios y detectar casos de abuso de los mismos.
- Mayor rapidez y eficacia al revocar permisos en caso de ser necesario, algo que puede ser crítico para la seguridad en determinadas circunstancias.

El control de acceso puede, además, ser un requisito indispensable para la adecuación a determinados estándares de calidad e incluso, a la legislación vigente (en el sector sanitario, por ejemplo).

Los principales retos a que se enfrenta habitualmente el control de acceso a infraestructuras y servicios son:

- Verificar la identidad de los usuarios.
- Verificar que el usuario está solicitando el acceso a un determinado servicio y no se trata de un falso positivo.
- Integrar múltiples niveles de permisos para un usuario concreto.
- Determinar con rapidez y fiabilidad el nivel de permisos del usuario en cualquier momento.
- Gestionar cambios en los requisitos de acceso de los usuarios.
- Restringir los permisos de acceso a los usuarios no autorizados.
- Mantener una base de datos actualizada donde figuren todos los usuarios y los derechos de los que gozan.
- Escalabilidad a la hora de añadir más usuarios al sistema
- Empleo de tecnologías modernas
- Integración del control de acceso con el IoT
- Automatización de procesos de identificación de individuo y toma de decisiones, basado en criterios establecidos por los propietarios de la infraestructura o administradores del sistema.

Normalmente, en el control de acceso se trabaja en zonas que pueden ser departamentos, áreas, puertas independientes, estacionamientos, ascensores, dominios de red, etc. La idea es que cada usuario tenga predeterminados unos permisos de acceso a dichas zonas.

Fases del control de acceso

El desarrollo de un sistema de control de acceso suele tener las tres fases siguientes:

1. La definición de las políticas de seguridad
2. La representación mediante un modelo formal, y
3. la implementación de los mecanismos de seguridad.

Las tres fases, que se asemejan a las del proceso de desarrollo de software, se muestran gráficamente en la figura siguiente.

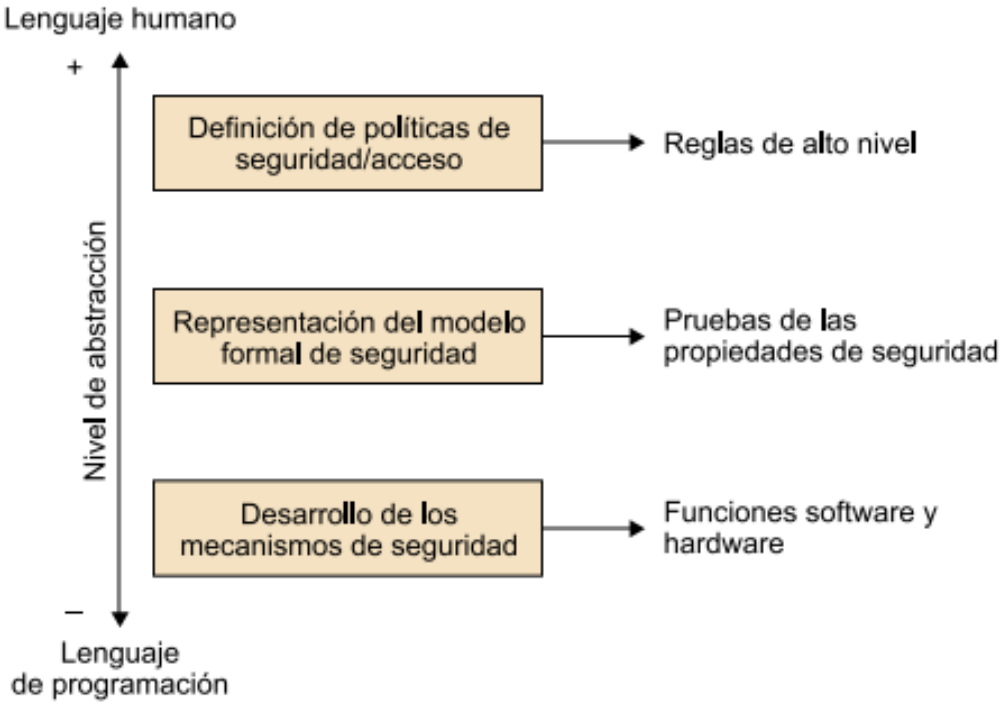

Figura 2.2: Fases del control de acceso

Durante la primera fase del proceso (definición de políticas de seguridad) se establecen mediante lenguaje natural el conjunto de reglas que regulan el acceso a los recursos del sistema de forma abstracta. En esta fase del proceso, las reglas pueden ser vagas e incluso ambiguas, ya que a menudo hacen referencia a leyes, procesos de funcionamiento propios, y procedimientos organizacionales.

Esta ambigüedad hace que las reglas, en esta fase, deban ser interpretadas y no sean aptas para un sistema informático. Por ejemplo, podríamos considerar la regla T̈odos los ingenieros de nivel avanzado tienen acceso al ordenador centralën este caso ïnivel avanzadoës un concepto vago que posteriormente debería ser formalizado.

En la segunda fase del proceso (representación del modelo formal de seguridad) se representa formalmente el conjunto de reglas y su funcionamiento. Esta formalización permite demostrar que un sistema cumple con un determinado conjunto de propiedades de seguridad. Uno de los primeros modelos de seguridad que se propusieron fue el de Bell-LaPadula en 1973. Posteriormente apareció el modelo HRU de Harrison, Ruzzo, y Ullmann en 1976 que formalizó, entre otros, el concepto de matriz de acceso .

Finalmente, la tercera fase del proceso (desarrollo de los mecanismos de seguridad) consiste en la implementación del modelo mediante el uso de lenguajes de programación.

Todo sistema de control de acceso considera los siguientes elementos básicos:

- **Objetos** (también llamados objetivos). Son todas aquellas entidades de un sistema susceptibles de ser protegidas. En el caso de un sistema operativo pueden ser: archivos, directorios, programas, dispositivos, terminales, puertos, etc. En el caso de una base de datos, tenemos: tablas, relaciones, vistas, procedimientos, etc.

- **Acciones.** Todo aquello que se puede realizar sobre un objeto. Las acciones típicas que podemos realizar sobre un fichero son: lectura, escritura, creación, eliminación. En el caso de que el objeto sobre el que se realiza la acción sea un programa, cabría añadir la opción de ejecución.

- **Sujetos** (también llamados iniciadores). Es cualquier entidad con capacidad para requerir el acceso a objetos del sistema. Los sujetos típicos de un sistema son sus usuarios y los procesos del sistema (por ejemplo, un navegador web, un procesador de textos, etc.).

En todo sistema informático, los sujetos realizan acciones sobre los objetos.

El sistema de control de acceso es el encargado de decidir si un determinado sujeto tiene permiso para ejecutar una determinada acción sobre un determinado objeto. La decisión de permitir o denegar el acceso a los recursos se realiza en base a las políticas de acceso.

Las políticas de acceso son el conjunto de reglas que permiten determinar si un sujeto puede realizar una determinada acción (lectura, escritura, modificación, eliminación o ejecución) sobre un objeto. La figura 2.3 muestra un esquema de los componentes principales de un sistema y su interacción con el sistema de control de acceso. Obsérvese cómo el sistema de control de acceso (en el centro de la figura) recibe peticiones de los sujetos para realizar acciones. Evalúa estas peticiones mediante el uso de una política de acceso y actúa en consecuencia permitiendo o denegando el acceso a los objetos.

Tipos de control de acceso

En función de cómo se aplican y gestionan las políticas de acceso podemos distinguir tres tipos fundamentales de control de éste:

1. **Control de acceso obligatorio.** Las políticas son evaluadas por el sistema entendido como un único ente central.

   Los sujetos del sistema no pueden rehacer/redefinir las políticas. Por ejemplo, no pueden dar permisos de acceso a otros usuarios. Los objetos y sujetos del sistema pertenecen a diversas clases de acceso. Así, para acceder a un objeto de una determinada clase hace falta que el sujeto pertenezca a una clase igual o superior (en términos de privilegios).

   Este tipo de acceso se inspira en el funcionamiento militar en el que la información (por ejemplo, planes de ataque) solo puede ser vista por aquellas personas que gozan de un nivel de seguridad suficiente (por

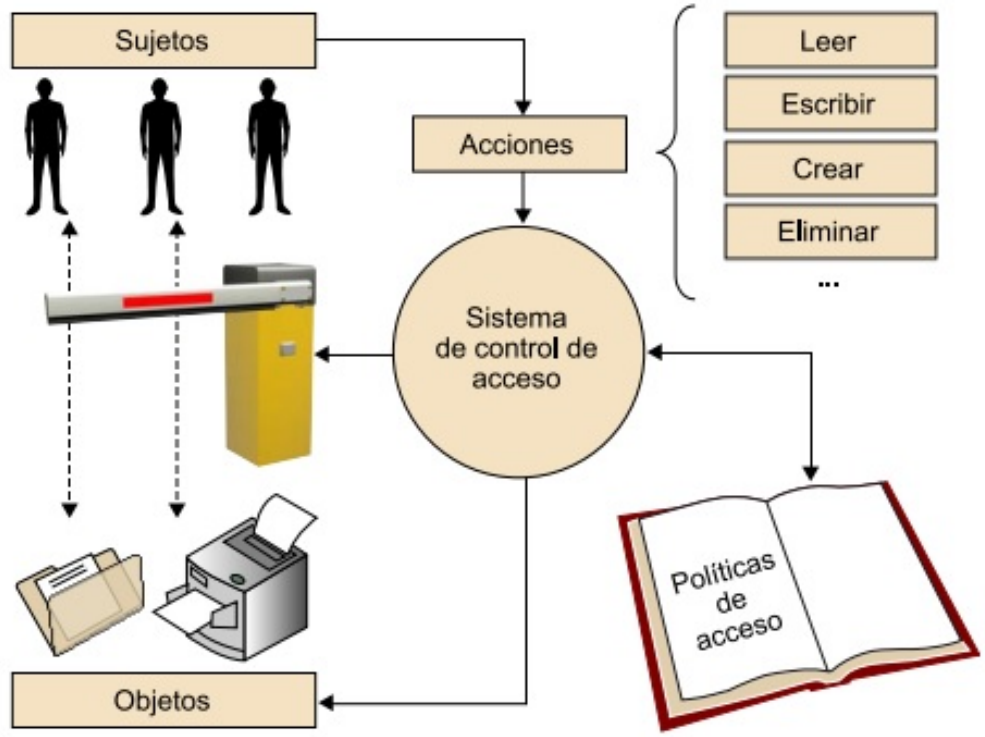

Figura 2.3: Esquema de control de acceso obligatorio

ejemplo, nivel de coronel).

En un sistema control de acceso obligatorio (Mandatory Acces Control MAC), las políticas son evaluadas de forma centralizada por una autoridad .

Típicamente se usan sistemas de seguridad multinivel basados en clasificaciones de los sujetos y los objetos del sistema. Todos los sujetos y objetos tienen asignada una clase de acceso.

Una clase de acceso es un elemento de un conjunto de clases parcialmente ordenado. Donde el orden viene dado por una relación de dominancia representada por $\geq$. La clase de acceso asociada a un objeto indica la sensibilidad de la información contenida en el objeto.

La sensibilidad de una determina información puede entenderse como

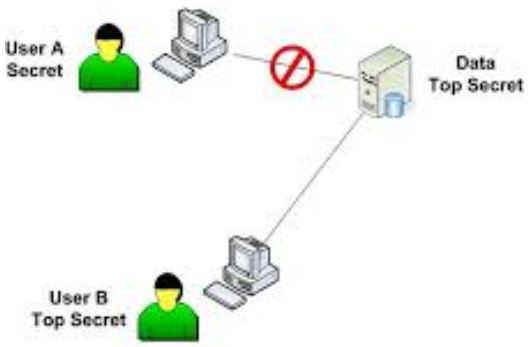

Figura 2.4: Estructura de un sistema de control de acceso

el daño potencial que puede causar la revelación no autorizada de dicha información.

Por otro lado, la clase de acceso asociada a un sujeto indica su nivel de autorización formal (en inglés, clearance). La autorización formal hace referencia al nivel de confianza en un sujeto. Puede verse como la confianza que se tiene en que el sujeto no revelará información sensible a sujetos sin autorización.

Así pues, el control de acceso obligatorio equivaldría a la restricción de acceso a objetos basada en la sensibilidad de la información contenida en los objetos y la autorización formal (clearance) de los sujetos para acceder a una información de ese nivel de sensibilidad.

2. **Control de acceso discrecional**. Las políticas son gestionadas por los propietarios (sujetos) de los recursos (objetos).

   Los sujetos pueden modificar las políticas asociadas a los objetos del sistema. Por ejemplo, un propietario de un determinado objeto del sistema puede dar privilegios de acceso sobre ese objeto a otro sujeto. Este tipo de control de acceso es usado generalmente por los sistemas

operativos.

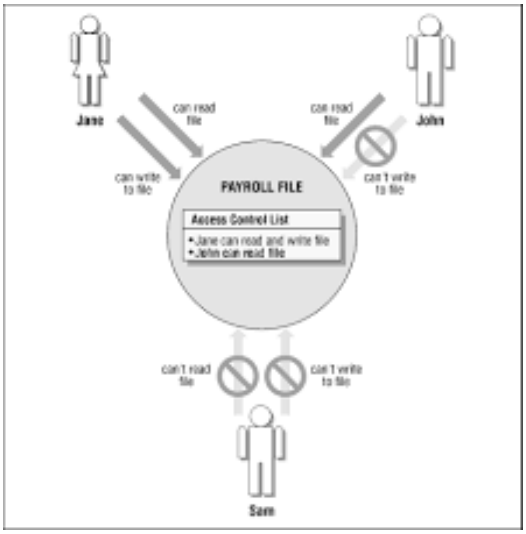

Figura 2.5: Esquema de control de acceso discrecional

El control de acceso discrecional (Discretional Access Control DAC) se basa en evaluar qué sujeto realiza la petición de acceso a los recursos y en un conjunto de reglas de acceso explícitas definidas por algún individuo con privilegios o administrador de sistema.

Recibe el nombre de discrecional debido a que los usuarios del sistema tienen la capacidad de transferir sus privilegios de acceso a otros usuarios. A diferencia del control de acceso obligatorio, el modelo DAC no usa un sistema centralizado en el que solo una única autoridad otorga y revoca privilegios de acceso a los recursos. En consecuencia resulta necesario el uso de políticas de administración.

En los sistemas DAC las políticas de administración regulan los procesos de gestión de privilegios (por ejemplo, transmisión, revocación...) entre los sujetos del sistema.

Existen diferentes variantes de este tipo de control de acceso. A continuación se citan dos de ellas:

- Implementación con propietario (owner):

  Los usuarios (dueños) tienen en la implementación DAC la potestad de determinar las políticas y/o asignar los atributos de seguridad. Un ejemplo de esto es el sistema de permisos del Unix File System y su sistema de usuarios y grupos con permisos de lectura - escritura - ejecución para los objetos (archivos y directorios).

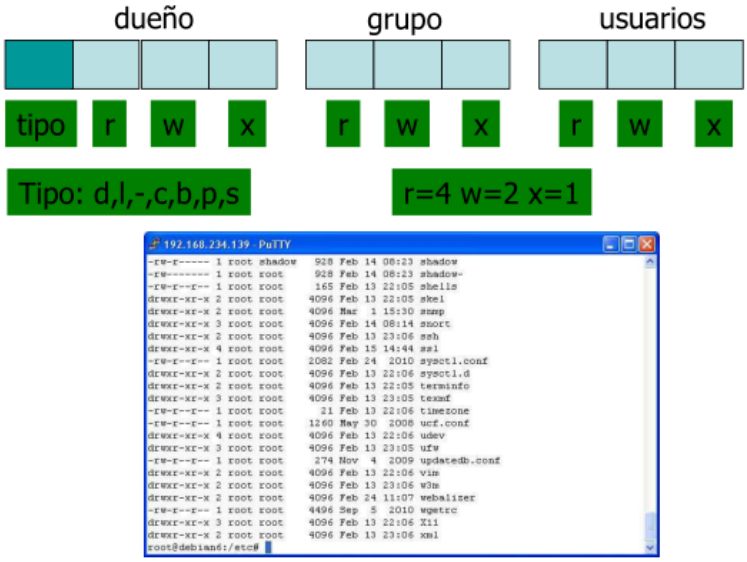

Figura 2.6: Sistema de permisos del Unix File System

- Implementación con capacidades:

  Los sistemas que implementan seguridad por capacidades se describen como que proveen control de acceso discrecional porque permiten a los usuarios transferir sus accesos a otros usuarios, aunque el sistema de control de capacidades no está fundamentado en una restricción del acceso "basado en la identidad de los sujetos"(los sistemas por capacidades no permiten, en general, que los permisos sean pasados a otro usuario; el sujeto que espera recibir sus permisos primero tiene que tener acceso a recibir las capacidades de otro sujeto. Los sujetos no pueden pasar al acceso de cualquier sujeto en el sistema).

3. **Control de acceso basado en roles**. Éste tipo se control de acceso es al que se refiere este proyecto, por tanto en este apartado solamente se ofrece una definición muy general, para permitir de ese modo un análisis más detallado en la sección inmediatamente posterior.

   En RBAC (Role Based Access Control) las políticas son definidas por el sistema pero, a diferencia de las políticas de acceso obligatorio, el acceso no se evalúa en función de permisos individuales sino mediante permisos de clase (o de rol). Cada rol tiene asignados ciertos privilegios y cada sujeto del sistema tiene asignado un rol, así el sujeto adquiere los privilegios del rol al que pertenece. Este tipo de control de acceso es un caso general de los dos anteriores.

2.1.2. Roles en control de acceso: RBAC

En muchas ocasiones resulta poco práctico tener que definir los privilegios de acceso de forma individualizada. El control de acceso basado en roles (Role-Based Access Control RBAC) surge a fines de la década del 80 y toma un fuerte impulso en los 90. Este sistema se fundamenta en la idea de asignar permisos/privilegios para realizar acciones a roles en vez de a sujetos del sistema. Así, cada sujeto del sistema tiene un rol asignado y puede realizar todas aquellas acciones para las que su rol tiene privilegios.

Es decir, el concepto básico de RBAC es que los usuarios son asignados a roles, los permisos son asociados a roles y los usuarios adquieren permisos siendo miembros de roles.

Un rol es la función que alguien o algo cumple. Por ejemplo, en los sistemas operativos suele hacerse una clara distinción entre el rol de administrador y el rol de usuario básico.

Obsérvese que la gestión de los permisos se simplifica claramente respecto a los anteriores modelos puesto que una vez asignados los permisos a los roles, la única tarea consiste en asignar correctamente los roles a los sujetos del sistema.

El modelo de control de acceso basado en roles es lo suficientemente flexible como para funcionar como el control de acceso obligatorio y el control de acceso discrecional. En realidad podemos considerar que tanto el modelo

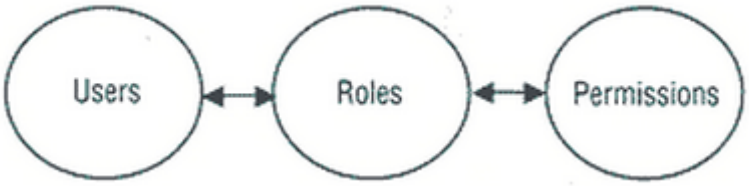

Figura 2.7: Modelo global de RBAC

MAC como el DAC son casos particulares del modelo RBAC.

Para implementar RBAC se necesitan mecanismos que permitan:

- Identificar los roles en un sistema y asignar los sujetos a los roles definidos.
- Establecer los permisos de acceso a los objetos para cada rol.
- Establecer permisos a los sujetos para que puedan adoptar roles.

A continuación se hace un repaso a las características principales de RBAC:

Administración de autorizaciones: La clave del funcionamiento de RBAC es la gestión de las autorizaciones por parte de los responsables de la organización y de los administradores del sistema.

La asignación de permisos a usuarios tiene dos partes:

1. Asociar usuarios a roles: Este hecho implicará la creación de una base de datos con información sobre los usuarios:
   - Identificación.
   - Roles a los que pertenecen.
2. Asignar permisos para objetos a roles.
   - Comparación con otros roles.
   - Tareas específicas de cada rol.

Si un usuario cambia de tareas (por ejemplo, es ascendido en su organización) solamente modificando el rol al que pertenece en dicha base de datos, el sistema se encontraría actualizado y en el mismo estado de funcionamiento.

Jerarquía de roles: Los roles también poseen relaciones de jerarquía.

Pueden heredar privilegios y permisos de otros roles de menor jerarquía, simplificando la administración de las autorizaciones. Ésta filosofía comparte conceptos con la programación orientada a objetos, en la cual un objeto puede "heredar" atributos y métodos de su "padre" y extenderlos.

Menor privilegio:

- Permite implementar la política del menor privilegio posible.
- Si una tarea no va a ser ejecutada por un usuario, entonces su rol no tendrá los permisos para hacerla, de esta manera se minimizan riesgos de daños

División de responsabilidades:

- Se basa en el principio de que ningún usuario tenga suficientes privilegios para usar el sistema en su propio beneficio.
- Por ejemplo, una persona encargada de autorizar un pago no debe ser el beneficiario de ese pago. Se puede establecer de dos formas:
  - Estáticamente: definiendo los roles excluyentes para un mismo usuario;
  - Dinámicamente: realizando el control al momento de acceso.

A continuación, se procedió a realizar un estudio más pormenorizado de los componentes y funcionamiento de RBAC, para lo que se acudió a la definición formal realizada por el NIST [1].

RBAC define cinco conjuntos de elementos de datos básicos: usuarios (USERS), roles (ROLES), objetos (OBS), operaciones (OPS) y permisos (PRMS). Como se ha mencionado, RBAC está fundamentalmente definido en término de usuarios individuales asignados a roles y los permisos asignados a roles. Se puede considerar al rol como la manera de describir relaciones muchos-a-muchos entre usuarios individuales y permisos. La figura 2.8 esquematiza la relación de los elementos que componen al modelo.

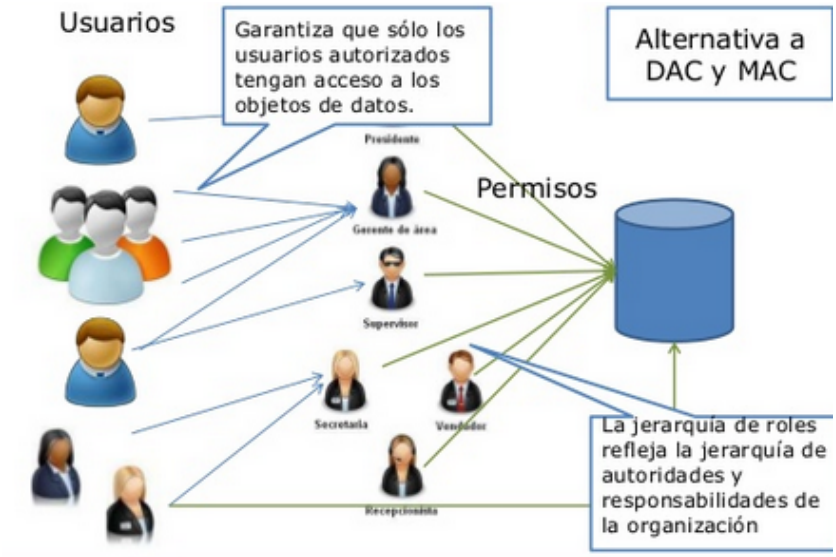

Figura 2.8: Aproximación de RBAC

Por razones de simplicidad un usuario está definido como un ser humano (pero el concepto puede ser extendido a máquinas, redes, etc). Un rol es un trabajo o función en el contexto de una organización con alguna semántica asociada respecto a la autoridad y responsabilidad conferida al usuario asignado al mismo. Un permiso es una aprobación para realizar una operación sobre un objeto determinado.

Una operación es una acción que al ser invocada ejecuta una tarea para el usuario. Un objeto es una identidad que contiene o recibe información.

El propósito de cualquier mecanismo de control de acceso es proteger los recursos del sistema. Cualquier incremento en la flexibilidad de controlar el acceso a estos recursos también fortalece la aplicación del principio del menor privilegio. Para que un usuario pueda ejercer los permisos para los cuales está autorizado, es necesario que éste active los roles que tienen asociados estos permisos. RBAC define un conjunto de sesiones (SESSIONS) donde cada sesión es un mapeo entre el usuario y un subconjunto de roles activos, los cuales están asignados al usuario. La utilización de sesiones facilita en gran medida la aplicación de este principio, ya que aumenta la granularidad del control de acceso.

Las decisiones de acceso son tomadas por usuario y sesión, es decir un

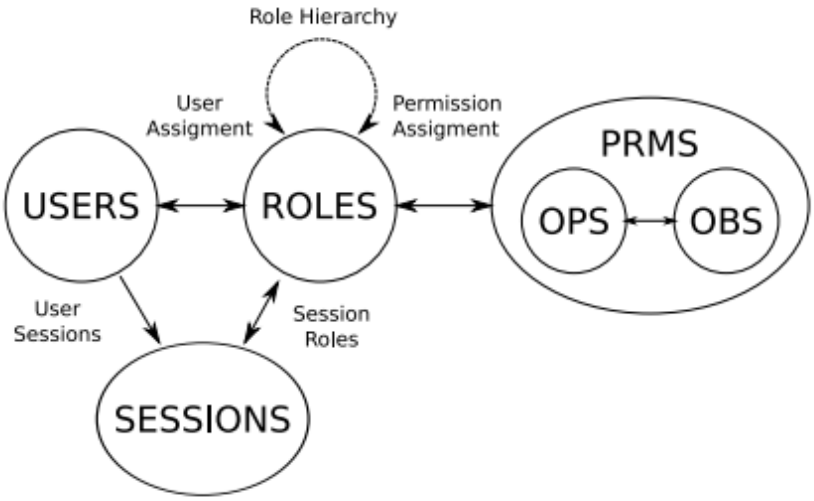

Figura 2.9: Esquema formal de RBAC

usuario está autorizado a realizar una determinada operación sobre un determinado objeto si existe algún rol activo en dicha sesión que tenga asignado el permiso correspondiente.

Para concluir con los aspectos de RBAC de interés para este proyecto, se muestra en la figura 2.9 el esquema definitivo a seguir de implantación de la filosofía de control de acceso basado en roles:

2.1.3. Geolocalización en control de acceso: GeoRBAC

Un estadio más del modelo que se desarrolla en el actual proyecto, y tal vez el más importante, es la introducción de la componente espacial.

Se ha hecho un repaso a los diferentes sistemas de control de acceso, enumerando sus características, y tratando de definir su importancia en los entornos de seguridad, tanto física, como lógica.

Sin embargo, hoy en día el concepto de Internet de las cosas está cambiando el enfoque de muchos mecanismos relacionados con esta área. Cuando un individuo desea acceder a unas instalaciones, a unos recursos, o a una red se le pide una identificación, como ya se ha comentado. Hasta ahora, esa

identificación requería, en mayor o menor grado, la interacción del usuario con el sistema. Tal vez a partir de contraseñas, identificadores físicos u otros elementos existentes. IoT, en aplicación al control de acceso, trata de establecer formas alternativas que supongan una interacción directa entre objetos del usuario (que lo identifiquen) y los componentes del sistema destino al que usuario trata de acceder, de forma automática. En este aspecto es donde entra en juego el concepto de la geolocalización.

GeoRBAC introduce la localización geográfica como una restricción o variable añadida en la activación de roles: GeoRBAC podría explicarse de la siguiente forma [2]:

El amplio y generalizado despliegue de servicios basados en localización y aplicaciones móviles, así como la creciente preocupación por la gestión y distribución de información geográfica en aplicaciones estratégicas como protección medioambiental y seguridad nacional han resultado en una fuerte demanda de sistemas de control de acceso basados en la geolocalización.

Es suficiente con que las organizaciones definan zonas de actuación,y los correspondientes roles y funciones que pueden desempeñar en ellas. Por ejemplo, en la Universitat Politècnica de Valencia podrían definirse las zonas a las que tienen acceso los alumnos de telecomunicaciones. En este esquema, los alumnos de la escuela podrían entrar en los dos edificios pertenecientes a la ETSIT con rol de Alumno, así como en el de Diseño con rol de Prácticas de Laboratorio. Y a partir de ahí, definir diferentes parámetros de seguridad. Se observa este ejemplo en la Figura 2.10.

Las aplicaciones relacionadas con estos términos poseen requisitos interesantes sobre los sistemas de control de acceso. En particular, los permisos asignados a los usuarios dependen de su posición en ese espacio. Las políticas de control de acceso deben tener en cuenta la ubicación de los objetos y la posición espacial de los sujetos. Además, como en muchas otras aplicaciones, los usuarios con frecuencia pertenecen a categorías claramente definidas.

Como ejemplo ilustrativo, considérese una aplicación móvil para el personal y los pacientes de una organización o complejo sanitario. A los individuos se les otorga un terminal con capacidad de captación y distribución de información sobre su posición geográfica mediante el cual pueden hacer peticiones de servicio a un servidor. La organización consiste en individuos que tienen diversos roles funcionales: por ejemplo Enfermero, Doctor y Paciente. Dependiendo del contexto organizacional, los servicios disponibles a los usuarios

Figura 2.10: Ejemplo de creación de zonas espaciales para GeoRBAC

pueden variar basándose en el rol al que pertenezcan. Por ejemplo, los servicios disponibles a los doctores serán diferentes a los de los Enfermeros, no solamente por las cuestiones de preferencia sino principalmente por razones funcionales. Además, la disponibilidad de los roles y por tanto los servicios de acceso pueden depender de la posición del usuario que realice la petición. Por ejemplo, un doctor puede tener permiso para acceder al registro y datos de los pacientes solamente cuando se encuentra en las dependencias (lógicas o físicas) del departamento al que ha sido asignado.

Asimismo, los roles pueden estar interrelacionados por relaciones de precedencia con significado espacial. Por ejemplo, un doctor es un miembro del personal de la organización y como tal puede estar autorizado a acceder a servicios adicionales cuando se encuentre localizado dentro de los límites del hospital en que trabaja.

Por otra parte, pueden surgir conflictos de intereses entre roles, debido a la posición de los individuos. Por ejemplo, a un doctor no le debería estar permitido ser, al mismo tiempo, director del hospital, por tanto, en la misma localización. Para solventar estas dificultades, un modelo de control de acceso basado en roles y geolocalización se antoja necesario.

Desde que las organizaciones poseen ciertas características y servicios que conciernen a la posición geográfica y los miembros de la misma están divididos en categorías, como enfermero y doctor, los modelos de control de acceso basado en roles (RBAC) [1] representan una elección razonable a la hora de implementar opciones de seguridad.

En los últimos tiempos han sido propuestos varios sistemas de control de acceso basados en roles y geolocalización que extraen los datos geográficos almacenados en bases de datos espaciales DBMS. Aunque algunas de estas propuestas preliminares han sido evaluadas como mejoras de los mecanismos de control de acceso basados en información contextual, como información temporal, dichas aproximaciones han resultado excesivamente simples y no se han tenido demasiado en cuenta. Como modelo, GeoRBAC extiende el modelo RBAC con el concepto de rol espacial y apuesta por la representación homogénea de todos los aspectos incluyendo roles, objetos, sujetos y información contextual como la posición geográfica del usuario.

En el proyecto actual, el modelo espacial que se adopta obedece a los estándares de la OGC (Open GeoSpatial Consortium) [1].

2.1.4. Location Based Services

Locational-Based Service (LBS)[3], o en castellano, Servicios Basados en Localización es una definición que engloba todos aquellos sistemas o aplicaciones que aprovechan los datos espaciales, coordenadas geográficas, para ofrecer un servicio o una información basada en éstos.

Existen cientos de ejemplos de aplicaciones conocidas, y no tanto, que ofrecen LBS. Algunos de ellos son los siguientes:

- Posicionamiento de Google.
- Servicios de guía de conducción por GPS
- Obtención de vuelos u ofertas a partir de la posición del usuario que navega en la página web
- Oferta de locales de restauración o comercios cerca de la zona
- Páginas amarillas
- Contactos de redes sociales, etc.

La ubicación del usuario es una dimensión importante en este nuevo mundo de servicios de datos. No solo ha permitido a las empresas concebir completamente nuevos conceptos de servicio, sino también está teniendo el potencial para hacer los servicios de mensajería y de Internet móvil más relevantes cuando la información se ajusta al contexto local de los clientes. Por ejemplo, es muy habitual que los dispositivos móviles traigan incorporadas aplicaciones meteorológicas que informan del parte en la zona en la que se encuentra el usuario sin que éste tenga que intervenir de ninguna manera.

Además, la información de ubicación es capaz de aumentar la calidad de experiencia de los usuarios que consumen aplicaciones móviles. Es por esta razón que la mayoría de empresas del sector de las telecomunicaciones, como las operadoras móviles o las grandes compañías de Internet como Google o Facebook presten cada vez más atención a los servicios y aplicaciones basados en la localización.

Un sistema LBS se caracteriza por utilizar tecnología de sistemas de información geográfica, alguna de las tecnologías de posicionamiento bien sea del lado cliente (Por ejemplo los sistemas GPS anteriormente mencionados) o del lado del servidor (Por ejemplo un servicio de posicionamiento suministrado por el operador de la red; generalmente algún método de triangulateración

espacial y/o ubicación por celdas) y tecnología de comunicación de redes para transmitir información hacia una aplicación LBS que pueda procesar y responder la solicitud.

Este esquema responde exactamente a cómo se quiere desarrollar este proyecto.

Generalmente hay dos modos de clasificación de los servicios LBS: activos y pasivos.

1. Un sistema LBS activo es aquel a partir del cual el usuario obtiene información de su posición espacial y la aprovecha para relacionarla con elementos de su entorno. Hace un uso activo de su geolocalización. Ejemplos de este tipo de aplicaciones podrían ser:

   - Buscar restaurantes cercanos
   - Cómo llegar desde mi ubicación hasta algún destino.

2. Los sistemas LBS pasivos generalmente son diseñados para organizaciones que desean llevar un control de la geolocalización de sus recursos móviles, y en base a ello tomar decisiones o políticas no-directamente relacionadas con la posición geográfica ni su entorno.

   Unos ejemplos comunes de sistemas LBS pasivos son:

   - Seguimiento de flotas
   - Rastreamiento del flujo de mercancías
   - Control de envíos
   - Servicios de transporte como : taxi, ferrocarril, transporte aéreo
   - Vehículos de urgencia médica, etc.

Por definión, la aplicación que se va a desarrollar, y por ende el sistema global GeoRBAC que lo envuelve se trata de un sistema LBS pasivo.

2.2. Primera aproximación

Para introducir definitivamente el contenido de este proyecto, comentar que su origen fue un desarrollo emprendido para un trabajo de calificación de la asignatura de Redes Corporativas II.

En este trabajo se pretendía implementar un procesador de enventos basado en la tecnología ESPER y en Java, en la que se analizaban la entrada

o salida de diferentes usuarios en una plantilla de la planta de un Hospital.

Se analizaba, a partir de sus interacciones con los espacios geométricos predefinidos en la planta su permiso para acceder a las mismas.

Y el diseño funcional consistió, básicamente, en:

- Definición de tres roles:
  - Doctor
  - Becario
  - Enfermero
- Asignación de permisos por habitación. Ejemplo: Habitación 1: Roles permitidos: Doctor y Enfermero.
- Creación gráfica de la aplicación
- Desarrollo del algoritmo de procesado de eventos. Complex Event Processor (CEP).

El resultado se observa en las capturas de pantalla del applet Java desarrollado: En la figura 2.11 se muestra una vista general de la interfaz que permite el seguimiento del acceso. Los puntos que se mueven por los p̈asillosde la planta son usuarios, representando sus respectivos roles según el color del punto. La leyenda de los roles aparace en la figura 2.12.

Por último, en la captura 2.13 un usuario con un rol no permitido en esa habitación se muestra con un círculo rojo alrededor.

Sin embargo, en esta aproximación no hubo separación modular, ni enfoque MVC, estando toda la implementación centralizada en una codificación en Java, enfocando el trabajo en la puesta en práctia a través de un applet sencillo de Java de la exploración divulgativa de los conceptos tecnológicos de acceso a base de datos y gestión de eventos que involucraba el mismo.

En definitiva, debido a el estudio de la permisividad o no de acceso de un rol de trabajador sanitario a una habitación determinada surgió la idea de desarrollar e implementar en un **entorno real** un sistema más complejo de RBAC, involucrando, en el caso del proyecto que yo he desarrollado, la creación de una aplicación móvil que permita identificar al usuario y proporcionar al sistema procesador de eventos la información geográfica del mismo.

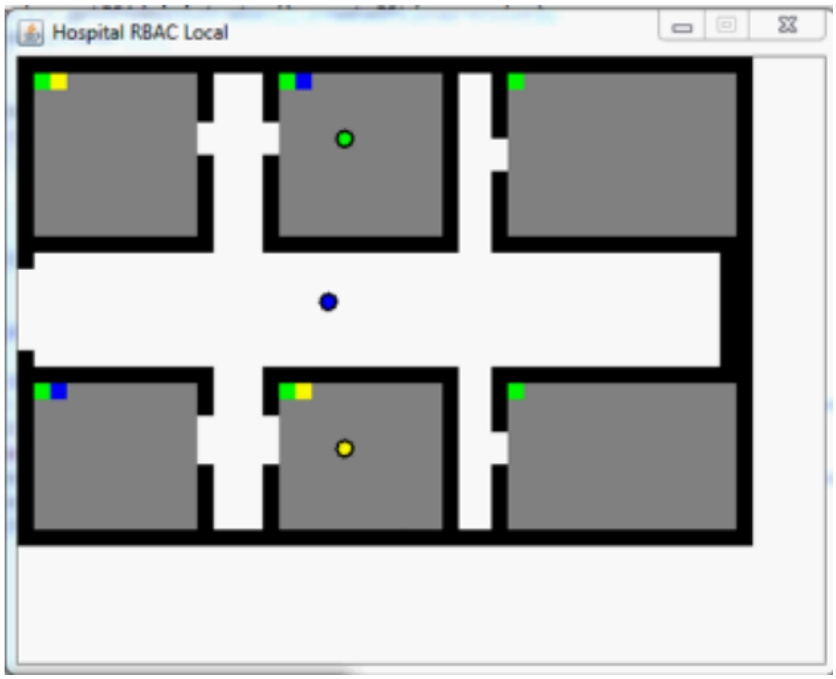

Figura 2.11: Representación gráfica Primera Aproximación

Figura 2.12: Diferentes roles definidos

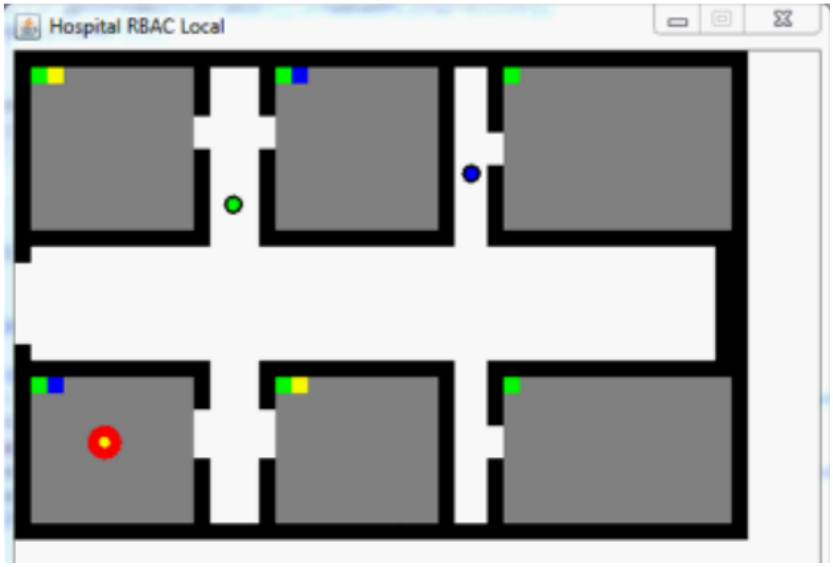

Figura 2.13: Acceso incorrecto

Capítulo 3

Arquitectura e Implementación

3.1. Análisis de Requisitos

3.1.1. Lista

Tras observar la evolución que se ha seguido hasta el punto de arranque del proyecto que se presenta en este documento, se procede en este capítulo a explicar en detalle los requisitos previos que se valoraron a la hora de desarrollarlo.

Este proyecto está enmarcado en un sistema de control de acceso basado en roles y geolocalización completo. Concretamente, trata de implementar de una forma íntegra la parte de identificación de acceso y distribución de información geográfica en un entorno real por parte del usuario que trata de ganar acceso a unas instalaciones.

Tal y como se ha dicho, se trata del desarrollo de una aplicación para dispositivo móvil que consiga que éste sea el método de identificación de usuario, y el punto de interconexión del mismo con el procesador central encargado de implementar la lógica de seguridad pertinente.

En este apartado se cita, en primer lugar una lista de requisitos que se valoraron para la disposición global del sistema y para el caso de la plataforma móvil y , a continuación, una serie de escenarios que se contemplaron, particularizándolos en el caso que aplica en este proyecto.

Requisitos Globales:

- Crear un sistema escalable.

Dado el caso que se incremente la cantidad de usuarios en una organización, como el número de roles en los que haya que incluirlos o que las instalaciones crezcan y haya que añadir nuevas zonas geográficas en el sistema, se crea de tal manera que la forma de funcionamiento no se vea alterada, y el número de modificaciones a realizar sobre el sistema sean las menores posibles.

Esta cualidad permite incrementar la cantidad de operaciones (concurrentes) a la vez que se mantiene un nivel de servicio aceptable. Es, por tanto, escalabilidad horizontal la que se busca en el sistema completo de GeoRBAC. Para ello se usan componentes y tecnologías que permitan implementar dicha propiedad.

- Combinación de conceptos:
  - CEP: Complex Event Processing: Motor de eventos que analice la capacidad de acceso de un individuo a una zona determinada en función de su rol.
  - Almacenamiento de datos de geolocalización según los estándares de la OGC.
  - Internet de las Cosas: Que los diferentes componentes que formen parte de la solución final interaccionen entre ellos con la mínima intervención del usuario.
  - Introducción de los dispositivos móviles como proveedores de la identidad del usuario que trata de acceder.
  - Tecnologías modernas de representación de los datos al usuario: mapas en los que se vea a los individuos moviéndose por las instalaciones, etc.
- Constitución en tres partes claramente diferenciadas
  - En la sección inmediatamente posterior se especificará cada una de ellas y se centrará la implementación de una de las mismas en el proyecto actual.
    - Identificación de usuario y distribución de información geográfica mediante dispositivo móvil
    - Motor de eventos, decisiones de acceso e implementación de los elementos centrales del sistema en diferentes servidores

    - Representación al usuario. HMI (Human-Machine-Interface)
- Creación de puntos de interacción entre ellas para la garantía de un correcto funcionamiento del sistema completo.

Requisitos Particulares: A continuación se listan los requisitos que se identificaron al comienzo del desarrollo de la aplicación móvil para la identificación de usuario. Más adelante, en el momento de la especificación del desarrollo de la misma, se explicarán las dificultades que han ido surgiendo a lo largo de dicho desarrollo.

Creación de una aplicación para dispositivo móvil con las siguientes características:

- Interfaz intuitiva
- Interfaz adaptable a los diferentes dispositivos
- Capacidad de captación de posición geográfica
- Distribución de información mediante de diversos protocolos
- Configuración de los parámetros básicos de usuario
- Informar acerca de la información de usuario relevante para GeoRBAC en cualquier instante:
    - ID
    - Roles
    - Posición
- Capacidad de recepción de información de los servidores centrales del sistema
- Escalable: Que no interfiera con el resto de aplicaciones en funcionamiento en el dispositivo
- Que pueda ejecutarse en segundo plano

Todas estas características han sido implementadas y se detallarán en la sección de implementación.

3.1.2. Escenarios

Al comienzo del proyecto, una vez analizado el estado del arte y en el paso previo a la elección de caso de uso concreto a desarrollar, surgió una idea de desarrollo que finalmente se descartó.

En principio se consideró aplicar el conocimiento que ya se poseía sobre sistemas RBAC a aplicaciones marítimas. En particular se planteó la posibilidad de implementar una aplicación móvil, y a su vez el sistema de GeoRBAC completo en la cual ésta se encuentra incluida, que permitiera crear zonas o *canales marítimos* en el Mar Mediterráneo, entre las zonas de Valencia, Dènia e Ibiza, de tal forma que, a partir de las posiciones de los barcos que navegaran por dichos canales, pudieran o no hacerlo según sus roles.

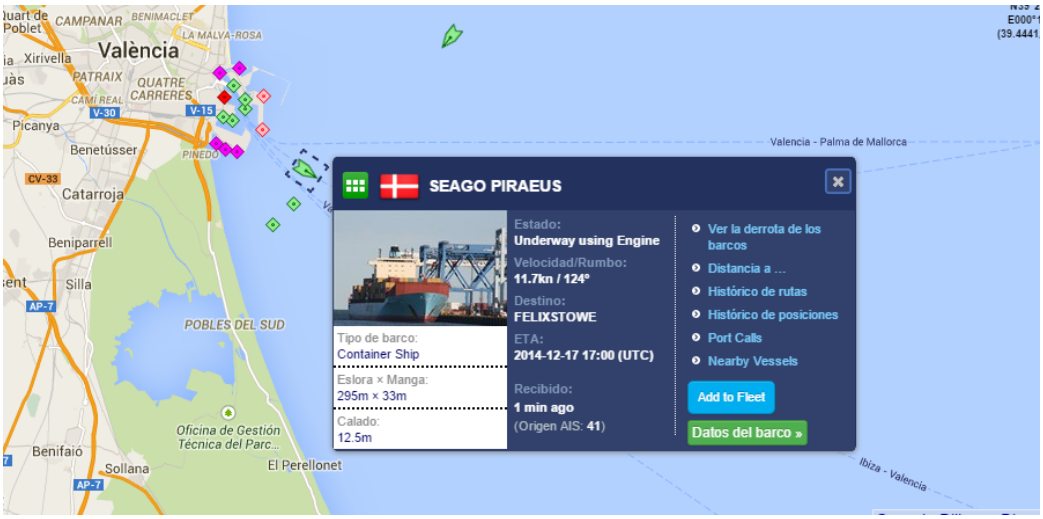

Figura 3.1: Disposición de barcos con sistema AIS

El diseño que se esbozó fue conseguir las posiciones de los barcos mediante la extracción de mensajes AIS que éstos emiten, por ley, sobre los datos de su embarcación (eslora, manga, tipo de barco (rol en nuestro supuesto caso), etc) entre los que se incluye posición geográfica. En la imagen 3.1 se muestra una imagen de la página web *www.marinetraffic.com* que captura las señales AIS de los barcos de una zona y las muestra en una interfaz gráfica, tanto su posición como dichos datos mencionados.

Sin embargo, este caso de uso, como se ha comentado, decidió ser descartado debido a las complicaciones logísticas existentes para su ejecución. Ya que uno de los requisitos del sistema global de GeoRBAC en general, y de

la parte de identificación de usuario en particular, era su implementación en un entorno real se optó por evaluar otro escenario.

Finalmente se escogió realizar un control de acceso de alumnos de la Universitat Politècnica de València. Para ello, el escenario consistía en la creación de zonas basadas en las coordenadas de los edificios de la ETSIT, a los cuales solamente tendrían acceso unos roles determinados.

En este nuevo escenario, el método de identificación del usuario y desde donde debería enviar los datos de geolocalización debía ser un dispositivo móvil, de tal forma que le permitiera moverse libremente por las instalaciones de la UPV y que el sistema central pudiera controlar sus intentos de acceso a las zonas definidas.

En la imagen 3.2 se observa una imagen de la definición de una zona planteada para el desarrollo del escenario definitivo del sistema de GeoRBAC y la aplicación móvil.

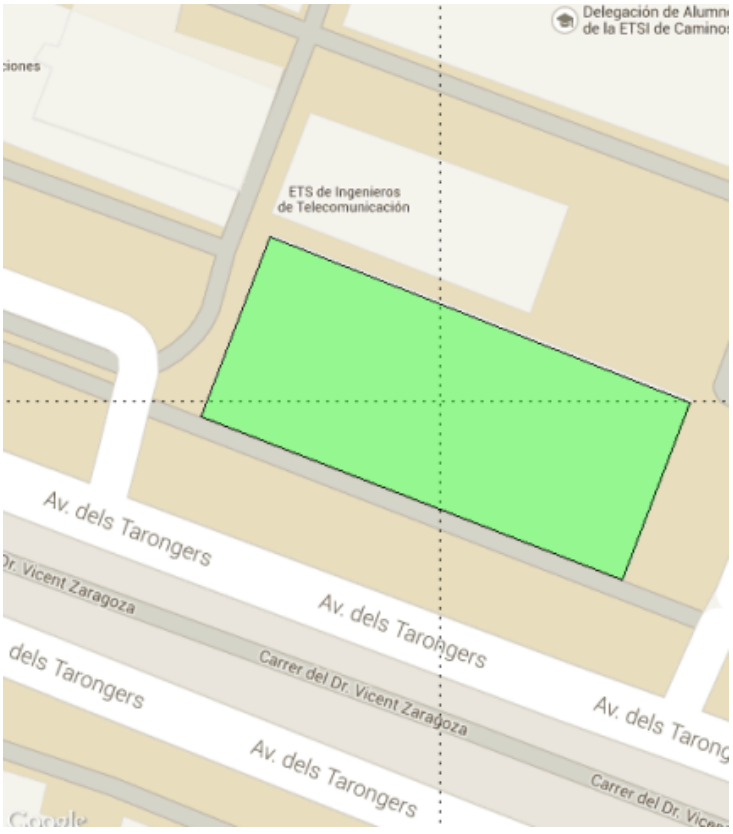

Figura 3.2: Escenario definitivo

3.2. Diseño de la solución

3.2.1. Componentes globales

En este apartado se muestra una visión global del sistema de control de acceso basado en roles y geolocalización GeoRBAC dentro del cual se encuentra enmarcado el proyecto final que ha realizado el autor de este documento.

En la imagen 3.3 se muestra el esquema de componentes que forman parte del sistema GeoRBAC. Inmediatamente después se encuentra una breve explicación de cada una de las partes, así de como los puntos de interacción entre ellas.

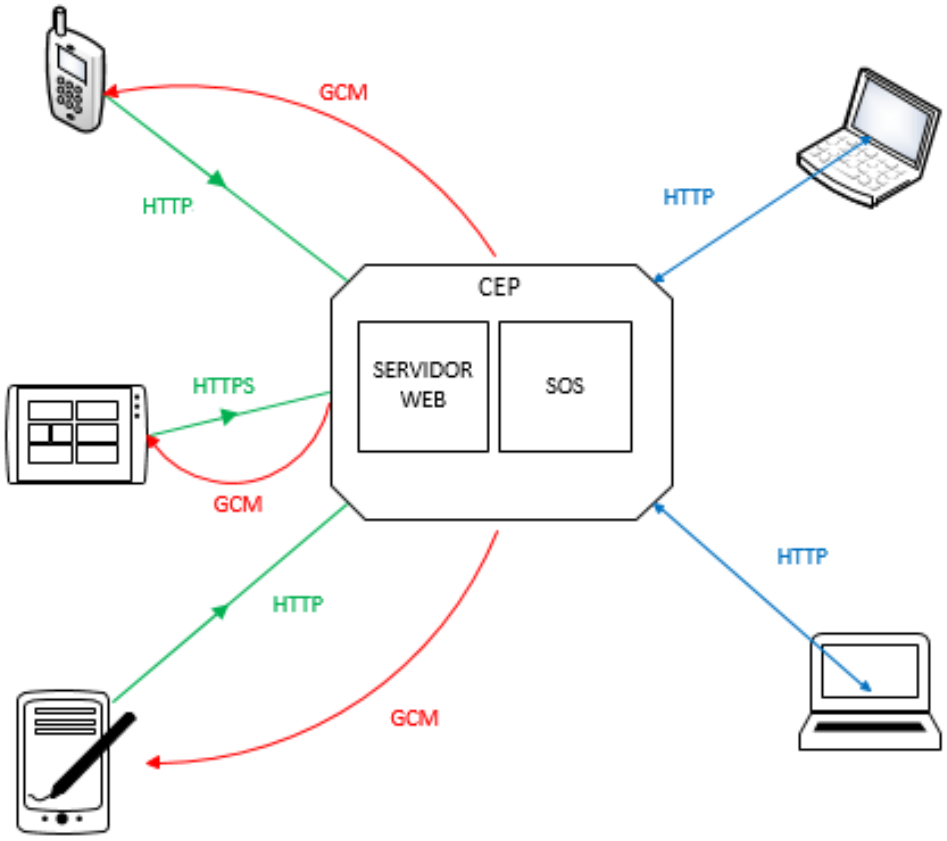

Figura 3.3: Esquema global del sistema GeoRBAC

La explicación se realiza siguiendo el esquema de derecha a izquierda:

Aplicación cliente:

Se trata de una aplicación gráfica que muestra la posición actualizada de los diferentes sensores(dispositivos) registrados en la base de datos del SOS y la representa en un mapa donde aparecen reflejadas las zonas definidas de interés por la organización. También posee un log visible donde informa de las diferentes acciones que van sucediendo en el mapa de representación: las entradas y las salidas de los usuarios a las instalaciones.

Su interacción con el resto de componentes del sistema se limita a establecer una conexión mediante WebSockets con el servidor central de CEP e interpretar los datos de posiciones y sensores que éste le envíe.

Procesador de eventos (Complex Event Processor o CEP):

Es la parte que se encarga de la lógica central del sistema. Está desarrollada principalmente en Java, y sus funciones son:

- Registrar a los dispositivos de usuario en la base de datos del SOS
- El servidor web SOS implementador recibirá las posiciones de los dispositivos
- Se encargará de procesar los eventos y decidir el acceso o no en función del rol y la posición aportadas por el evento generado
- Difundirá a la aplicación cliente los datos necesarios para que ésta pueda realizar su función de muestra

Identificación de usuario:

Esta es la parte que he realizado en el PFC, y cuyo desarrollo viene reflejado en las páginas de este documento. Tanto el diseño como la implementación particular de la aplicación de dispositivo móvil de identificación y distribución de posiciones geográficas vienen referidos en los siguientes apartados. Como se ve en el esquema, la interacción de esta parte con el resto del sistema es únicamente con la parte central del CEP, enviando la solicitud de registro y la notificación de los datos de geoposicionamiento mediante web, y recibiendo comunicaciones por parte del servidor mediante Google Cloud Messaging (GCM).

3.2.2. Elección de la plataforma

Para la implementación práctica de este proyecto, se hubo de escoger, como se ha dicho, una plataforma que ejerciera de identificador de usuario. Se escogió la realización sobre un dispositivo móvil, y, en particular, en el sistema operativo Android.

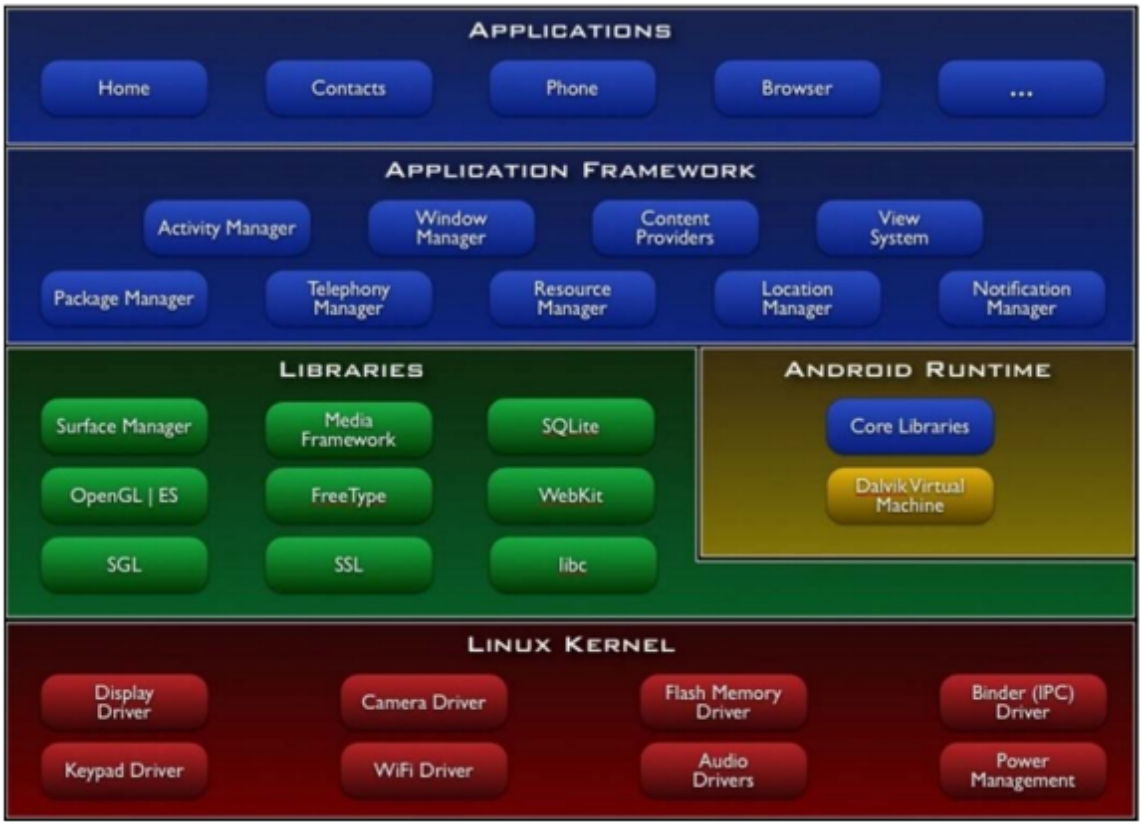

Figura 3.4: Arquitectura del Sistema Operativo Android

Existen muchas plataformas para dispositivos móviles (iPhone, Symbian, Windows Phone, Anroid, BlackBerry, Palm, Linux Mobile...): sin embargo Android presenta una serie de características que lo hacen diferente.

Combina, en una misma solución, las siguientes cualidades [4]:

- **Plataforma realmente abierta.** Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y "customizar" el sistema sin pagar royalties.

- **Adaptable a cualquier tipo de hardware.** Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día podemos encontrar relojes, cámaras, electrodomésticos y gran variedad de sistemas empotrados que se basan en este sistema operativo. Este hecho tiene sus evidentes ventajas, pero también va a suponer un esfuerzo adicional al programador. La aplicación ha de funcionar correctamente en dispositivos con gran variedad de tipos de entrada, pantalla, memoria, etc. Esta característica contrasta con la estrategia de Apple. En iOS tenemos que desarrollar una aplicación para iPhone y otra diferente para iPad

- **Portabilidad asegurada.** Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en una gran variedad de dispositivos, tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.

- **Arquitectura basada en componentes inspirados en Internet.** Por ejemplo, el diseño de la interfaz se hace en XML, lo que permite que una misma aplicación se ejecute en un móvil de pantalla reducida o notebook.

- **Filosofía de dispositivo siempre conectado a Internet.** Una clara aplicación del IdC. Gran cantidad de servicios incorporados en el propio dispositivo por el hecho de emplear librerías estándar Android Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.

- **Aceptable nivel de seguridad en cuanto al dispositivo se refiere.** Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet . . .)

- **Optimizado para baja potencia y poca memoria.** Por ejemplo, Android utiliza la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.

- **Alta calidad de gráficos y sonido.** Gráficos vectoriales suavizados, animaciones inspiradas en Flash, gráficos en 3 dimensiones basados en OpenGL. . .

- **Facilidad de instalación del ejecutable** de la aplicación de interés en cualquier dispositivo que disponga del sistema operativo Android. El archivo .apk basta para hacer funcionar la aplicación en el terminal.
- **Fomenta activamente la retroalimentación.** Android no solo cuenta con la comunidad más grande a nivel mundial de desarrolladores sino también el mayor movimiento de estos con multitud de eventos, concursos, competiciones y reuniones así como múltiples vías de comunicación como foros y chats oficiales para fomentar la participación y la colaboración para encontrar mejoras e ideas para futuras versiones.
- Android tiene sistema de multitarea inteligente, capaz de gestionar varias aplicaciones abiertas a la vez dejando en suspensión aquellas que no se utilicen y cerrarlas en caso de resultar ya inútiles para evitar un consumo de memoria.

Por otra parte, pensando sobre el pragmatismo del desarrollo a realizar, influyó en gran manera la cantidad de usuarios que hoy en día disponen del sistema operativo Android.

Esto es importante ya que a la hora de ofrecer la aplicación a un número elevado de empleados de una corporación, o usuarios de algún recinto público o privado, es conveniente realizarlo sobre una plataforma con la que los mismos se sientan familiarizados.

Sin embargo, en los últimos, años el sistema operativo Android, así como el número de aplicaciones de muy diversa índole (recreativa y visual, pero también laboral y corporativa) están experimentando un gran crecimiento entre los usuarios de smartphones y tablets.

En gran parte, esto está siendo debido a las características previamente citadas, así como la capacidad de las empresas de desarrollar aplicaciones sobre esta plataforma de una forma sencilla y efectiva.

En la Figura 3.2.2 se muestra, en procentaje, la cuota de mercado de los diferentes sistemas operativos de dispositivos móviles en el año 2011, 2012 y 2013.

Modelo de Android

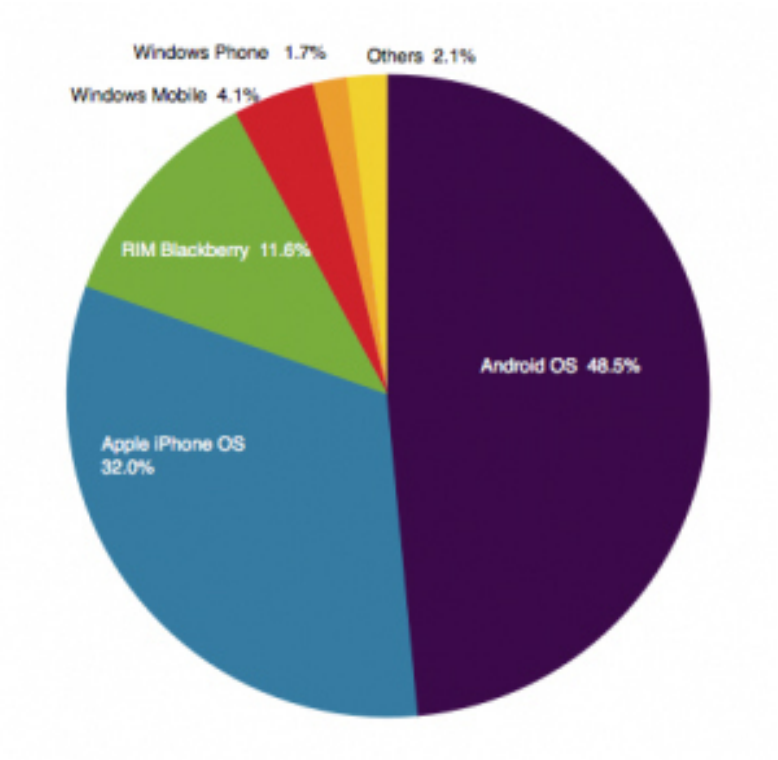

Figura 3.5: Gráfico comparativo de sistemas operativos móviles [5]

En Android se utiliza el patrón de arquitectura llamado Modelo Vista Controlador (MVC) cuya principal bondad consiste en separar los datos de una aplicación, la interfaz de usuario y la lógica de negocios en tres componentes distintos que se relacionarán para al final tener como resultado nuestra aplicación.

De esta forma es posible seccionar de forma más fácil el equipo de trabajo y que los programadores se dediquen a desarrollar lo componentes de tal forma que se construyan módulos o librerías con funcionalidades específicas que incluso podrían reutilizarse en proyectos posteriores y no simplemente en el proyecto actual.

Para lograr esto, el diseño de la arquitectura de una aplicación juega un papel importante , así como la capacidad de abstracción que se tenga en la aplicación. Por ejemplo, un usuario podría desarrollar una librería que permitiera hacer conexiones con la API de Twitter. En las clases correspondientes podría diseñar todas las funciones que llamen a los protocolos de Twitter:

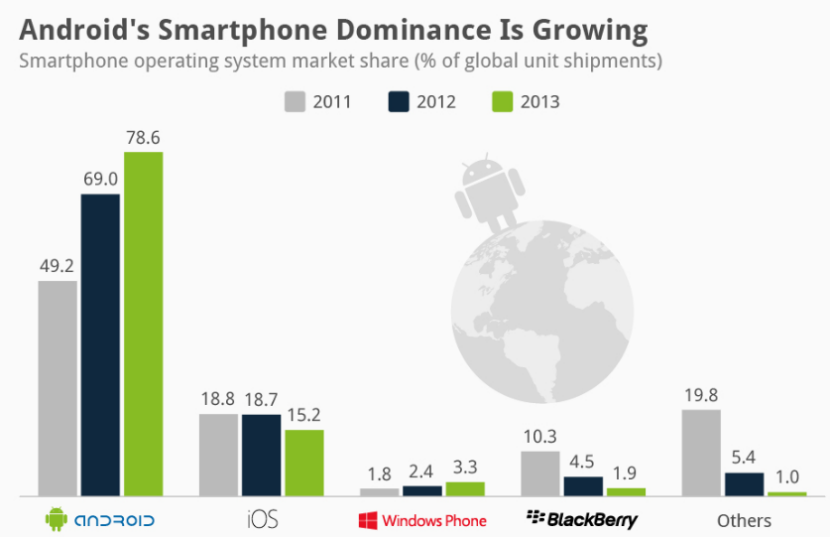

Figura 3.6: Cuota de mercado en dispositivos móviles [5]

realizar un tuit, recuperar información de tuits, followers, trending topics, etc. Esta librería, con un buen diseño y abstracción podría ser útil para todos los proyectos que necesiten conectarse con Twitter.

A continuación se ofrece una definición simplificada de cada uno de los componentes de la arquitectura MVC.

- **Modelo.** Se refiere con modelo a las representaciones que se construyen basadas en la información con la que operará la aplicación. En Java, el modelo viene siendo análogo a los beans que tienen la particularidad de ser reutilizables, haciendo las aplicaciones **escalables**. En esta parte del modelo también juega la decisión de qué modelo para almacenar información se utilizará: Bases de datos, Web Services, Archivo de Properties, Almacenamiento Interno del Dispositivo Móvil.

  Es decir, responde a lo siguiente:

  1. Contiene el núcleo de la funcionalidad (dominio) de la aplicación.
  2. Encapsula el estado de la aplicación.
  3. No sabe nada / independiente del Controlador y la Vista

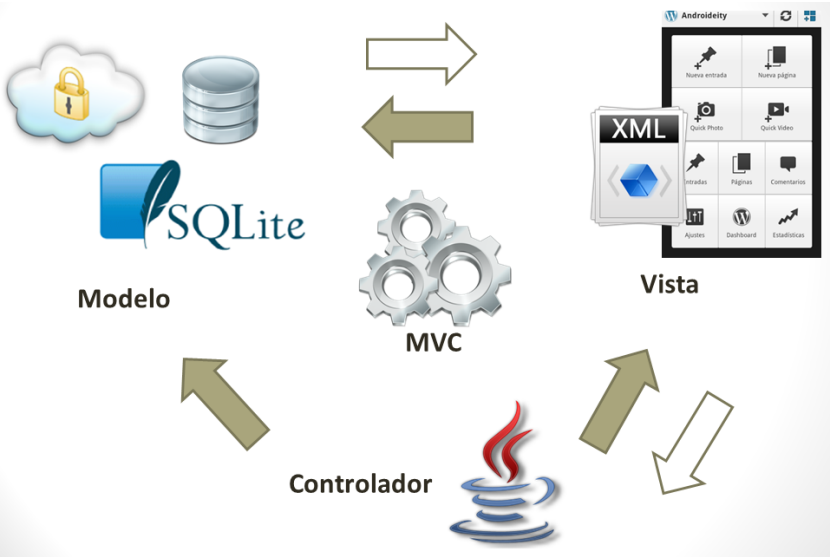

Figura 3.7: Esquema conceptual MVC

- **Vista.** La vista es la interfaz con la que va a interactuar el usuario. En Android, las interfaces se construyen en XML. Una analogía válida es comparar esta parte de la arquitectura MVC con el desarrollo web con CSS. Se construye un esqueleto en XML que equivale al HTML de un sitio web. Posteriormente, con ayuda de estilos, que también se escriben en XML, se le da formato de colores, posiciones, formato, etc. a dicho esqueleto.

  Cumple las siguientes funciones:

  1. Es la presentación del Modelo.
  2. Puede acceder al Modelo pero nunca cambiar su estado.
  3. Puede ser notificada cuando hay un cambio de estado en el Modelo.

- **Controlador.** Componente que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno.

  El controlador es responsable de:

1. Recibir los eventos de entrada (touch en pantalla táctil, un cambio en un campo de texto, etc.).
2. Contiene reglas de gestión de eventos, del tipo 'SI Evento Z, entonces Acción W'. Estas acciones pueden suponer peticiones al modelo o a las vistas.

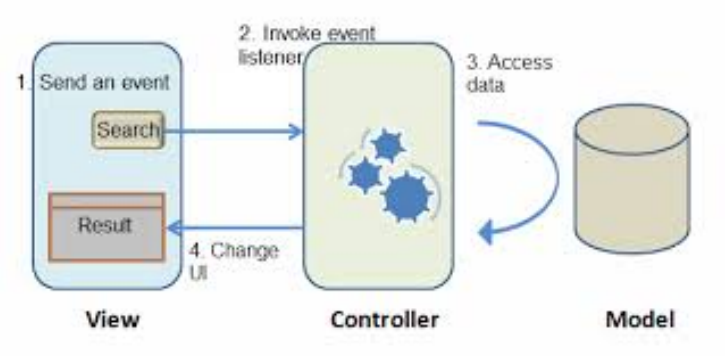

Figura 3.8: Arquitectura Modelo-Vista-Controlador

En cuanto al flujo que sigue una aplicación, a partir de los conceptos de MVC son:

1. Comienza cuando el usuario interactúa con la aplicación, la interfaz actual es la vista.
2. El controlador recibe la notificación de la acción solicitada.
3. El modelo es llamado para ser modificado.
4. El controlador nuevamente llama a la vista correcta que desplegará la situación de la aplicación actualizada.
5. El usuario ya dispone de la nueva interfaz para seguir interactuando con la aplicación y volver a iniciar el ciclo cuando solicite otra acción.

   En la sección de Implementación de este proyecto se hará referencia a los diferentes componentes de esta arquitectura relacionándolos con los diferentes elementos de la codificación final.

Componentes de Android

Las aplicaciones en Android tienen su propio entorno de ejecución:

- Cada aplicación se ejecuta en su propio proceso Linux. El sistema lo crea cuando ejecutamos la aplicación y lo destruye cuando no se use pasado un rato o cuando el Sistema necesite recursos para otra aplicación.
- Cada proceso se ejecuta en su propia máquina virtual, de esta manera está aislada del resto. De esta forma ante cualquier fallo en la aplicación solo afecta a su máquina virtual, no al resto.
- A cada aplicación se le asigna un identificador de usuario (uid) distinto. Los permisos de los archivos que refieren a la aplicación (caché, datos etc) son solo accesibles por dicho usuario. Es posible asignar un mismo uid a dos aplicaciones para que compartan una misma máquina virtual y recursos.

Este proceso se produce mediante el intercambio de acciones entre diferentes componentes.

Una de las características principales de Android es la reutilización de componentes de una aplicación por otra.

Por ejemplo, imaginemos que estamos desarrollando una aplicación que almacena datos de libros junto con una fotografía de su portada. En lugar de tener que escribir el código para capturar o seleccionar la imagen de la portada, podemos pasar el control a la aplicación de la cámara del teléfono, o a la galería, así una vez tomemos una foto o seleccionemos una imagen de la galería se nos devuelve el control a nuestra aplicación con la imagen seleccionada.

Para poder realizar estas operaciones, estamos obligados a dividir nuestras aplicaciones en módulos independientes que solo realicen una tarea concreta.

Veamos ahora otro ejemplo, muchos terminales tienen la opción de compartir algo en las redes sociales, por ejemplo Twitter, un módulo claramente definido de esta aplicación es por ejemplo la opción de "enviar un mensaje o tweet", si seguimos la filosofía de dividir nuestras aplicaciones en módulos, la función de enviar un mensaje sería una actividad independiente que recibe como parámetro el mensaje a enviar, si no recibe parámetro se mostrará el formulario para escribirlo. Dicha actividad usará la API de Twitter para

enviar el mensaje y finalmente cerrará la actividad devolviendo el control a la aplicación que la llamó.

De esta forma, y con los filtros adecuados en el AndroidManifest.xml, cada aplicación que quiera compartir algo en twitter llamará a esta actividad pasandole como parámetro el mensaje. Con esto llegamos a la conclusión de que las aplicaciones Android no tienen un punto de entrada y otro de salida, podemos definir todos los que necesitemos.

Para realizar todas estas operaciones, Android proporciona cuatro tipos de componentes básicos [6]:

Actividades

Son las encargadas de mostrar la interfaz de usuario e interactuar con él. Responden a los eventos generados por el usuario (pulsar botones etc). Heredan de la clase Activity. El aspecto de la actividad se aplica pasando un objeto View (Encargado de dibujar una parte rectangular en la pantalla, pueden contener más objetos View, además todos los componentes de la interfaz (botones, imágenes etc) heredan de View) al método Activity.setContentView(), que es el método encargado de dibujar la pantalla. Normalmente las vistas ocupan toda la pantalla, pero se pueden configurar para que se muestren como flotantes. Las actividades también pueden llamar a componentes que se mostrarán sobre su View (como diálogos o menús).

Gracias estas actividades, Android permite una sencilla reutilización de componentes y comunicación ente aplicaciones, siempre sujetas a ciertas medidas de seguridad, que facilita, por ejemplo, la actualización o sustitución de componentes por parte del usuario, resultando un sencillo y efectivo método para utilizar novedades o introducir mejoras en el software.Por cada pantalla distinta hay una actividad distinta, normalmente las aplicaciones tienen una actividad fijada como punto de entrada.

A continuación se muestra el ciclo de vida de una Actividad en Android. Es una cuestión importante ya que en base a este ciclo han de tomarse decisiones a la hora del desarrollo de la aplicación que afectan directamente a su funcionamiento y a su relación con los aspectos de geolocalización referentes a este proyecto.

El ciclo de vida de las aplicaciones de Android no se trata únicamente de abrir y cerrar a gusto del usuario, si no que las mismas, una vez iniciadas,

permanecen cargadas en memoria siempre que se disponga de recursos para ello. En caso contrario el propio sistema operativo se encargará de destruirlas definitivamente. Dicho ciclo de vida se rige por las llamadas a los métodos onCreate, onStart, onResume, onPause, onStop, onStop, onDestroy y onRestart. En la figura 3.9 puede entenderse este flujo de forma más intuitiva.

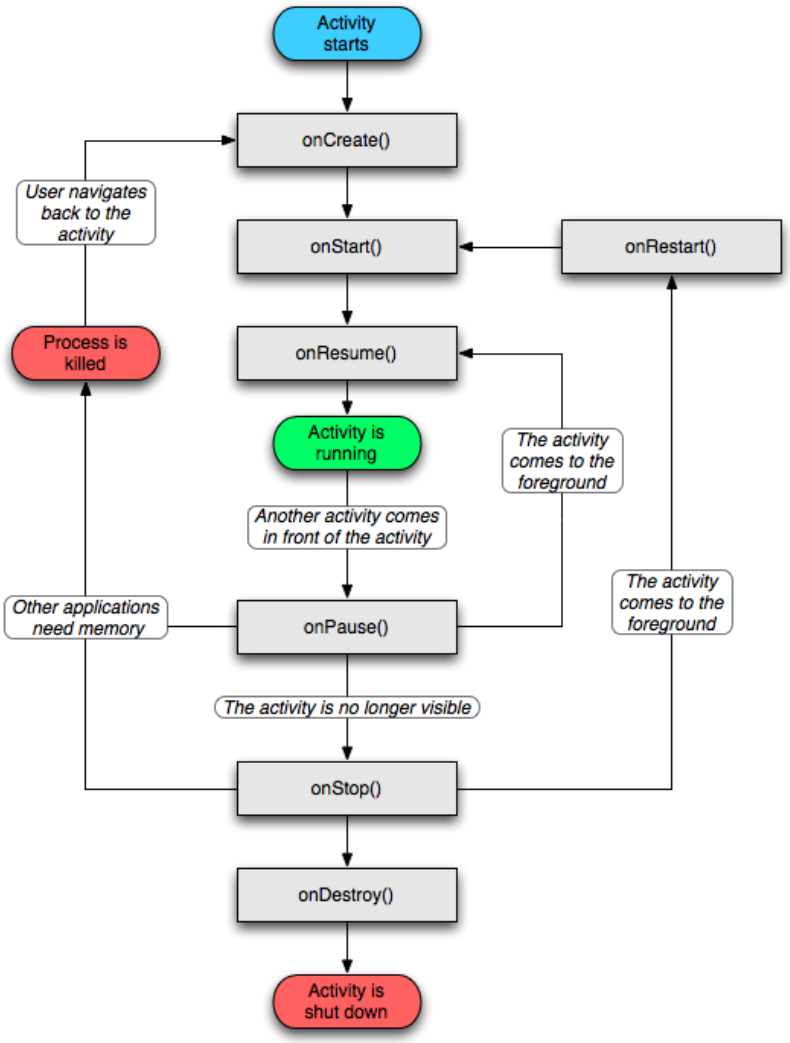

Figura 3.9: Ciclo de vida de una Actividad en Android

Servicios

No tienen interfaz visual y se ejecutan en segundo plano, se encargan de realizar tareas que deben continuar ejecutándose cuando nuestra aplicación no está en primer plano. Todos los servicios extienden de la clase Service.

Es posible lanzar o conectar con un servicio en ejecución con la interfaz que proporciona la clase Service. Los servicios disponen de un mecanismo para ejecutar tareas pesadas sin bloquear la aplicación ya que se ejecutan en un hilo distinto.

Receptores de mensajes de distribución(BroadcastReceiver)

Reciben un mensaje y reaccionan ante él. Heredan de la clase BroadcastReceiver. No tienen interfaz de usuario, pero pueden lanzar Actividades como respuesta a un evento o usar notificaciones para informar al usuario.

Android habitualmente lanza muchas notificaciones de sistema (llamadas entrantes, nuevos correos, nuevos sms etc). Por ejemplo, una aplicación de correo electrónico tendría un BroadcastReceiver escuchando el mensaje de nuevo correo, que lanzaría el servicio cada vez que detectara uno. Cuando esto sucediera, se mandaría un aviso a la barra del sistema para alertar al usuario.

Proveedores de contenido (Content Providers)

Ponen un grupo de datos a disposición de distintas aplicaciones, extienden de la clase ContentProvider para implementar los métodos de la interfaz, pero para acceder a esta interfaz se ha de usar una clase llamada ContentResolver. Con esta clase se permite acceder al sistema de ficheros, bases de datos SQLite o cualquier otra fuente de datos unificada. Un lector de correo podría disponer de un ContentProvider para acceder a la bandeja de entrada y los datos del mensaje.

3.3. Implementación

3.3.1. Elementos de Android utilizados

AndroidManifest.xml

Android Manifest archivo más importante en un proyecto de programación sobre Android, y que debe estar presente en la raíz del proyecto: AndroidManifest.xml.

Dicho archivo es generado automáticamente y modificable gráficamente o mediante codificación directa. El archivo presenta información esencial sobre la aplicación al sistema operativo Android. Información necesaria para que pueda ejecutar la aplicación.

Principales tareas que realiza AndroidManifest.xml:

- Utiliza el nombre de paquete Java como identificador único de la aplicación.
- Describe los componentes de la aplicación: Actividades, servicios, proveedores de contenido.
- Para ello utiliza el nombre de las clases que implementan cada uno de estos componentes y publica sus capacidades. Esto permite al sistema operativo conocer qué componentes tiene y bajo qué condiciones pueden ser lanzados.
- Especifica qué permisos tiene la aplicación para acceder a partes protegidas del API que proporciona el sistema Android.
- Declara la mínima versión del sistema operativo en el que funcionará la aplicación.
- Indica las librerías que utiliza el proyecto y por lo tanto tienen que ser empaquetadas al crear la aplicación.
- Permite declarar una clase 'Instrumentation' que sirve para monitorizar la interacción de la aplicación con el sistema. Esta declaración solo estará presente mientras se desarrolla y prueba la aplicación. Ya que será borrada antes de que la aplicación se vaya a publicar.

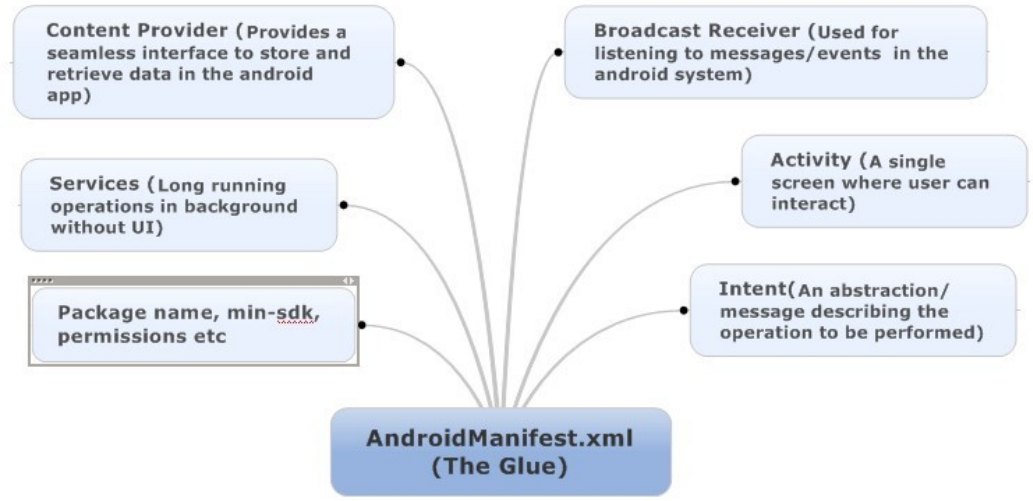

Figura 3.10: Visión global de AndroidManifest

Con la siguiente se pretende ofrecer una visión general de las funciones de AndroidManifest.xml.

Partes de AndroidManifest.xml

A continuación se muestra un ejemplo básico de AndroidManifest.xml y se explica cada una de sus partes:

Cosas a tener en cuenta:

- El elemento padre (manifest) del archivo debe de contener una declaración del espacio de nombres y del nombre que asigna al paquete que forma la aplicación.
- Cada manifiesto incluye un único elemento que contendrá información básica para la aplicación como el icono, nombre o tema que utiliza.
- Cada una de las actividades (controladores (Controlador en la arquitectura MVC explicada anteriormente) de cada pantalla de la interfaz) que aparecerán en la aplicación deberán de aparecer en el manifiesto.

Descripción de los elementos básicos:

1. **Etiqueta manifest**

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.loginandregisterapi"
  android:versionCode="1"
  android:versionName="1.0" >
  <uses-sdk
    android:minSdkVersion="10"
    android:targetSdkVersion="17" />
  <uses-permission android:name="android.permission.INTERNET"/>
  <application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name="com.example.loginandregisterapi.LoginActivity"
      android:label="@string/app_name" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

Figura 3.11: Ejemplo de AndroidManifest

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.loginandregisterapi"
  android:versionCode="1"
  android:versionName="1.0" >
```

Figura 3.12: Etiqueta manifest

Este el elemento raíz del manifiesto y sus dos atributos principales y obligatorios son:

- xmlns:android: Define el espacio de nombres de Android y siempre debe de ser el mismo -
- package: El nombre del paquete define la aplicación y actúa como

identificador único de la misma. Por lo que si has publicado una aplicación con un nombre y luego se cambia, los usuarios de la primera versión no podrán actualizar a la siguiente.

2. **Etiqueta uses-sdk**

```
<uses-sdk
  android:minSdkVersion="10"
  android:targetSdkVersion="17" />
```

Figura 3.13: Etiqueta uses-sdk

Elemento (etiqueta) de segundo nivel obligatorio que determina la compatibilidad de la aplicación con una o más versiones del sistema operativo. Esta compatibilidad viene expresada en base al nivel de API del sistema Android que soporta. Sus dos atributos principales son:

- android:minSdkVersion (obligatorio): Determina el mínimo nivel de API que debe de tener el sistema operativo Android que pretenda ejecutar la aplicación. El sistema evitará que la aplicación se instale en un sistema que tenga un nivel de API inferior del especificado.
- android:targetSdkVersion (opcional): Si no se especifica se toma el valor de minSdkVersion. Determina el nivel del API con el que fue construida la aplicación. Por lo que se espera que tome ventajas del nivel de API especificado pero es totalmente retro-compatible con versiones antiguas hasta la indicada mediante minSdkVersion.

3. **Etiqueta uses-permission**

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

Figura 3.14: Etiqueta uses-permission

Etiqueta opcional de segundo nivel que sirve para indicar un permiso necesario que requiere la aplicación para acceder a alguna parte protegida del API que proporciona el sistema Android.

Esta declaración alertará a los usuarios que la aplicación utilizará ciertos permisos. El único atributo disponible y obligatorio es android:name. El cual indica un permiso que necesita la aplicación.

4. **Etiqueta application**

Etiqueta de segundo nivel que contendrá otras etiquetas que declararán cada uno de los componentes de la aplicación. Además permite atributos que pueden afectar a todos los citados componentes de la aplicación. Solo se puede declarar una vez este elemento en el manifiesto. Y admite multitud de atributos aunque los más importantes y utilizados son los siguientes:

```
<application
  android:allowBackup="true"
  android:icon="@drawable/ic_launcher"
  android:label="@string/app_name"
  android:theme="@style/AppTheme" >
```

Figura 3.15: Etiqueta application

- android:allowBackup(opcional). Valor por defecto a true. Indica que si se hace un backup del sistema Android, las aplicaciones que tengan este valor a true se guarden junto con el backup del sistema.
- android:description: Descripción larga de la aplicación y sus funcionalidades.
- android:icon: identificador del recurso que será el icono de la aplicación.Los recursos crearán un identificador en la clase R para facilitar sus acceso en el código fuente de la aplicación.
- android:label. Identificador de la cadena de texto que dará nombre a la aplicación y que será el que verá el usuario en el sistema operativo.
- android:permission. Especificamos el nombre de un permiso que será necesario si otras aplicaciones hacen uso de partes de la aplicación. También es posible definir el permiso necesario para cada

una de las actividades (siguiente etiqueta) utilizando este atributo en cada una de ella.

- android:theme. Identificador al recurso que especifica el tema por defecto de todas las actividades de la aplicación. Las actividades pueden sobreescribir individualmente el tema general con sus respectivos atributos android:theme.

5. **Etiqueta activity**

```
<activity
  android:name="com.example.loginandregisterapi.LoginActivity"
  android:label="@string/app_name" >
</activity>
```

Figura 3.16: Etiqueta activity

Etiqueta de tercer nivel y que es uno de los sub-elementos de la etiqueta application. Una actividad es el controlador (arquitectura MVC) que va a interactuar con una pantalla de la interfaz gráfica. Y por lo tanto debemos de especificar cada actividad del proyecto con su etiqueta activity correspondiente. Si una actividad no se encuentra especificada en el manifiesto, ésta no podrá lanzarse. Con el consiguiente error en la aplicación. Principales atributos:

- android:ExcludeForRecents(opcional). Si su valor es 'true' significa que la aplicación asociada con esta actividad no aparecerá en la lista de aplicaciones recientes del sistema Android,
- android:exported (opcional). Con este atributo es posible restringir a que esta actividad pueda ser lanzada por componentes de otra aplicación. Con esto se limita la exposición a otras aplicaciones. Por lo tanto si el valor es 'true' se acepta dicho uso.
- android:icon (opcional). Si se desea que el icono que aparece junto a la etiqueta superior de la pantalla de la interfaz gráfica sea distinto al icono general de la aplicación, se puede especificar con el identificar del recurso de tipo imagen que se desea.
- android:label (obligatiorio). Etiqueta que verá el usuario en la parte superior (junto al icono) cuando la actividad muestre la

pantalla gráfica correspondiente. El valor será una referencia a un identificar de recurso de tipo cadena.

- android:name (obligatorio). Este atributo indica el nombre de la clase (ruta completa) que implementa la actividad.
- android:screenOrientation (opcional). Con este atributo indicamos que orientación tendrá la pantalla de la interfaz gráfica que controla nuestra actividad. Los valores más usados son 'portrait' (vertical), 'landscape' (horizontal) o el valor por defecto 'unspecified' con el que el dispositivo elige la orientación.

6. **Etiqueta intent-filter**

```
<activity
  android:name="com.example.loginandregisterapi.LoginActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Figura 3.17: Etiqueta intent-filter

Etiqueta que se sitúa como sub-elemento de la anterior etiqueta (activity). Esta etiqueta sirve para agrupar el número de acciones que concretarán el ámbito en el que se va a ejecutar la actividad. Las actividades pueden declarar el tipo de acciones que pueden llevar a cabo y los tipos de datos que pueden gestionar.

Si una actividad no tiene declarado ningún 'intent filter', se considera una opción para cualquier 'intent' con una acción. Por lo tanto es importante definir qué acciones puede manejar la actividad.

Hay que tener en cuenta que solo es necesario declarar filtros cuanto los 'intents' son lanzados de forma implícita. Ya que es el sistema Android el encargado de buscar la actividad adecuada. En los 'intents' lanzados de forma explícita ya se especifica la actividad que se quiere ejecutar.

7. **Etiqueta action**

```
<activity
  android:name="com.example.loginandregisterapi.LoginActivity"
  android:label="@string/app_name" >
  <intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
  </intent-filter>
</activity>
```

Figura 3.18: Etiqueta caption

Una acción que el 'intent-filter' soporta. Las acciones son cadenas de texto estándar que describen lo que la actividad puede hacer. El único y obligatorio atributo es android:name. En el cual indicaremos el nombre de la acción. En el anterior ejemplo, indicamos que esta es la actividad principal de la aplicación y por lo tanto la que controlará el inicio de la aplicación.

Imaginemos que una actividad especifica las siguientes acciones:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <action android:name="android.intent.action.EDIT" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
</intent-filter>
```

Figura 3.19: Acciones de una actividad de ejemplo

Esto indica que la actividad está disponible para ver y editar elementos del tipo vnd.android.cursor.dir/vnd.google.note

8. **Etiqueta category**

Básicamente indica si la actividad va a ser lanzada desde el lanzador de aplicaciones, desde el menú de otra aplicación, directamente desde otra actividad, etc.

9. **Etiqueta data**

   Mediante esta etiqueta añadiremos una especificación de datos para las acciones especificadas. Puede ser un tipo de datos o una URI.

En este proyecto se ha configurado el archivo AndroidManifest conforme los diferentes componentes han ido siendo implementados: Actividades, Servicios.. Basándonos en el estudio que se acaba de realizar de este archivo, se han podido realizar las diferentes acciones sobre el mismo para garantizar el correcto funcionamiento de la aplicación. Debido a los requerimientos de la aplicación, ha sido necesario establecer los siguientes permisos en el archivo AndroidManifest.xml.

- Lectura del estado del dispositivo móvil
- Acceso a internet
- Lectura y escritura en esctructuras de almacenamiento del dispositivo
- Acceso a sistemas de geolocalización

La codificación exacta del mismo se encuentra disponible al final de este documento, en el Anexo 1.

SharedPreferences

El archivo ShraredPreferences es propio de los dispositivos Android y posee la funcionalidad inicial de almacenar un valor y compartirlo con otros métodos o actividades.

Una de las características más interesantes de este método de almacenamiento es que este valor continuará grabado y recordado aunque se cierre la aplicación o se reinicie o apague el dispositivo móvil.

Por ejemplo, gran cantidad de aplicaciones que instalamos en nuestros teléfonos tienen la particularidad de que si es la primera vez que se ejecuta ofrece una serie de opciones, y si ya lo ha hecho con anterioridad ofrece otras diferentes. Esto puede realizarse con SharedPreferences.

La clase SharedPreferences de Android permite a los desarrolladores usar sus métodos, funciones y procedimientos para leer y escribir en un fichero

de configuración de aplicación Android. Las aplicaciones Android permiten guardar sus datos de configuración en uno (o varios) ficheros de tipo XML, ubicados dentro de la carpeta de instalación de la aplicación, en la subcarpeta shared_prefs.

Esta clase proporciona opciones para permitir elegir el método de seguridad para el acceso a estos archivos de configuración. Las posibilidades son:

- *MODE_PRIVATE*: sólo la aplicación podrá leer y escribir datos de configuración en el fichero XML.
- *MODE_WORLD_READABLE*: sólo la aplicación podrá escribir datos de configuración, el resto de aplicaciones del dispositivo Android podrá leer los datos de configuración pero no modificarlos.
- *MODE_WORLD_WRITEABLE*: todas las aplicaciones del dispositivo Android podrán leer y escribir datos de configuración en el fichero XML.
- *MODE_MULTI_PROCESS*: no suele usarse, puede servir cuando la aplicación tiene múltiples procesos, para comunicación entre ellos.

Los métodos que proporciona la clase SharedPreferences para leer los datos del fichero de configuración XML son:

- contains: comprueba si existe una preferencia en el fichero XML.
- getAll: obtiene todos los valores de todas las preferencias guardadas en el fichero XML.
- getBoolean: obtiene una preferencia de tipo Booleano (true, false).
- getFloat: obtiene una preferencia de tipo Float (numérico).
- getInt: ontiene una preferencia de tipo Int (entero).
- getLong: obtiene una preferencia de tipo Long (entero largo).
- getString: obtiene una preferencia de tipo String (texto).
- getStringSet: obtiene una preferencia de tipo StringSet (conjunto de cadenas).

- registerOnSharedPreferenceChangeListener: registra una devolución de llamada que se invoca cuando ocurre un cambio en las preferencias.
- unregisterOnSharedPreferenceChangeListener: anula el registro de una devolución de llamada anterior

En nuestro caso, los casos en los que hemos empleado SharedPreferences para los siguientes elementos:

1. Almacenamiento del periodo de refresco de la posición geográfica: Acceso con getLong()
2. Almacenamiento de la ID del usuario: Acceso con getString().
3. Almacenamiento de los roles. Acceso con getString()

La evidencia del desarrollo sobre SharedPreferences se encuentra más adelante en esta memoria.

Los métodos que proporciona la clase SharedPreferences para guardar los datos en el fichero de configuración ó preferencias XML son:

- apply: guarda los cambios realizados en las preferencias cuando el objeto SharedPreferencias está en modo edición.
- clear: elimina todos los valores de las preferencias.
- commit: guarda los cambios realizados en las preferencias.
- putBoolean: guarda un valor de tipo booleano (true, false).
- putFloat: guarda un valor de tipo Float.
- putInt: guarda un valor de tipo Int.
- putLong: guarda un valor de tipo Long.
- putString: guarda un valor de tipo String.
- puntStringSet: guarda un valor de tipo StringSet.
- remove: elimina el valor de una preferencia.

En el archivo SharedPreferences los registros siguen el patrón clave-valor, es decir, cada una de ellas estará compuesta por un identificador único (p.e. nombre o email) y un valor asociado a dicho identificador (p.e.prueba o modificado@email.com), y el fichero XML resultante de usar SharedPreferences tendrá el siguiente formato:

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="nombre">prueba</string>
    <string name="email">modificado@email.com</string>
</map>
```

Figura 3.20: Ejemplo fichero XML de SharedPreferences

Toast

Un toast muestra un mensaje en un pequeño popup, cuyo tamaño no excede el contenido del mensaje y no interrumpe la ejecución de la aplicación.

Este tipo de mensjes son muy útiles cuando se desea mostrar cualquier mensaje informativo, donde no se necesita la interacción del usuario, por ejemplo si guardamos los datos de un usuario en la base de datos de una aplicación.

Un toast se muestra durante unos segundos al usuario para luego volver a desaparecer automáticamente sin requerir ningún tipo de actuación por su parte, y sin recibir el foco en ningún momento (o dicho de otra forma, sin interferir en las acciones que esté realizando el usuario en ese momento). Aunque son personalizables, aparecen por defecto en la parte inferior de la pantalla, sobre un rectángulo gris ligeramente translúcido. Por sus propias características, este tipo de notificaciones son ideales para mostrar mensajes rápidos y sencillos al usuario, pero por el contrario, al no requerir confirmación por su parte, no deberían utilizarse para hacer notificaciones demasiado importantes.

Su utilización es muy sencilla, concentrándose toda la funcionalidad en la clase Toast. Esta clase dispone de un método estático makeText() al que se debe pasar como parámetro el contexto de la actividad, el texto a mostrar, y la duración del mensaje, que puede tomar los valores LENGTH_LONG o LENGTH_SHORT, dependiendo del tiempo que se quiera que la notificación aparezca en pantalla. Tras obtener una referencia al objeto Toast a través de este método, ya sólo resta mostrarlo en pantalla mediante el método show().

A continuación se muestra un ejemplo de un Toast por defecto.

En el caso de este proyecto se utiliza la funcionalidad Toast para infor-

Figura 3.21: Ejemplo de mensaje Toast

mar al usuario de las acciones que realiza debidas a su interacción con la herramienta.

Servicios

Ya se ha comentado las características principales de los servicios en el contexto de desarrollo Android en la sección de Elección de la plataforma, sin embargo, debido a la gran importante que ha representado el uso de los servicios en el proceso de desarrollo de la aplicación tuvo que incidirse sobre ellos de una forma más profunda para conocerlos y utilizarlos mejor.

En Android los servicios pueden servir para dos funciones diferentes: La primera de ellas consiste en crear un servicio para indicar al sistema que van a realizarse implementaciones que incluirán operaciones que han de ejecutarse en segundo plano, durante un periodo más o menos duradero. Se inician mediante el método startService(). Estos servicios permanecerán en ejecución a no ser que les llegue un aviso explícito de detención. La segunda de las funcionalidades consiste en la implementación de la posibilidad de que una aplicación interactúe y se comunique con otras. Existirán métodos de intercomunicación entre ellas a partir de estos servicios. Se inician con el método bindService().

Cuando una aplicación usa bindService() tiene la intención de conectarse

con dicho servicio y usar alguno de los citados métodos de comunicación con otras aplicaciones.

Cada vez que un servicio es creado, el sistema instancia el servicio y llama al método onCreate(). El servicio habrá de codificar las operaciones necesarias en cada caso, y normalmente, ya que desea ejecutar un gran número de las mismas y durante un largo periodo de tiempo, creará un hilo adicional para que se ejecuten en el mismo, evitando así la posible interferencia con el resto de la aplicación o procesos del sistema operativo.

Por lo tanto, si el servicio necesita un uso intensivo de la CPU, como es nuestro caso en la obtención y distribución de posiciones geográficas, o puede quedar bloqueado en algún momento de la ejecución de dichas operaciones, es recomendable la creación de un hilo (thread) diferente para ejecutarlas.

Los servicios en Android siguen lo que se denomina ciclo de vida, que se muestra en la figura: 3.22

Como se ha comentado existen dos tipos de servicios según como hayan sido creados. Las funciones de estos servicios son diferentes y, por lo tanto, también su ciclo de vida. Si es iniciado a partir de startService(), la secuencia será:

1. Creación de servicio
2. Llamada a onCreate()
3. Llamada a onStartCommand()
4. El servicio continúa ejecutándose hasta que se llame a stopService()

Es interesante comentar que puede definirse un servicio de este tipo de dos formas, depende el comportamiento que haya de tener en situaciones de poca memoria del dispositivo móvil.

En este contexto los servicios se detienen. En función de lo que sucede después se divide en:

- $START\_STICKY$: En cuanto se cargue la batería lo suficiente que trate de reiniciarse el servicio

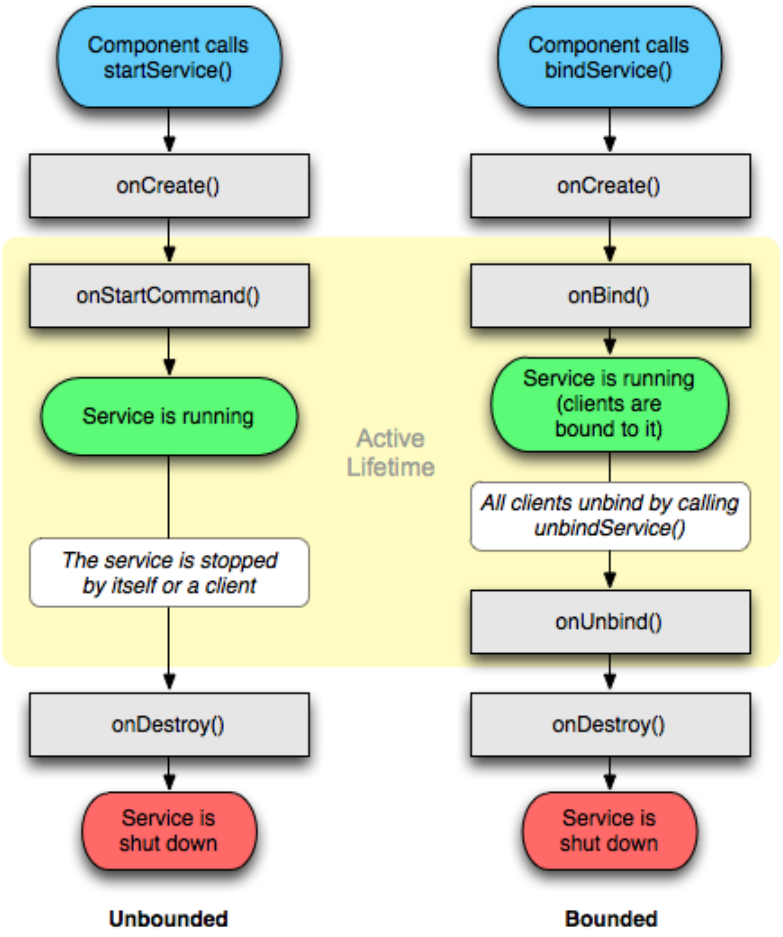

Figura 3.22: Ciclo de vida de un Servicio de Android

- *START_NOT_STICKY*: Que no se vuelva a reiniciar hasta que no se solicite explícitamente.

La política de los servicios en Android es la de otorgarles mayor prioridad que a los procesos de ejecución en segundo plano, pero menos que la actividad en la pantalla de usuario o la interacción con éste.

Si el tipo de proceso del que se trata es del que se arranca mediante bindService() la secuencia es la siguiente:

1. Comprobación que el proceso está en ejecución
2. En caso negativo, se crea con el método onCreate() sin llamar a onStartCommand()

3. Llamada a onBind()

4. Establecimiento de la conexión hasta que se llame a stopService()

En el caso de este proyecto, se ha usado un servicio, como se verá en la siguiente sección, para implementar la solicitud de posiciones geográficas a los proveedores de localización y a su vez enviarla al sistema central de GeoRBAC mediante HTTPS. Se ha utilizado para un correcto funcionamiento de la aplicación y para fomentar su escalabilidad, ya que es un hilo que puede requerir bastantes recursos computacionales, por tanto debe ejecutarse en un servicio para no interferir en el funcionamiento normal de la aplicación y el sistema operativo en el dispositivo móvil.

Hilos

Otro componente interesante de Android que se ha utilizado en el caso de uso de este proyecto está directamente relacionado con los servicios que acaban de estudiarse, y en concreto en su codificación en el proyecto actual: se trata de los hilos de ejecución.

Al iniciarse una aplicación en el sistema Android, éste arranca un hilo de ejecución: el hilo principal.

Este hilo es el que atiende y ejecuta los métodos básicos de las actividades y servicios que ya se han analizado onCreate(), onStartCommand(), etc.

Cuando se crean nuevas actividades/pantallas o servicios no se crea un nuevo hilo, tal y como se ha comentado, sino que es este hilo principal quien se encarga de todo.

Precisamente por esa razón es interesante crear hilos de ejecución secundarios que operen en background y no interfieran en la actividad del hilo principal.

Al ejecutarse en un contexto secundario no podrá acceder a los métodos de la vista de usuario o atender a los eventos e interacciones que éste genere, así como tampoco podrá modificar la interfaz que está observando. De esta forma no se bloquea en ningún caso el hilo principal, que puede seguir atendiendo los eventos de usuario.

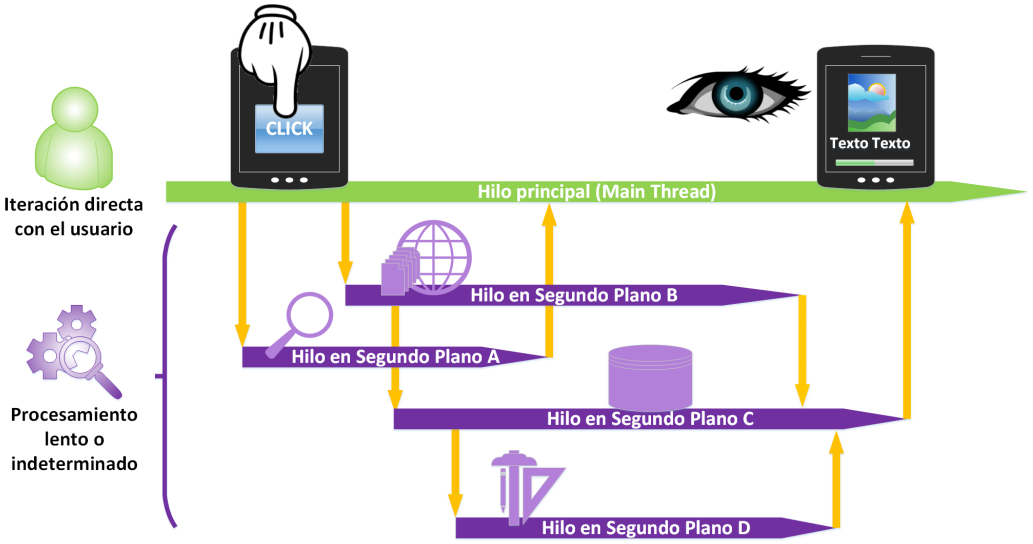

Figura 3.23: Ejemplo de utilidad de hilos de ejecución [7]

Es, en definitiva, recomendable utilizar hilos de ejecución secundarios en los que casos que haya que realizar muchas operaciones o que de ejecutarse en el hilo(thread) principal se corra el riesgo de bloqueo.

Se ha utilizado para este proyecto la forma de implementación basada en AsyncTask. Este tipo de implementación responde a la figura 3.24:

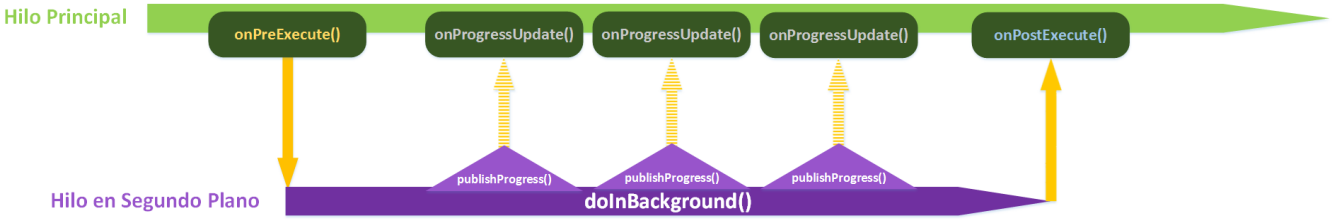

Figura 3.24: Ciclo de vida de AsyncTask [7]

De forma general, el funcionamiento de la programación sobre AsyncTask es el siguiente:

Al crearse el objeto de la clase AsyncTask se llama, para empezar, a su primer método onPreExecute() que se ejecuta sobre el hilo principal. Al terminar este AsyncTask crea un hilo secundario y ejecuta su trabajo dentro de

doInBackground(). Durante la ejecución en segundo plano es posible realizar llamadas al hilo principal (por ejemplo, para incrementar la barra de carga a medida que se ejecuta el hilo en segundo plano) desde doInBackground(), y con ayuda de publishProgress(), a un método sobreescrito llamadoonProgressUpdate(). No es obligatorio llamar a publishProgress(), pudiendo no llamarlo nunca con lo que nunca se entraría en onProgressUpdate(), o llamándolo las veces que se desee. Al terminar de ejecutarse doInBackground() se llama de inmediato a onPostExecute(), y se da por acabado a este hilo en segundo plano.

Menú de opciones

Los menús son uno de los componentes más conocidos de las aplicaciones Android. A través de ellos los usuarios pueden modificar parámetros de configuración o funcionamiento de su aplicación, a la vez que ofrecen una herramienta gráfica al mismo para interactuar con la lógica del backend.

Existen diferentes tipos de menús, basados en iconos, listas, etc. En el caso de este proyecto, el menú que se ha utilizado para interacción con el usuario ha sido el menú de opciones u OptionsMenu.

Este tipo de Menú es el primero y más conocido de los que dispone una Actividad. Aparece, en función de la versión de la aplicación y de la distribución de API del dispositivo que esté utilizando, en una de las esquinas de la pantalla, y es el que aparece cuando el usuario pulsa el botón MENU, si es que lo hay. Dentro de este menú se destacan dos tipos:

- **Menú de iconos:** Aquel que despliega una serie de iconos con un nombre relativo a ellos al pulsar el usuario la tecla MENU. Estos iconos son configurables por parte del programador, aunque el límite de número son seis iconos .
- **Menú Expandido.** Este muestra una lista vertical de elementos desplegados al pulsar los botones de más información del menú de iconos.

Al abrirse este menú de opciones, Android invoca al método onCreateOptionsMenu() de la clase Activity padre, a continuación sobreescribe es método en la actividad que hayamos desarrollado (la cual heredará de Activity) y crea un objeto de tipo Menu.

De forma similar a la definición de vistas o del manifiesto, se agregan opciones a dicho menú a través de etiquetas XML. Otra opción (menos recomendable, ya que altera ligeramente la concepción MVC) es creando las opciones directamente desde la codificación con el método add(). Este método añade un ítem al menú y permite añadirle característica a dicho elemento: icono, acceso rápido con combinación de teclas, etc.

El método add() posee varias implementaciones en función de sus parámetros de entrada (está sobrecargado), y la opción que se usa de forma habitual, coincidente con la que se ha empleado en este proyecto, es aquella con itemId como parámetro, que se refiere al número que identifica a la opción del menú que estamos creando. Este parámetro es empleado por el sistema al invocar la función onCreateOptionMenu() que se encarga de crear el menú de opciones.

Figura 3.25: Ejemplo de Menú de Opciones

Cuando el usuario interactúa con el menú de opciones y toca en la pantalla sobre alguno de los ítems, se llama al método onOptionsItemSelected(). A partir del objeto MenuItem que se genera, y con el método getItemId() se puede saber qué opción ha seleccionado el usuario, y a partir de ahí codificar una respuesta u otra en función de la naturaleza de la aplicación.

En este proyecto solo se ha utilizado la primera de las opciones, y la evidencia de su implementación y codificación se encuentra más adelante en esta memoria.

Diálogos

Al igual que en otros sistemas, un diálogo es una ventana que aparece por delante de la pantalla visualizada (activity) evitando que, mientras sea mostrado, la actividad acepte eventos. Los diálogos se utilizan para notificar al usuario tareas o acciones de corta duración, relativas a la actividad, tales como la introducción de datos de login, o como mostrar una barra de progreso.

Todos los diálogos heredan de la clase Dialog. La práctica común es utilizar alguna de las subclases ya definidas en Android o bien extender éstas o la propia clase Dialog, para crear un diálogo propio.

Los tipos de diálogos disponibles son:

- **Alertas.** Se trata de diálogos que implementan la clase AlertDialog, o una subclase que la extienda, y que podrán mostrar de cero a tres botones, listas de ítems seleccionables (vía checkboxes o radiobuttons), o un layout personalizado..
- **Diálogos de Progreso.** Son diálogos que muestran una barra (o un círculo) de progreso. Pueden disponer de botones. Se implementan con la clase ProgressDialog, que es una subclase de AlertDialog.
- **Calendario para escoger fecha.** Permiten seleccionar una fecha (día, mes, año)
- **Reloj para escoger hora.** Permiten seleccionar una hora (hora, minutos, am/pm)
- **Diálogos** que se pueden personalizar.

En particular en este proyecto se ha utilizado AlertDialog, para informar sobre los autores del proyecto global de GeoRBAC y para solicitar al usuario la introducción del periodo de refresco de la posición. Con esta información, que se almacenará en el fichero SharedPreferences, la aplicación solicitará al proveedor de geolocalización.

Estas clases definen el estilo y la estructura de un diálogo, pero la clase DialogFragment es la que se usa como contenedor del diálogo. Dicha clase proporciona todos los controles que nse necesitan para crear un diálogo y modificar su apariencia.

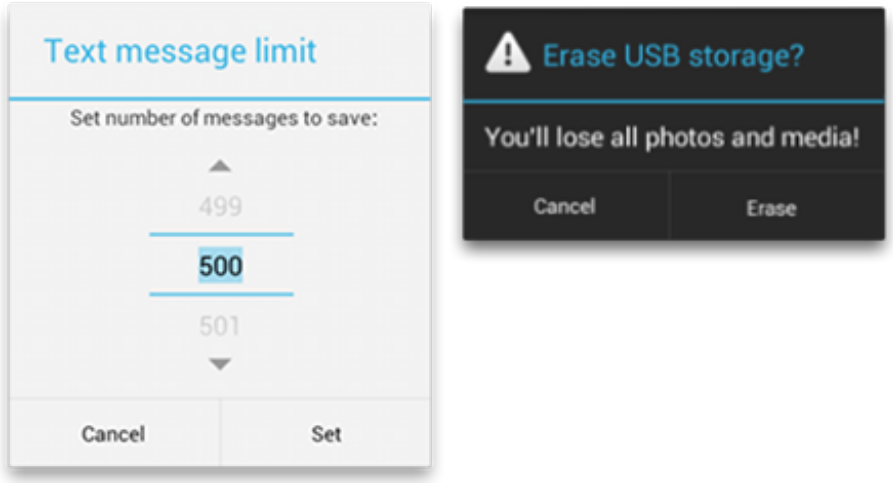

Figura 3.26: Ejemplos de AlertDialog

El uso de la clase DialogFragment asegura la correcta gestión de los eventos que puedan ocurrir, como por ejemplo pulsar el botón Atrás o rotar la pantalla. Además, usando dicha clase es posible reutilizar el interfaz de los diálogos como componente en un interfaz más grande.

La evidencia de uso de este elemento tecnológico de Android se encuentra más adelante en este documento.

Geolocalización

El principal problema a la hora de realizar aplicaciones para dispositivos móviles que requieran geolocalización es la imposibilidad de realizar una ubicación correcta debido a que el dispositivo no capta la señal de un número adecuado de satélites.

Este hecho se da en interiores de edificios pero también se puede dar en exteriores, por ejemplo, en ciudades con una densidad de edificaciones que apantalle la señal GPS o en situaciones en las que simplemente el dispositivo no dispone de un receptor GPS o prefiere no emplearlo.

La plataforma Android dispone de un interesante sistema de posicionamiento que combina varias tecnologías:

- Sistema de localización global basado en GPS. Este sistema solo funciona si disponemos de visibilidad directa de los satélites. Está sometido a la problemática que hemos comentado.

- Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios. Con este sistema Android trata de combatir la situación expuesta, ofreciendo una localización precisa en cualquier entorno.

A estos dos sistemas de posicionamiento se les denomina técnicamente Proveedores de Localización o Location Providers, en referencia a los proveedores de posición geográfica de los que se depende en cada caso.

En una aplicación es posible utilizar uno de los dos métodos o los dos a la vez, como es nuestro caso.

La aplicación desarrollada en este proyecto contempla que el usuario del dispositivo móvil pueda moverse tanto por exteriores con alta visibilidad de satélites GPS como en zonas más recónditas o interiores, principalmente.

Es relevante comentar el proceso completo de obtención de posiciones geográficas, ya que constituye un ítem importante en el proyecto actual, y se ha convertido en una de las grandes dificultades del mismo.

Proceso:

La aplicación consulta en primer lugar los proveedores disponibles mediante el método getAllProviders(), a partir de la clase LocationManager, tras haber accedido al servicio de localización proporcionado por el sistema Android mediante el método getSystemService (LOCATION_SERVICE).

```
LocationManager locManager = (LocationManager)getSystemService(LOCATION_SERVICE);
List<String> listaProviders = locManager.getAllProviders();
```

Figura 3.27: Consulta de proveedores de localización disponibles

Al realizar estas operaciones, se obtiene una lista de todos los Location Providers disponibles en ese momento para ese dispositivo móvil. A partir de esta lista resulta interesante poder acceder a parámetros de cada uno de los proveedores. Estos parámetros podrán después usarse para la selección del

proveedor al que solicitar posición.

Unos ejemplos de los valores a los que se puede acceder son:

- getAccuracy(): Consulta el valor de precisión horizontal de ese proveedor.
- getName(): Nombre del proveedor
- getPowerRequirement(): Requerimientos de potencia de señal recibida mínima
- hasMonetaryCost(): Valor verdadero/falso que inforama si la consulta de datos a ese proveedor repercutirá en un sobrecoste al usuario.
- requiresNetwork(): Es verdadero s el proveedor requiere acceso a una red de datos (p.e. Internet)
- requiresSatellite(): Es verdadero si el proveedor requiere acceso a un sistema de posicionamiento basado en satélite (p.e. GPS)

Un código de ejemplo de lo que se acaba de comentar se muestra en las siguientes líneas:

```
LocationManager locManager = (LocationManager)getSystemService(LOCATION_SERVICE);
List<String> listaProviders = locManager.getAllProviders();

LocationProvider provider = locManager.getProvider(listaProviders.get(0));
int precision = provider.getAccuracy();
boolean obtieneAltitud = provider.supportsAltitude();
int consumoRecursos = provider.getPowerRequirement();
```

Figura 3.28: Consulta de parámetros de los proveedores de la lista

En el ejemplo mostrado en la imagen 3.28 se accede al primer proveedor de la lista, get(0), y los parámetros consultados son la precisión e información sobre el consumo de recursos que supondrá al dispositivo.

El siguiente paso consiste en escoger el mejor proveedor de entre los que ha devuelto la lista anterior.

Para ello Android dispone de una clase auxiliar para la definición de parámetros de criba: Criteria. A continuación se muestra un ejemplo de la utilización de la clase Criteria con los parámetros establecidos de precisión y de necesidad de aporte del dato de Altitud.

```
Criteria req = new Criteria();
req.setAccuracy(Criteria.ACCURACY_FINE);
req.setAltitudeRequired(true);
```

Figura 3.29: Uso de la clase Criteria

Tras definir los criterios, se realiza la criba de entre los mostrados anteriormente a únicamente los que cumplen con los requisitos impuestos:

```
Criteria req = new Criteria();
req.setAccuracy(Criteria.ACCURACY_FINE);
req.setAltitudeRequired(true);
```

Figura 3.30: Criba de proveedores

El ciclo continúa con la comprobación de que los proveedores que cumplen dichos criterios se encuentran activos y disponibles para consultar la posición geográfica.

En este punto es en el que se selecciona qué tipo de proveedor de localización queremos escoger.

Existe un método, isProviderEnabled() que, en función de su parámetro de entrada escoge un proveedor de uno u otro tipo preferentemente:

- LocationManager.NETWORK_PROVIDER. Localización por la red telefónica o datos.
- LocationManager.GPS_PROVIDER. Localización por GPS.

En este momento ya se ha escogido un proveedor, y hay que proceder a obtener una posición válida. Sin embargo, Android no proporciona ningún

método intuitivo y rápido mediante el cual realizar esta consulta.

En la clase LocationManager existe el método getLastKnownLocation(String provider). Sin embargo este método puede ser engañoso:

- No devuelve la posición del momento actual, en el cual se llama a dicho método
- La invocación de este método no supone una orden de captura de nueva posición geográfica por parte del proveedor
- Simplemente devuelve la última posición de que dispuso dicho proveedor y que tiene almacenada.

Esta posición no se sabe cuándo fue tomada, por tanto su fiabilidad es baja. Para corregir estos defectos es necesario implementar mecanismos de forma manual en el código de la aplicación, y así se hizo en el caso de este proyecto.

La forma adecuada de trabajar es la llamada a requestLocationUpdates(), que conceptualmente significa la petición de aviso cuando un proveedor obtenga una nueva posición.

Se requieren dos implementaciones concretas y la aplicación sería capaz de disponer de la posición actualizada proporcionada por los sistema de GPS o de Redes:

1. Llama a requestLocationUpdates() con parámetros de entrada:
   - Nombre del proveedor de localización
   - Tiempo entre updates: 0. En cuanto llegue una, que sea comunicada.
   - Distancia: 0. Que no sea un requisito.
   - Instancia de un escuchador de eventos de localización LocationListener
2. Implementación de dicho escuchador: En el método onLocationChanged() es donde se programará la lógica de la aplicación que he de interactuar y manipular las posiciones geográficas, ya que este método será el que se invoque cuando llegue una nueva localización.

Conexiones HTTP

La herramienta básica de comunicación de una aplicación Android con Internet son los sockets.

En concreto, en este proyecto se ha utilizado un nivel de abstracción ligeramente superior, ya que se han empleado métodos definidos en Android para el intercambio de mensajes tipo petición-respuesta del protocolo HTTP y HTTPS

En primer lugar, es conveniente analizar la conexión a establecer (TCP) y las secuencias de la comunicación, que siguen la arquitectura Cliente-Servidor:

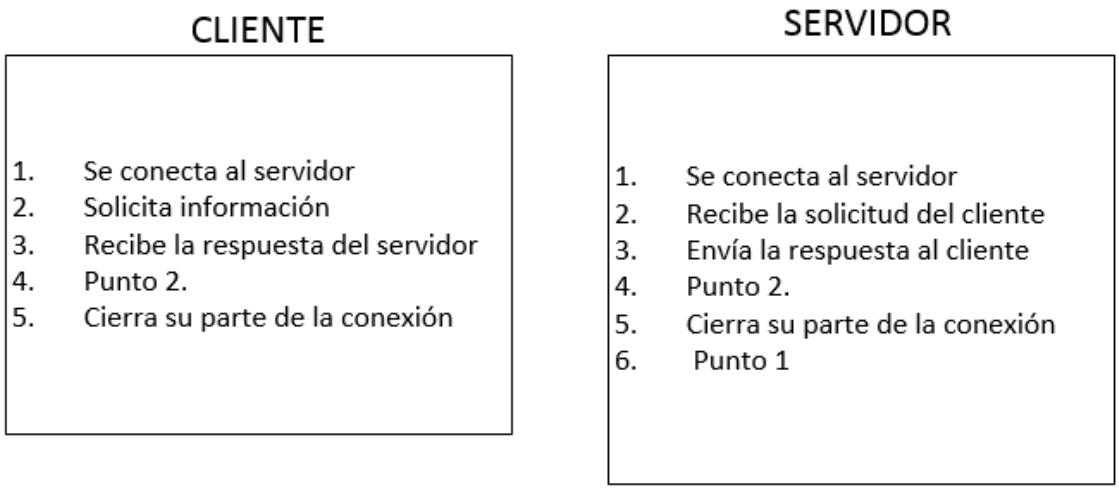

Figura 3.31: Secuencia de comunicación Cliente-Servidor HTTP

Con este mecanismo y el uso de las estructuras estándar de mensajes HTTP se obtiene la forma más utilizada de acceder a recursos de servidores web a lo largo del mundo.

Hoy en día, el uso de la World Wide Web no es meramente de consulta de información, como lo fue en un principio, sino que se ha convertido en una potente herramienta tecnológica que es usada para el intercambio de todo tipo de ficheros o comunicación entre diferentes servicios y aplicaciones de

diversa índole.

A continuación se listan algunas consideraciones previas al desarrollo de elementos de HTTP en la aplicación de Android:

- El paquete android.net proporciona acceso a propiedades de conectividad del dispositivo mediante de la clase ConnectivityManager: permite saber si el dispositivo está conectado a alguna red.
- El paquete java.net, implementado en parte en Android, proporciona soporte para programar aplicaciones en red, y proporciona algún soporte para el protocolo HTTP.
- La librería HttpClient de Apache, implementada en Android, ha sido diseñada para implementar clientes HTTP en Android.

En nuestro proyecto hemos utilizado las herramientas para conexión e intercambio HTTP en Android proporcionadas por las librerías de Apache: org.apache.commons.httpclient.*.

La principal función de la biblioteca Apache HttpClient es ejecutar métodos HTTP (GET, POST, PUT, HEAD,...).

La ejecución de un método HTTP implica el intercambio de peticiones HTTP y respuestas HTTP, que normalmente son realizadas internamente por HttpClient de manera transparente al programador.

El programador proporciona un objeto petición que ha de ser ejecutado y HttpClient se encarga de transmitir la petición y de recibir la respuesta a través de una conexión TCP, elevando una excepción si hay algún problema A continuación se muestra una secuencia de comunicación HTTP en Android mediante las clases brindadas por la librería Apache.

Petición GET: Esta petición consiste en la consulta HTTP de una página web a la que se accede mediante una URL. Pueden introducirse parámetros a la petición incluyendo, al final de dicha URL el carácter ? seguido del nombre del parámetro, el símbolo $\bar{y}$ su valor. Se añaden más valores concatenando el carácter & y la misma estructura anterior.

El primer paso es crear un cliente HTTP, a continuación indicar la URL y preparar la petición.

```
HttpClient httpclient = new DefaultHttpClient();
HttpGet httpget = new HttpGet("http://192.168.1.33:3000/subastas.xml");
```

Figura 3.32: Primeros pasos petición HTTP

Inmediatamente después, ejecutar la petición y recibir la respuesta.

```
HttpResponse response = httpclient.execute(httpget);
HttpEntity entity = response.getEntity();
```

Figura 3.33: Petición GET HTTP

Por último, se procesa el contenido de la respuesta del servidor y se codifica la lógica de la aplicación según la naturaleza de la misma.

```
if (entity != null) {
  XmlPullParserFactory parserCreator = XmlPullParserFactory.newInstance();
  XmlPullParser parser = parserCreator.newPullParser();
  parser.setInput(entity.getContent(), null);
  int parserEvent = parser.getEventType();
  while (parserEvent != XmlPullParser.END_DOCUMENT)
  {
    switch (parserEvent)
    {
    case XmlPullParser.START_TAG:
    //...
    }
    parserEvent = parser.next();
  }
}
```

Figura 3.34: Respuesta a la Petición GET HTTP

Es importante incluir todas las operaciones relacionadas con HTTP en un bloque try-catch, de la misma forma que se realiza en Java, ya que suelen ser fuente de errores que hay que tratar en la aplicación.

Petición POST: Esta petición consiste en el envío de un formulario con datos en el cuerpo del mensaje HTTP, en HTML que se enviará al servidor para que procese dichos datos.

En este caso es interesante añadir diferentes tipos de cabeceras HTTP, así como una lista de valores en el cuerpo del mismo.

En esencia, en Android posee la misma estructura que la petición GET, sin embargo adquiere un formato ligeramente distinto, que se muestra a continuación:

Preparación de los datos, de la URL y del cliente que ejecutará la petición POST:

```
List<NameValuePair> formparams = new ArrayList<NameValuePair>();
formparams.add(new BasicNameValuePair("nombre_campo1", "valor campo 1"));
formparams.add(new BasicNameValuePair("nombre_campo2", "valor campo 2"));

UrlEncodedFormEntity entity = new UrlEncodedFormEntity(formparams, "UTF-8");
HttpPost httppost = new HttpPost("http://192.168.1.33:3000/subastas.xml");
httppost.setEntity(entity);
```

Figura 3.35: Primeros pasos petición POST HTTP

Ejecución de la petición POST y obtención del cuerpo del mensaje de respuesta:

```
HttpResponse response = httpclient.execute(httppost);
HttpEntity responseEntity = response.getEntity();
```

Figura 3.36: Petición POST HTTP

Tratamiento de la respuesta:

```
if (responseEntity != null) {
  Log.i("Subastas", "getStatusCode=" + response.getStatusLine().getStatusCode());
  Log.i("Subastas", "responseEntity="+EntityUtils.toString(responseEntity));

}
```

Figura 3.37: Respuesta a la Petición POST HTTP

Como en el caso anterior, también deben encerrarse todas estas operaciones en una estructura de control de excepciones.

En el siguiente apartado de esta memoria se detallará la implementación de estos mecanismos en el caso particular de la aplicación de identificación de usuario para GeoRBAC.

GoogleCloudMessaging

GoogleCloudMessaging(GCM) para Android es un servicio que permite enviar dato s desde un servidor web a una aplicación móvil Android y viceversa. Este servicio maneja todos los aspectos relacionados con las colas, prioridades, almacenamiento y distribución de los mensajes al dispositivo Android objetivo. Es un novedoso servicio gratuito de Google.

Figura 3.38: Google Cloud Messaging (GCM)

Puede tratarse de un mensaje corto que indique a la aplicación de Android que hay datos nuevos que se pueden recuperar del servidor (por ejemplo, una película subida por un amigo) o puede ser un mensaje que contenga hasta 4 KB de datos de carga (de esta forma, aplicaciones como los mensajes instantáneos pueden consumir el mensaje directamente).

Una implementación complete de un Sistema GCM require tanto de una codificación en la aplicación destino como de una en el servidor central en-

cargado de enviar los mensajes a la aplicación.

GCM ofrece soporte bidireccional, como ya se ha comentado. Es decir, puede enviarse información desde el servidor central hasta la aplicación y también en sentido contrario.

Sin embargo, en el proyecto actual solo se ha implementado una vía, la que el origen de los datos es el servidor web implementado en el CEP y el destino de los mismos es la plataforma móvil instalada en el dispositivo del usuario del sistema de control de acceso GeoRBAC.

Solo ha sido necesaria esta vía ya que el mecanismo implementado de comunicación APP-Servidor es mediante HTTP, como ya se ha comentado en la sección anterior, y como se verá evidenciado en el siguiente apartado de este documento.

La secuencia que se sigue en una comunicación basada en GCM es la siguiente:

1. La aplicación Android envía al servidor los datos: sender id y application id. Con este envío y con estos datos hace una petición de registro en el servidor GCM.
2. Una vez registrado el servidor GCM le comunica a la aplicación el valor registration id.
3. Cuando la aplicación dispone del registration id, se lo envía al servidor de interés en nuestro proyecto, que es el que está situado en la parte central del esquema global de GeoRBAC, con la intención de que éste pueda enviar mensajes a GCM con la id del dispositivo destino.
4. Nuestro servidor registra esa información para su uso posterior.

Envío de notificaciones:

a)Como se ha comentado, cuando nuestra lógica de sistema necesita hacer una comunicación a la aplicación, envía al servidor de GCM el mensaje con el registration id almacenado del dispositivo destino.

b)GCM propaga este mensaje al dispositivo. Cabe mencionar que este mensaje se guarda hasta que el dispositivo pueda recibirlo, de tal manera

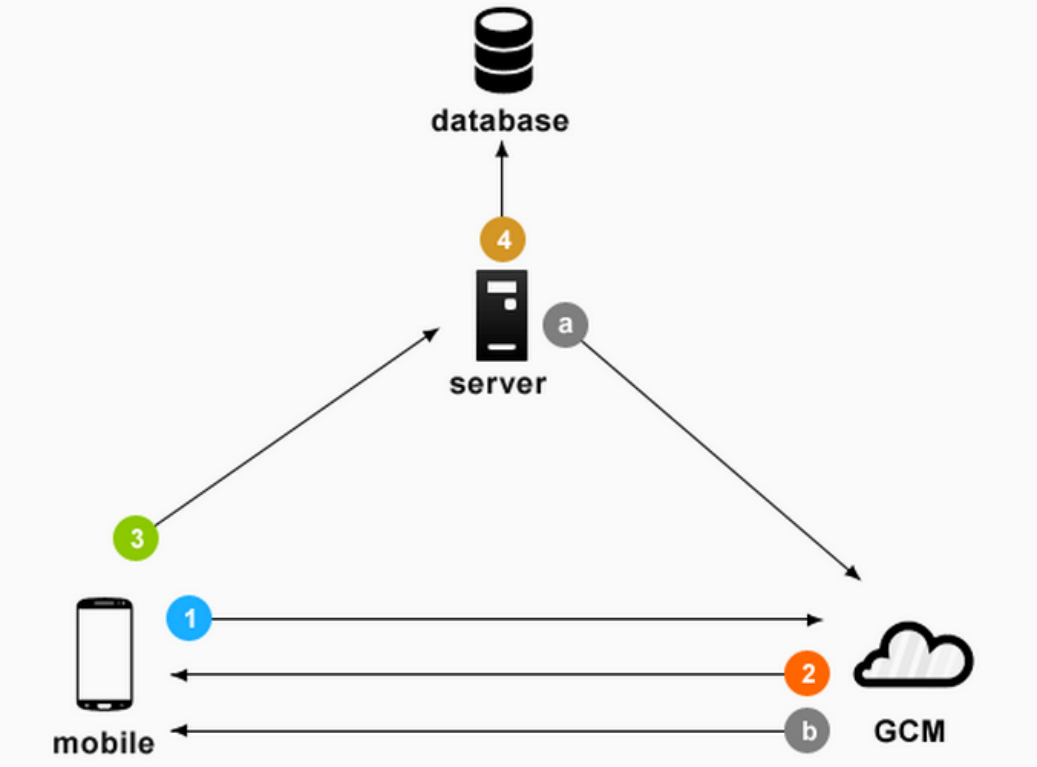

Figura 3.39: Esquema de funcionamiento de GCM [8]

que si se encuentra apagado o no disponible, GCM se asegurará de que el mensaje llegue a su destino, lo cual es un punto favorable de esta tecnología.

3.3.2. Procesos y capturas de pantalla

Registro de la aplicación

En la imagen 3.40 se muestra el proceso que se ha codificado para el registro del dispositivo móvil donde se encuentra instalada la aplicación como sensor. Tal y como se ha visto en el apartado de componentes globales de la solución, cada dispositivo móvil perteneciente a la organización que controle el sistema de GeoRBAC será reconocido por la lógica del mecanismo del CEP como un sensor registrado en el servidor SOS implementado.

Para ello, en primer lugar la aplicación deberá registrarse como tal, mediante una serie de peticiones HTTP frente al servidor central. Después ten-

drá que introducir datos de geolocalización mediante un formato XML determinado para que queden almacenadas las posiciones del dispositivo de una forma óptima en el SOS, siguiendo los estándares recomendados por la OGC.

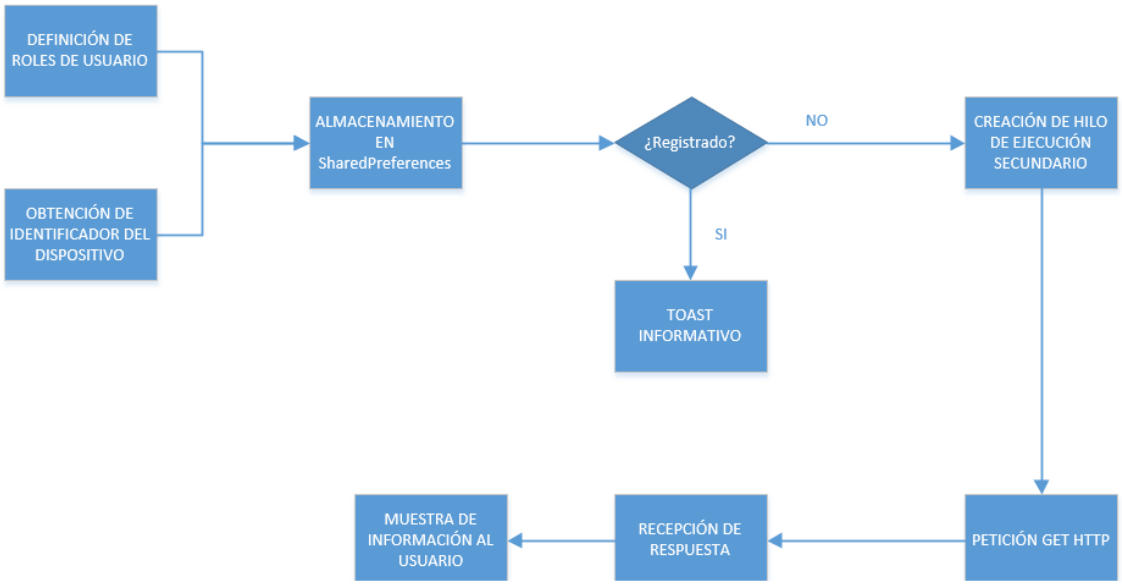

Figura 3.40: Proceso de registro del dispositivo móvil como sensor en el SOS [8]

A continuación se explican los pasos seguidos, y las tecnologías empleadas para cada uno de ellos:

1. **Definición de roles de usuario**
   Desde la aplicación se definen los roles de usuario de los que dispone ese dispositivo, estas directrices vendrán marcadas desde el servidor central de la arquitectura de GeoRBAC, y serán unos u otros en función de las políticas de la organización que desee implementar este mecanismo de seguridad.

2. **Obtención de identificador del dispositivo** Para este aspecto, se tomó la decisión de seleccionar el número SIM del dispositivo como identificador del mismo. Esto ofrece diversas ventajas, ya que ningún usuario podrá disponer del mismo SIM que otro en un instante.

   Android ofrece una serie de elementos que permiten consultar parámetros de configuración y propios del dispositivo móvil, como son el

número de teléfono, el IMEI, la SIM, los contactos, la conexión a internet, etc.

Para poder utilizar dichos métodos, y en concreto los necesarios para esta aplicación, hay que declarar los permisos pertinentes en el archivo AndroidManifest, el cual ya hemos estudiado.

En este caso, los permisos necesarios para esta parte de la implementación han sido:

- Estado del teléfono
- Internet

Puede observarse su codificación particular en el Anexo 1 de este mismo documento.

Tras la declaración de permisos, las operaciones que han de realizarse para acceder a dichos datos se muestran en la figura 3.41:

```
TelephonyManager telemamanger = (TelephonyManager)getSystemService(Context.TELEPHONY_SERVI
String sim = telemamanger.getDeviceId();
```

Figura 3.41: Obtención de la SIM del dispositivo

3. **Almacenamiento en SharedPreferences** Una vez se tienen los valores de los roles y la identificación de usuario el siguiente paso es el almacenamiento en el fichero SharedPreferences.

   Ya se ha hablado del mismo en la sección de Componentes de Android utilizados, y en este apartado se concreta su implementación para la aplicación a la que concierne este proyecto.

   En primer lugar, se declara como objeto común de toda la actividad.(Imagen 3.42).

   A continuación se usa un editor (imagen 3.43) para poder utilizar el mismo: tanto lectura como escritura.

```
SharedPreferences sharedPref;
```

Figura 3.42: Declaración de SharedPreferences

```
//Abrimos el archivo SharedPreferences donde guardamos clave:valor
sharedPref = PreferenceManager.getDefaultSharedPreferences(c);

SharedPreferences.Editor editor = sharedPref.edit();
```

Figura 3.43: Editor de SharedPreferences

Almacenado de los datos en SharedPreferences: (imagen 3.44) En la imagen se almacena un rol, una de las mejoras identificadas para la aplicación sería la ampliación del número de roles asignables.

```
editor.putString("sim", sim);
editor.putString("rol",rol);
```

Figura 3.44: Almacenamiento en SharedPreferences

4. **Comprobación de registro**
   En este punto es donde la aplicación comienza al pulsar el usuario el botón de R̈egistrar:

   Para este paso, lo primero que hace la aplicación es comprobar si ya se ha registrado, por si esta no fuera la primera vez que el usuario entra en la aplicación, o si pulsa el botón accidentalmente.

```
boolean regread = sharedPref.getBoolean("registered",false);
if(!regread){  //Si entra es que no está registrado, por tanto coge su SIM, su ROL y se registra
    //obtener número de teléfono para identificador
```

Figura 3.45: Comprobación de registro

Esta operación consiste en la consulta de un valor almacenado en SharedPreferences. Si este valor es verdadero significa que el usuario ya ha

registrado el dispositivo móvil como sensor y no es necesaria ninguna operación más al respecto. En este caso el sistema muestra un Toast informativo al usario confirmándole su registro. Se observa en la imagen 3.46.

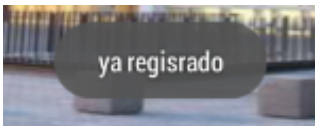

Figura 3.46: Toast Informativo

5. **Creación de un hilo de ejecución secundario**

   Como se observa en el diagrama de flujo, si el valor es falso entonces hay que proceder a registrarse. Para cumplir con las prácticas que se explicaron en la sección de Diseño de la solución, se crea un hilo de ejecución llamado RegisterTask, que será el encargado de realizar la petición HTTP al servidor central.

```
//Ejecutamos la tarea de registro
RegisterTask rtask = new RegisterTask(this);
rtask.execute(sim,rol);
```

Figura 3.47: Creación de hilo

6. **Petición GET HTTP**

   Una vez dentro del hilo ejecutándose en segundo plano, se realiza la petición GET:

```
DefaultHttpClient client = new DefaultHttpClient();
HttpGet get = new HttpGet("http://eogorriti.com:8088/sos/register?id="+params[0]+"&role="+params[1]);
HttpResponse resp = client.execute(get);
```

Figura 3.48: Creación de hilo

En la figura 3.48 se observa la realización de la petición utilizando componentes de la librería de Apache para cliente de HTTP, que ya se ha explicado con anterioridad en esta memoria.

La petición se hace a la URL donde está montado el servidor web que se ha implementado en el proyecto global de GeoRBAC. En esta petición se le informa al servidor, para que registre en el SOS como sensor a este dispositivo con los parámetros incluidos en la URL:

- Rol/Roles
- ID

Tras la contestación por parte del servidor, y debido a la naturaleza del funcionamiento y ciclo de vida de AsyncTask comentado anteriormente, cuando se recibe la respuesta, es procesa y el método onPostExecute() del hilo es llamado.

En la imagen 3.49 se muestra dicha operación:

```
            BufferedReader reader = new BufferedReader(new InputStreamReader(resp.getEntity().getContent()));

            String linea = reader.readLine();
            while(linea!=null){
                result+=linea;
                linea=reader.readLine();
            }
            reader.close();

        } catch (Exception e) {
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw);
            e.printStackTrace(pw);
            result+=sw.toString();


        return result;
    }


    @Override
    protected void onPostExecute(String result) {
        super.onPostExecute(result);

        Toast.makeText(act,result, Toast.LENGTH_LONG).show();
```

Figura 3.49: Proceso de RegisterTask

7. **Muestra de información al usuario**

Justamente en este método onPostExecute() comentado es donde mostramos la información al usuario, mediante un Toast exactamente igual al explicado para el caso en que el valor de SharedPreferences de registro es verdadero.

La imagen 3.50 ilustra la operación que se codifica, y en la figura 3.51 lo que verá el usuario en caso de un registro satisfactorio.

```
protected void onPostExecute(String result) {
    super.onPostExecute(result);

    Toast.makeText(act,result, Toast.LENGTH_LONG).show();
```

Figura 3.50: onPostExecute()

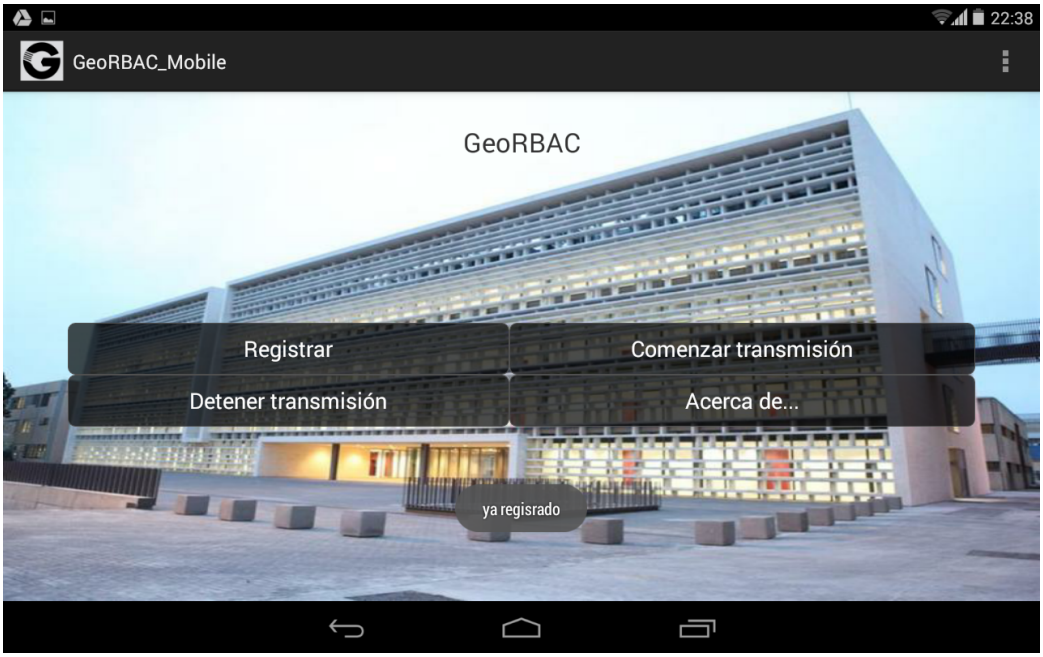

Figura 3.51: Interfaz de usuario

Captación y distribución de posición

Una de las partes más importantes de este proyecto ha sido conseguir una plataforma móvil fiable de identificación de usuario y ditribución a la lógica central del sistema GeoRBAC de la posición de dicho usuario en cualquier instante para poder ejercer control sobre el acceso basado en roles y en dicha situación espacial.

Y uno de los aspectos cruciales de la consecución de esa plataforma era la implementación de la capacidad del dispositivo móvil de ejecutar una serie de operaciones que obtuvieran la posición de una forma fiable, la distribuyeran al servidor web encargado de almacenarlas en el SOS y permitieran un funcionamiento normal de hilo principal y del sistema operativo.

Para conseguir dichos objetivos, se ha implementado esta funcionalidad siguiendo el diagrama de flujo que se representa en la imagen 3.52.

A continuación, tal como se ha hecho para el caso del registro de la aplicación como sensor en el SOS, se procede a explicar los pasos del diagrama de flujo de la captación y distribución de geolocalización.

1. **LocationManager** En primer lugar, tal como se ha comentado en el apartado de geolocalización en Android en Diseño de la solución, se crea una instancia del manejador LocationManager, que es el objeto mediante el cual podremos acceder a los servicios de localización de Android.

2. **Criteria** Siguiendo con las indicaciones teóricas explicadas anteriormente, creamos un objeto Criteria que nos permita definir criterios de criba a la hora de elegir proveedor de localización.

   En la figura 3.53 se observa la implementación del paso anterior y de éste.

3. **Definición del SOS**

   Se trata de la implementación de un objeto importado de una librería creada para el proyecto global de GeoRBAC.

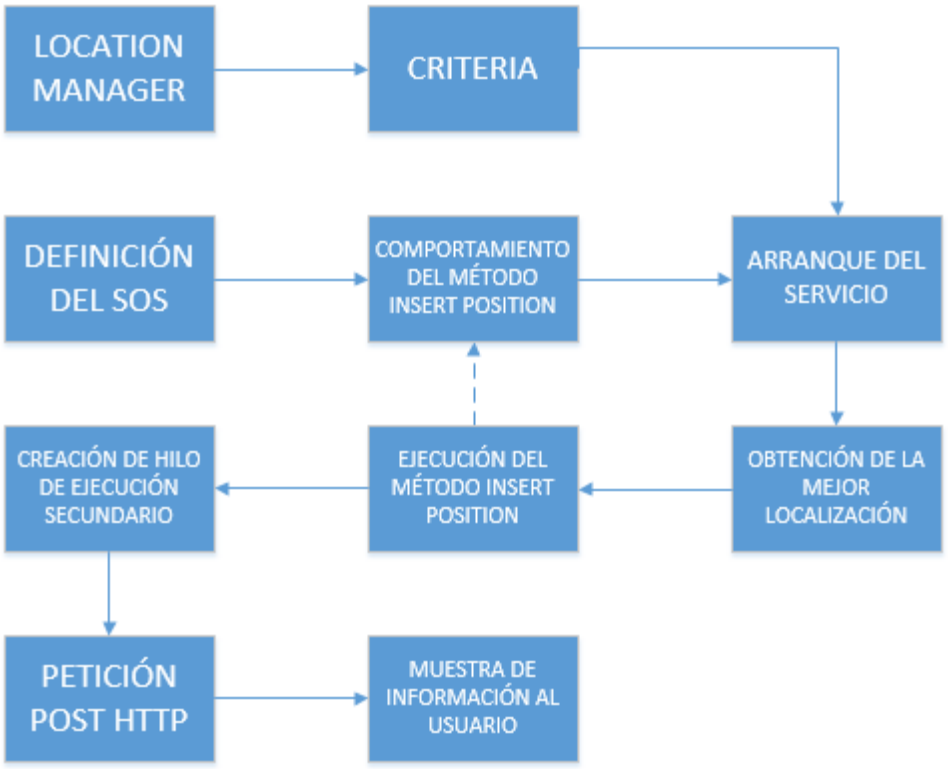

Figura 3.52: Diagrama de flujo de captación y distribución de posición geográfica

```
manejador = (LocationManager) getSystemService(LOCATION_SERVICE);
Criteria criteria = new Criteria();
```

Figura 3.53: LocationManager y Criteria

El objeto SOS se inicializa con la URL donde está situado el servidor SOS al que hay que enviar las posiciones espaciales. Al iniciar este objeto, hay que sobreescribir un método en uno de sus parámetros: el método performRequest(), que será llamado cuando se quiera introducir una nueva entrada de posición en la base de datos espaciales optimizada del SOS. Esta operación aparece reflejada en la imagen 3.54.

4. **Comportamiento del método InsertPosition**

```
sos = new Sos("http://eogorriti.com:8080/52n-sos-webapp/service",getApplicationContext()){
```

Figura 3.54: Instanciación del SOS en la aplicación móvil

A continuación ha de definirse el comportamiento del método Insert-Position. Se trata de un método implementado en la librería que se desarrolló para el proyecto global de GeoRBAC.

En el caso de Android hay que sobreescribirlo e indicarle las operaciones a realizar.

Las operaciones que se han implementado han sido:

- Generar el XML a enviar al servidor SOS para que éste reciba la información a partir del estándar definido de OGC. Se llama a la función getRequestEntity(), que ha sido codificada para generar dicho XML. La implementación de dicha función se halla en la figura 3.55, y consiste en la extracción de una plantilla situada en la carpeta assets del dominio de la aplicación, y la sustitución de los valores predefinidos por los que realmente se quiere enviar al SOS en ese momento.

- Creación de un nuevo hilo de ejecución para las conexiones HTTP que se encargarán de enviar el XML generado en el ítem anterior. En la figura 3.56 se muestra la implementación del comportamiento del método performRequest():

5. **Arranque del servicio**

El arranque del servicio de comunicación de posiciones al servidor SOS situado en el elemento central de la arquitectura de GeoRBAC se produce en el momento en el que el usuario pulsa el botón Comenzar transmisión.

En la anterior figura se observa el Toast mostrado al usuario para informar que el servicio ha sido arrancado (indicación del pid del servicio)

```
public String getRequestEntity() throws Exception{
    StringBuilder sb = new StringBuilder();
    Scanner scanner = new Scanner(sos.context.getAssets().open(template));
    while(true) try{
        String line = scanner.nextLine();
        for(String key : replaceTokens.keySet()) if(line.contains(key))
            line=line.replaceAll(key, replaceTokens.get(key));
        sb.append(line+"\n");
    }catch(Exception ex){
        break;
    }

    scanner.close();
    return sb.toString();
}
```

Figura 3.55: Método getRequestEntity()

```
@Override
public String performRequest(InsertPositionRequest arg0) throws Exception{
    arg0.getRequestEntity();

    PostTask pt = new PostTask(url,getApplicationContext());
    pt.execute(arg0);
    return "";
}
```

Figura 3.56: Implementación performRequest()

y se va a proceder a enviar posiciones:

Es entonces cuando el servicio arranca y comienza a ejecutar sus operaciones de forma periódica. El método del servicio que se ejecuta es onStartCommand(), y en este lugar es donde se ha codificado la funcionalidad principal de la geolocalización.

Para que la llamada a los servicios de geolocalización sea periódica se ha implementado de una manera especial, ya que en el entorno Android la clase Timer está muy limitada, su funcionamiento es complicado y su implementación nada recomendable. El método que se ha seguido ha sido:

- Creación de un objeto de la clase Handler

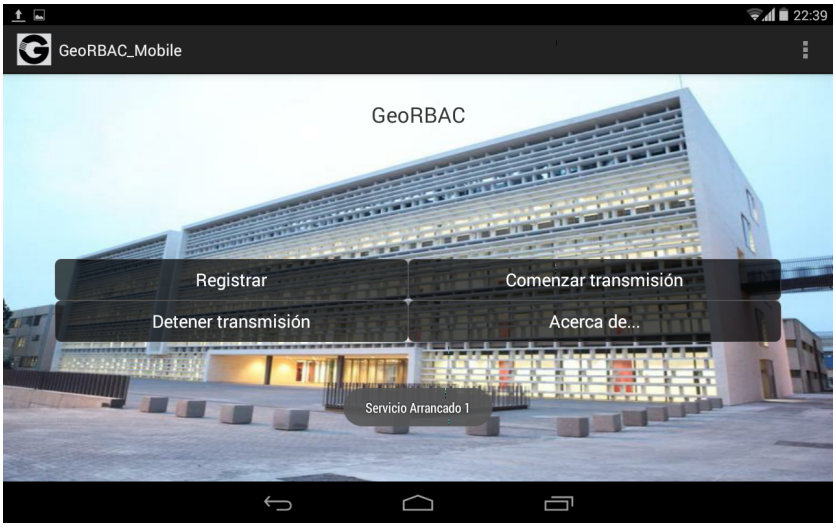

Figura 3.57: Arranque del servicio

- Creación de un Runnable, donde se codifica la lógica de la aplicación
- Indicación del periodo entre que el Handler llama al Runnable una vez y la siguiente. Este periodo será el guardado en una variable en SharedPreferences y que habrá sido configurado por el usuario.

Como comentario, se ha inicializado el servicio en modo START_STICKY , tal y como se ha comentado en apartados anteriores.

En las siguientes fases se observará las funciones que lleva a cabo el servicio, cuya implementación se muestra en la imagen 3.58.

6. **Obtención de la mejor localización**

La obtención de la mejor localización comienza en la llamada al método getBestLocation() del servicio. De forma esquemática, el funcionamiento de dicho método es el siguiente:

a) Analiza los proveedores GPS

b) Analiza los proveedores de redes de datos

```
@Override
public int onStartCommand(Intent intent, int flags, int idArranque){
    Toast.makeText(this,"Servicio Arrancado "+idArranque,Toast.LENGTH_SHORT).show();

    handler = new Handler();

    runnable = new Runnable(){
        @Override
        public void run(){
            sp = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
            localizacion = getBestLocation();
            lat = localizacion.getLatitude();
            loc = localizacion.getLongitude();

            Toast.makeText(SOSService.this, ""+lat+", "+loc, Toast.LENGTH_SHORT).show();
            try{
            sos.insertPosition(myId, new Position(lat,loc));
            }catch(Exception e){
                StringWriter sw = new StringWriter();
                e.printStackTrace(new PrintWriter(sw));
                Toast.makeText(getApplicationContext(), sw.toString(), Toast.LENGTH_LONG).show();
            }

            handler.postDelayed(runnable, sp.getLong("period", 10000L));//default period: 10 seconds
        }
    };

    handler.postDelayed(runnable, 500L);

    return START_STICKY;
}
```

Figura 3.58: Implementación del Servicio

c) Hace un cálculo para comprobar cuál de ellos ofrece una posición más actualizada

d) Se suscribe a dicho proveedor

e) Obtiene la posición que ofrece el mismo

Para poder ejecutarla es necesario haber configurado en el dispositivo el modo preciso de geolocalización, tal y como indica la imagen 3.59.

De esta forma, el resultado que nos aporta dicho método es una posición, mediante un objeto Position cuyos atributos son:

- Latitud
- Longitud

Estos valores son los que se enviarán encapsulados en la plantilla de XML (que se adjunta en el Anexo 2 de este documento) al servidor

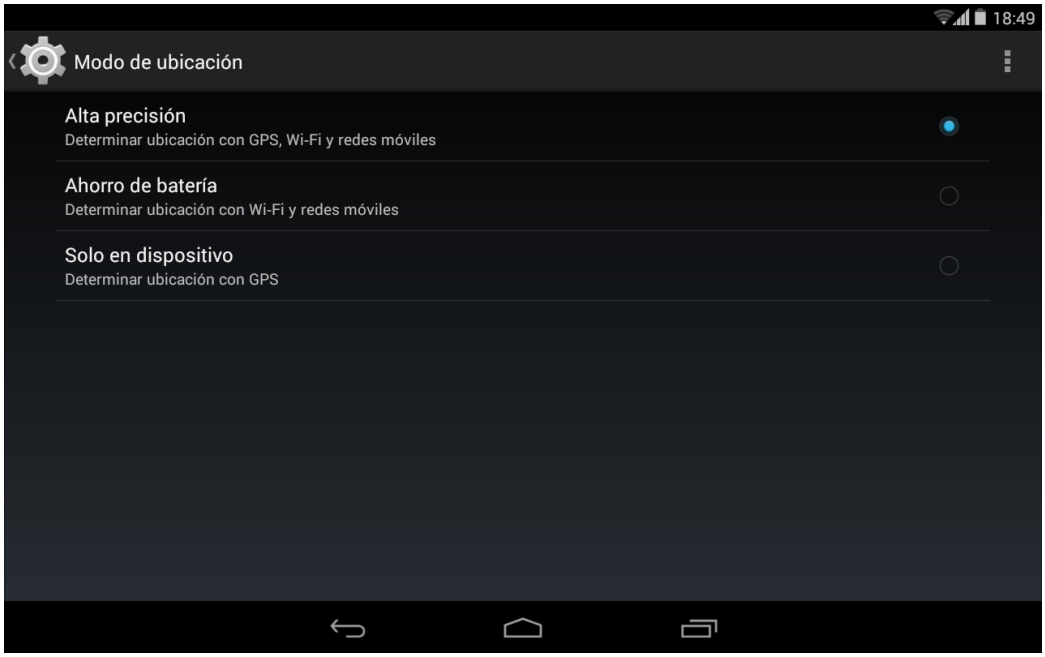

Figura 3.59: Configuración del modo preciso

SOS para su procesamiento, y también serán éstos los que se mostrarán al usuario para informarle de la posición mediante un Toast, tal y como se observa en la figura 3.60.

7. **Petición POST HTTP**

Como se observa en la imagen 3.58 las operaciones relacionadas con la petición POST HTTP se realizan en un hilo(thread) de tipo AsyncTask.

De forma similar a la petición GET, la secuencia implementada para realizarla ha sido la siguiente:

a) Creación de cliente HTTP

b) Creación de petición POST a partir de la URL

c) Inserción de cabeceras:

- Codificación UTF-8
- Servicio web SOAP + XML

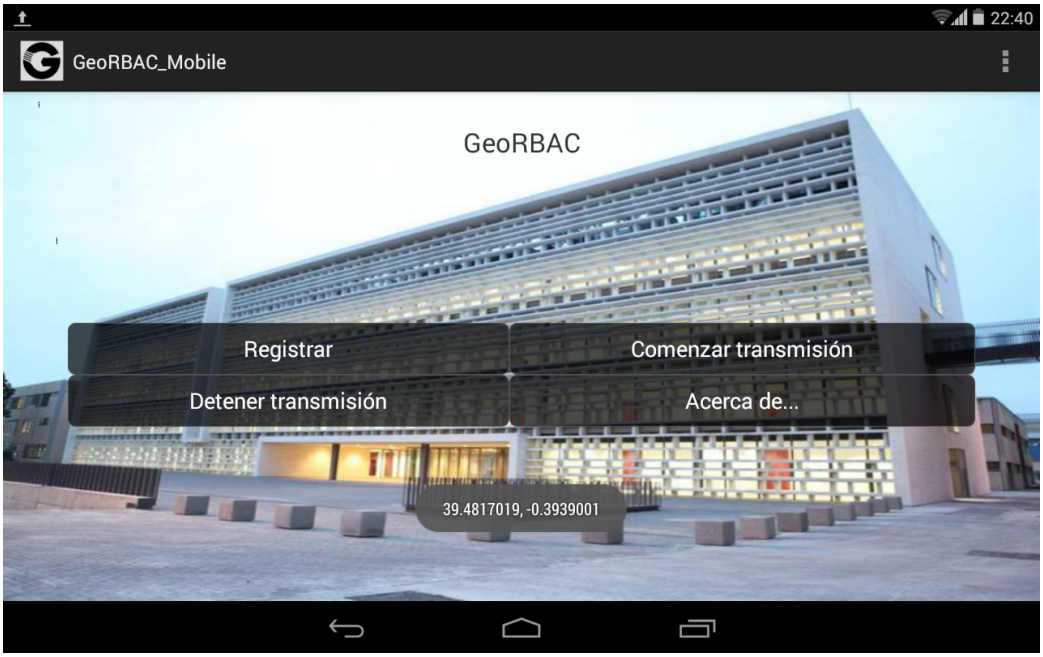

Figura 3.60: Distribución de posición

d) Ejecución de la petición

e) Recepción y procesado de la respuesta

Interacción con el usuario

En este apartado se muestran una serie de implementaciones relacionadas con la interfaz de usuario de la aplicación y la interacción entre ambos.

En la imagen 3.61 se muestra una vista general de la interfaz de usuario de la aplicación.

Debido a la gran variedad de dispositivos, APIs, versiones de Android y configuraciones disponibles, esta aplicación se ha desarrollado para que funcione correctamente adaptada a todas ellas. Además se ha incluido la opción de rotación de pantalla, sin que la experiencia del usuario se vea alterada, manteniendo un equilibrio y una forma en consonancia. Prueba de ello es la figura 3.62

Tal y como se ha comentado en el apartado de Diseño de la solución, en este proyecto se ha implementado una de las características más comunes de

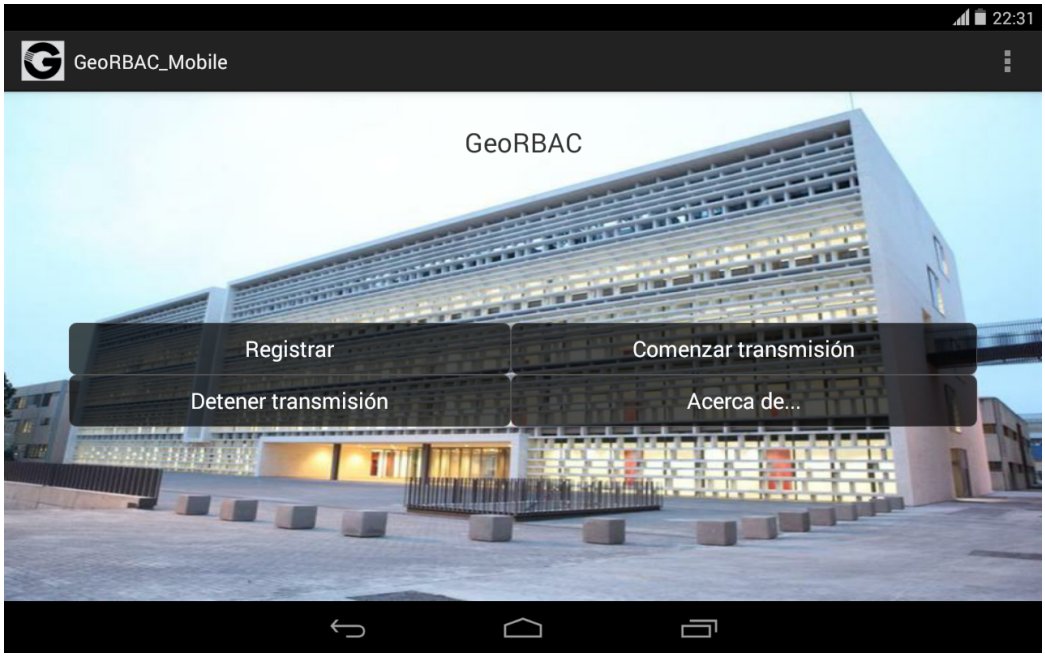

Figura 3.61: Interfaz Gráfica de Usuario

las aplicaciones de Android: el menú de opciones.

En nuestro caso, la opción que se ofrece al usuario es: Configuración. En esta opción el usuario introducirá el valor (en milisegundos) del tiempo de refresco de las posiciones geográficas, como se ha comentado en la sección anterior. Este valor será almacenado en el fichero SharedPreferences y más tarde accedido por las operaciones que lo requieran.

El resultado de estas implementaciones se muestra en las imágenes: 3.60, 3.64 y 3.65.

Por último, al pulsar el botón Acerca de.. el usuario observará un Alert-Dialog que informa sobre los autores del proyecto global de GeoRBAC.

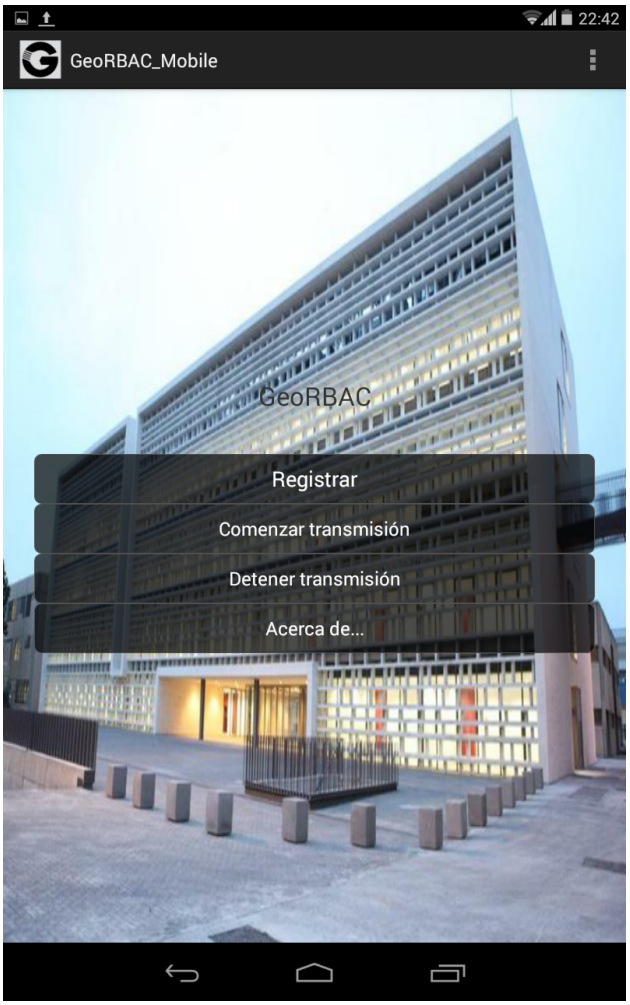

Figura 3.62: Interfaz Gráfica de Usuario pantalla vertical

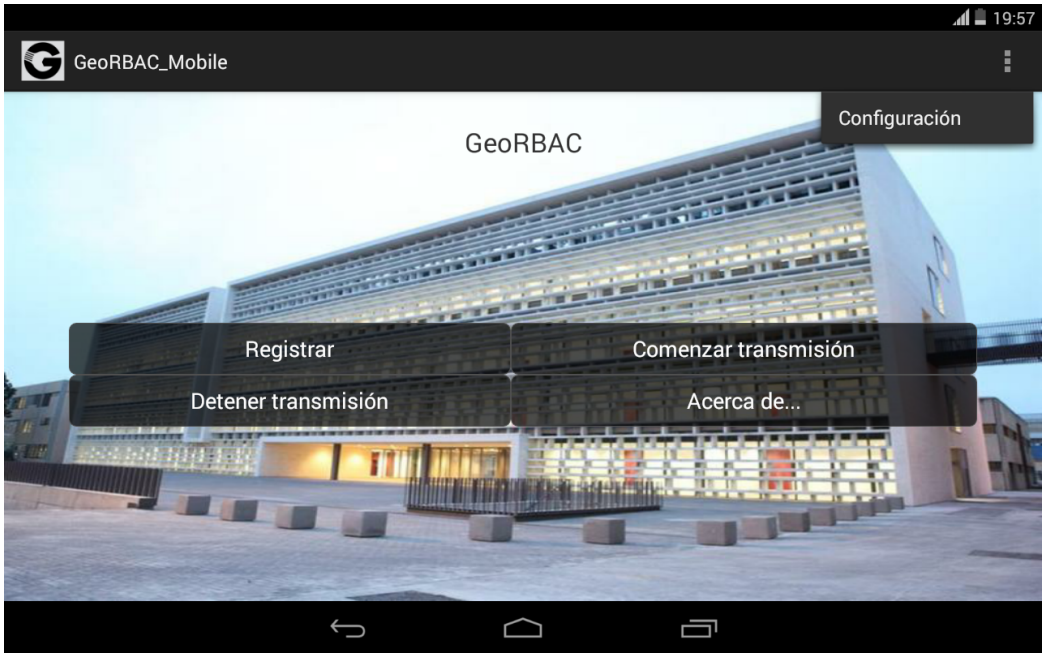

Figura 3.63: Menú de opciones

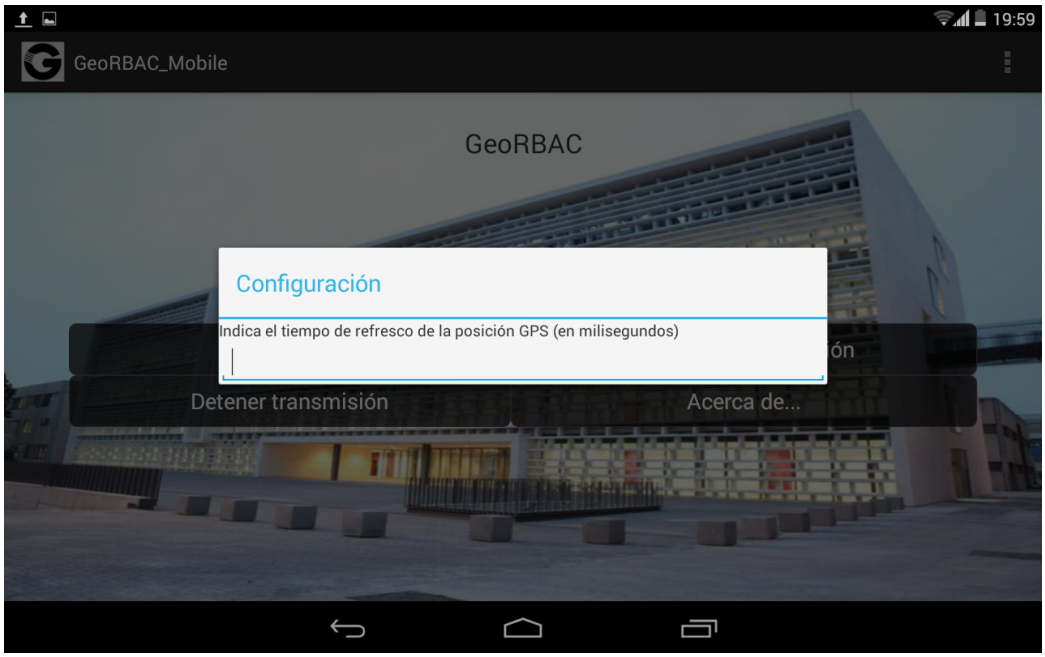

Figura 3.64: Configuración de tiempo de refresco

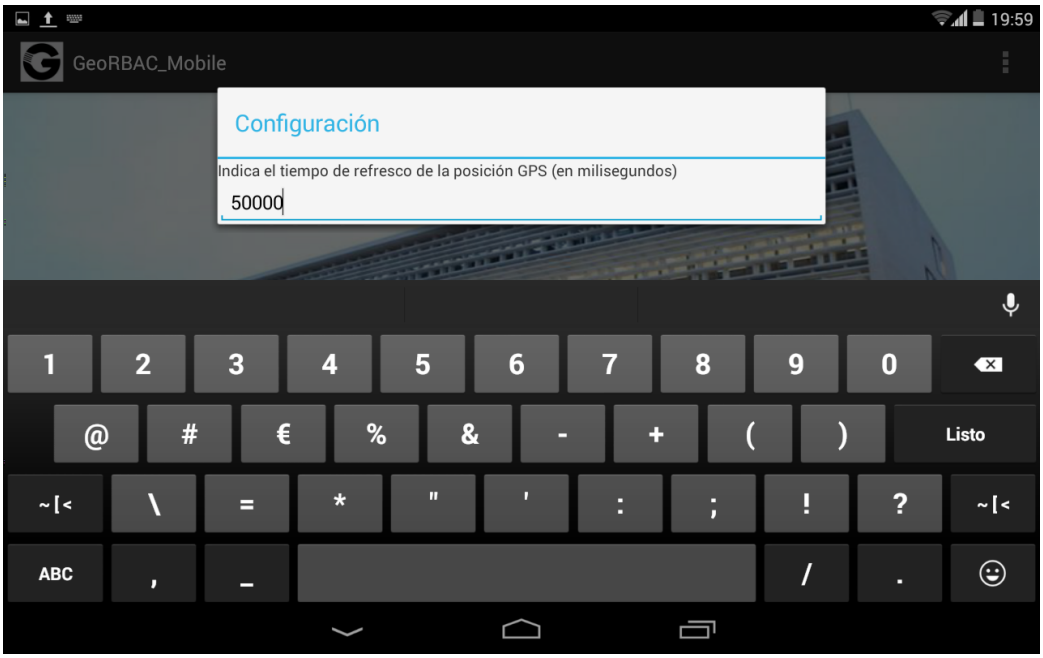

Figura 3.65: Configuración de tiempo de refresco

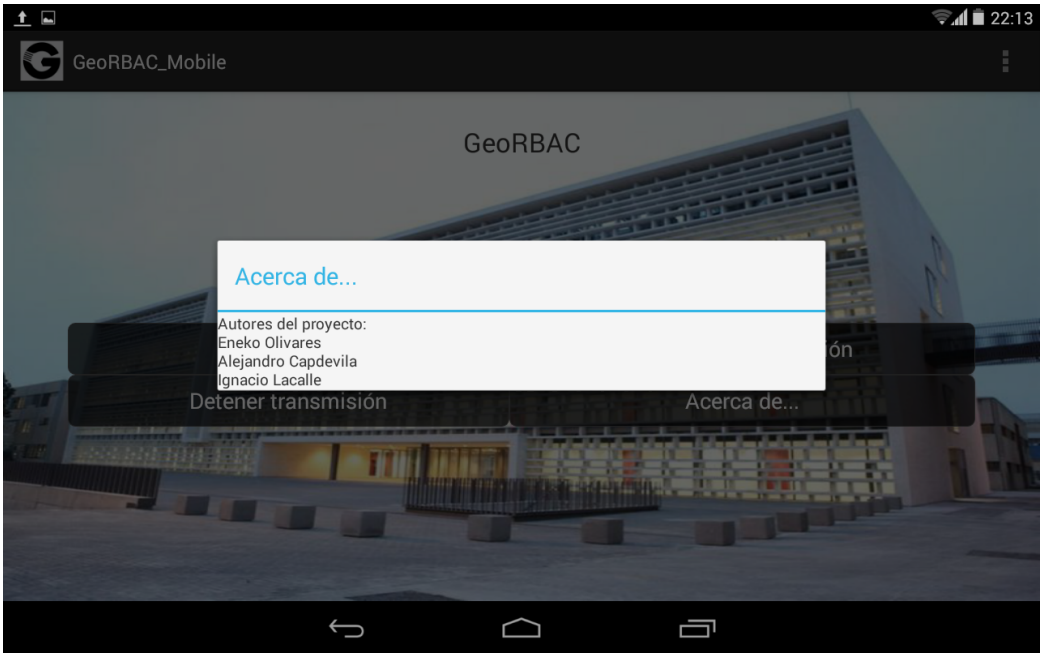

Figura 3.66: AlertDialog botón Acerca de...

Capítulo 4

Conclusión

En este proyecto se ha desarrollado una aplicación en Android que realiza la función de identificación de usuario y plataforma distribuidora de posiciones geográficas, enmarcado en un contexto global de un sistema de control de acceso basado en roles y geolocalización GeoRBAC completo.

En él se ha realizado un exhaustivo análisis sobre ciertas características del desarrollo de aplicaciones sobre el sistema operativo Android, focalizado principalmente en aquellos aspectos de interés para la geolocalización y la conexión web.

Este tipo de sistemas de control de acceso ofrecen un amplio campo de desarrollo futuro, ya que gracias al crecimiento de las ideas del Internet de las Cosas, de aquí unos años todos los dispositivos estarán interconectados. Incluso aquellos que hoy no se concibe que sean susceptibles de establecer comunicación con otros a través de las redes.

También el campo de las aplicaciones está sufriendo un uso masivo y exponencial en los últimos tiempos. Cada vez son más cosas las que pueden realizarse con ellas, y prueba de esto es la aplicación desarrollada en este proyecto, que es capaz de obtener posiciones geográficas de diferentes fuentes, logrando precisión en zonas de poca cobertura e incluso interiores, aspecto bastante complicado hasta ahora.

En la misma línea, a medida que vaya ampliándose la comunidad abierta de desarrolladores Android, y vayan surgiendo nuevas versiones del SDK, las funcionalidades que podrán realizarse sobre los aspectos de seguridad y geolocalización irán aumentando, permitiendo realizar aplicaciones cada vez más precisas y complejas.

En conclusión, al finalizar estos meses de trabajo se ha conseguido, por una parte un sistema global de control de acceso basado en roles y en posiciones geográficas. Este sistema, pese a ser muy modesto y contar con funcionalidades básicas, consigue escalabilidad horizontal y cumple con uno de los requisitos fundamentales que se plantearon al inicio y es que **ha sido implementado en un entorno real**.

Por otra parte, la aplicación desarrollada ha cumplido los objetivos planteados al inicio de la misma, funcionando correctamente y ofreciendo una plataforma móvil de identificación de usuario práctica y empleando las tecnologías más modernas disponibles en el entorno de desarrollo de aplicaciones en Android.

Resultados obtenidos

En este proyecto, pese a no haber realizado un análisis de prestaciones exhaustivos, sí pueden extraerse ciertas conclusiones de interés para futuros desarrollos en este ámbito:

- La localización combinada entre las tecnologías de GPS y red de datos han resultado ofrecer una precisión de entorno a 3 metros en el interior de los edificios en los que se ha probado: edificios nuevo y viejo de la ETSIT, siendo esta distancia menor incluso al salir a la zona abierta entre éstos.
- La ejecución de Android es muy sensible al nivel de optimización de código. Al ser una plataforma que dispone de recursos limitados a la hora de ejecutar las operaciones programadas, es crucial reducir el número de llamas al sistema, operaciones, etc. con la intención de lograr un mejor funcionamiento una mejor QoE para el usuario.
- Los dispositivos móviles como identificador de acceso consiguen responder a los requisitos planteados por el IoT y la infraestructura de seguridad basada en roles y geolocalización.

Mejoras identificadas y líneas futuras

Tras la conclusión comentada previamente, junto con el análisis renovado del estado del arte, se pueden argumentar las siguientes mejoras potenciales referentes a las implementaciones realizadas en este proyecto:

- Una de las limitaciones de la localización basada en GPS y red de datos es que es un posicionamiento plano, no tiene componente vertical. Esta

cuestión sería de relevancia en casos en que las instalaciones posean varias alturas y se deseen establecer criterios de acceso diferentes entre ellas.

- Podría desarrollarse una aplicación del estilo de la actual mejorando varios aspectos de ella:
  - Mayor capacidad de almacenamiento: Podría ser interesante que el dispositivo móvil almacenara más información sobre los accesos, características del sistema e información personal del usuario. Tal vez se pudiera conseguir empleando SQLite o sistemas más innovadores y ligeros que permitieran a la aplicación funcionar a la misma velocidad.
  - Mejora de la precisión en las medidas
  - Implementación de la aplicación en otras plataformas, para cubrir un espectro mayor en número y tipología de dispositivos, que funcionaran sobre plataformas como iOS, BlackBerry, Symbian, etc.
  - Mayor capacidad de comunicación bidireccional con el servidor central: Perfeccionar la interacción entre la lógica de la aplicación y el servidor central, haciendo más inteligente la inmersión del concepto de IoT en el sistema.
  - Introducción de más roles por usuario
  - Capacidad de asignación dinámica de roles: Aceptación e implementación del concepto que un usuario puede cambiar de roles e identificarse con el mismo dispositivo.
- Realizar análisis exhaustivo sobre datos concretos:
  - Tiempos de respuesta de la aplicación
  - Porcentaje de falsos positivos
  - Tiempos de obtención de localización
  - Gráficos sobre precisión de las posiciones
  - Análisis de la sobrecarga del sistema

En conclusión, se ha desarrollado una aplicación móvil integral, aunque con bastantes focos de mejora, enmarcada en una casuística sencilla de infraestructura de control de acceso incluyendo la unión de conceptos de roles y geolocalización, aplicando todos los componentes a la vida real mediante una implementación en que se monitorizaba el movimiento de alumnos

con sus dispositivos móviles por las instalaciones de la UPV, y obteniendo unos resultados satisfactorios, abriendo así una ventana a futuros desarrollos nutridos de una fuerte base teórica.

Capítulo 5

Anexo 1

En este Anexo se muestra el archivo AndroidManifest de la aplicación que se ha desarrollado en el proyecto actual.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.example.georbac_mobile"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />
      <uses-permission android:name="android.permission.READ_PHONE_STATE" />
      <uses-permission android:name="android.permission.INTERNET" />
      <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
      <uses-permission android:name="android.permission.WRITE_INTERNAL_STORAGE" />
      <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
      <uses-permission android:name="android.permission.READ_INTERNAL_STORAGE" />
      <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
      <uses-permission android:name="android.permission.ACCES_COARSE_LOCATION" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/logo"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".Main"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
          <service
          android:name=".SOSService" />
    </application>
</manifest>
```

Figura 5.1: Archivo AndroidManifest de la aplicación

Capítulo 6

Anexo 2

En este Anexo se muestra el archivo de plantilla para el envío de posiciones espaciales al SOS desde la aplicación que se ha desarrollado en el proyecto actual.

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/05/soap-envelope
     http://www.w3.org/2003/05/soap-envelope/soap-envelope.xsd">
    <env:Body>
        <sos:InsertObservation
            xmlns:sos="http://www.opengis.net/sos/2.0"
            xmlns:swes="http://www.opengis.net/swes/2.0"
            xmlns:swe="http://www.opengis.net/swe/2.0"
            xmlns:sml="http://www.opengis.net/sensorML/1.0.1"
            xmlns:gml="http://www.opengis.net/gml/3.2"
            xmlns:xlink="http://www.w3.org/1999/xlink"
            xmlns:om="http://www.opengis.net/om/2.0"
            xmlns:sams="http://www.opengis.net/samplingSpatial/2.0"
            xmlns:sf="http://www.opengis.net/sampling/2.0"
            xmlns:xs="http://www.w3.org/2001/XMLSchema" service="SOS"
             version="2.0.0" xsi:schemaLocation="http://www.opengis.net/sos/2.0
             http://schemas.opengis.net/sos/2.0/sos.xsd
              http://www.opengis.net/samplingSpatial/2.0 http://schemas.opengis.net/
             samplingSpatial/2.0/spatialSamplingFeature.xsd">
            <!-- multiple offerings are possible -->
            <sos:offering>offering_placeholder</sos:offering>
            <sos:observation>
                <om:OM_Observation gml:id="o1">
                    <om:type xlink:href="http://www.opengis.net/
                    def/observationType/OGC-OM/2.0/OM_GeometryObservation"/>
                    <om:phenomenonTime>
                        <gml:TimeInstant gml:id="phenomenonTime">
                            <gml:timePosition>time_placeholder</gml:timePosition>
                        </gml:TimeInstant>
                    </om:phenomenonTime>
```

Figura 6.1: Plantilla XML Parte 1

```
                </om:phenomenonTime>
                <om:resultTime xlink:href="#phenomenonTime"/>
                <om:procedure xlink:href="procedure_placeholder"/>
                <om:observedProperty xlink:href="http://www.52north.org/
                test/observableProperty/9_6"/>
                <om:featureOfInterest>
                    <sams:SF_SpatialSamplingFeature gml:id="ssf_test_feature_9">
                        <gml:identifier codeSpace="">featureOfInterest_placeholder
                        </gml:identifier>
                        <gml:name>52°North</gml:name>
                        <sf:type xlink:href="http://www.opengis.net/def/samplingFeatureType/
                        OGC-OM/2.0/SF_SamplingPoint"/>
                        <sf:sampledFeature xlink:href="featureOfInterest_placeholder"/>
                        <sams:shape>
                            <gml:Point gml:id="test_feature_9">
                                <gml:pos srsName="http://www.opengis.net/def/crs/
                                EPSG/0/4326">51.935101100104916 7.651968812254194</gml:pos>
                            </gml:Point>
                        </sams:shape>
                    </sams:SF_SpatialSamplingFeature>
                </om:featureOfInterest>
                <om:result xsi:type="gml:GeometryPropertyType">
                    <gml:Point gml:id="value">
                        <gml:pos srsName="http://www.opengis.net/def/crs/EPSG/0/4326">
                        position_placeholder</gml:pos>
                    </gml:Point>
                </om:result>
            </om:OM_Observation>
        </sos:observation>
    </sos:InsertObservation>
  </env:Body>
</env:Envelope>
```

Figura 6.2: Plantilla XML Parte 2

Capítulo 7

Anexo 3

Un aspecto de este Proyecto que merece la pena destacar es el entorno de desarrollo que se ha empleado, debido a las características especiales que posee el lenguaje nativo Android, basado en Java.

La herramienta que se ha empleado ha sido el IDE de Eclipse para Desarrolladores de Java, añadiendo los componentes necesarios para un satisfactorio desarrollo de aplicaciones Android.

Se va a hacer un repaso a las herramientas necesarias para poder efectuar las tareas comentadas:

Eclipse:
Eclipse es un IDE de desarrollo adaptado a muchos lenguajes, aunque basado en Java. Posee muchas ayudas visuales a la hora de codificar, lo que lo hace una herramienta interesante ya que esto facilita bastante el trabajo. Por otra parte, es gratuito y su desarrollo está formado por una comunidad muy activa, aportando nuevas actualizaciones constantemente y adapándose a las nuevas APIs y configuraciones en constante cambio.

En la imagen 7.1 se muestra una imagen del aspecto global de la herramienta:

SDK:

Otro elemento muy importante para desarrollar aplicaciones en Android, es su SDK (Sofware Development Kit). Es un archivo que no requiere instalación y que es la base del desarrollo en Android en Eclipse, ya que posee todas las librerías y recursos para poder programar en esta plataforma.

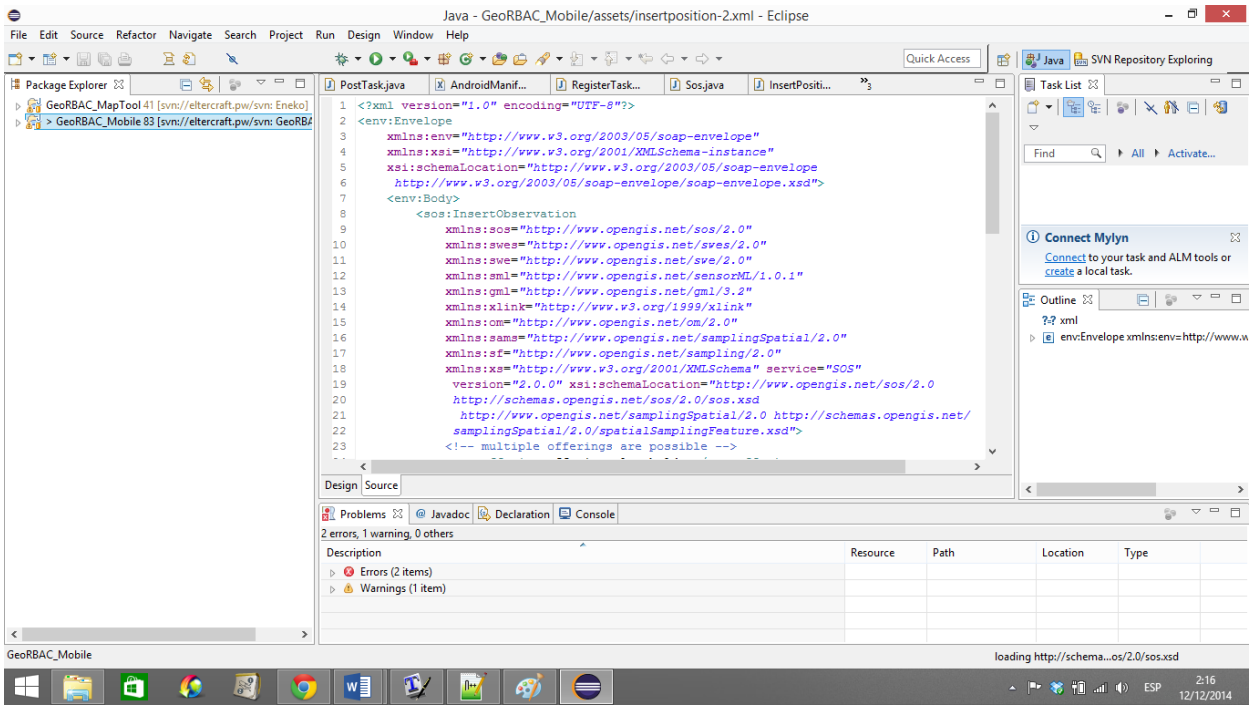

Figura 7.1: Entorno de desarrollo Eclipse

Plugin Eclipse:

Para integrar Android con Eclipse, se necesita un plugin que depende de la distribución del IDE Eclipse que se esté utilizando.

Una vez llegados a este punto, se trabaja con Android de manera normal, pero siempre teniendo en cuenta uno de los aspectos más importantes de la programación en Android: la estructura de carpetas Las carpetas en Android, tal y como está definido en su comportamiento por Google desde la existencia de las aplicaciones para esta plataforma, tienen que seguir el siguiente patrón:

Las más relevantes son comentadas a continuación:

- **src:** Carpeta que contiene el código fuente de la aplicación.
- **gen:** Carpeta que contiene el código generado de forma automática por el SDK.
- **R.java:** Define una clase que asocia los recursos de la aplicación con

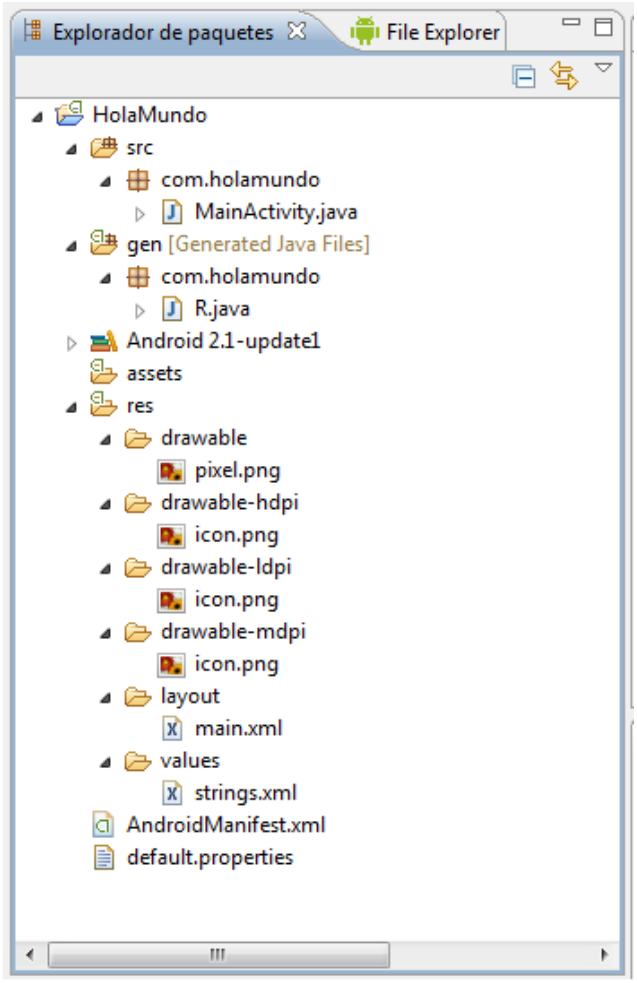

Figura 7.2: Estructura de carpetas en un proyecto Android

identificadores. De esta forma los recursos podrán ser accedidos desde Java.

- **assets:** Carpeta que puede contener una serie arbitraria de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, fuentes,...). A diferencia de la carpeta res, nunca se modifica el contenido de los ficheros de esta carpeta ni se les asociará un identificador. Esta carpeta se utiliza en el proyecto actual para almacenar el fichero XML plantilla para enviar una posición al SOS; como puede observarse en la subsección correspondiente en la sección de implementación.

- **bin:** En esta carpeta se compila el código y se genera el .apk, fichero

comprimido que contiene la aplicación final lista para instalar.

- **libs:** Código JAR con librerías que se quieran usar en el proyecto
- **res:** Carpeta que contiene los recursos usados por la aplicación.
- **drawable:** En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.
- **layout:** Contiene ficheros XML con las vistas de la aplicación.
- **menu:** Ficheros XML con los menús de cada actividad.
- **values:** También se usan ficheros XML para indicar valores del tipo string, color o estilo.
- **anim:** Contiene ficheros XML con animaciones Tween. Las animaciones son descritas al final del capítulo 4.
- **xml:** Otros ficheros XML requeridos por la aplicación.
- **raw:** Ficheros adicionales que no se encuentran en formato XML.
- **AndroidManifest.xml:** Explicado en el Anexo 1 de este docuento

Índice de figuras

2.1. Concepto de control de acceso 10
2.2. Fases del control de acceso 12
2.3. Esquema de control de acceso obligatorio 15
2.4. Estructura de un sistema de control de acceso 16
2.5. Esquema de control de acceso discrecional 17
2.6. Sistema de permisos del Unix File System 18
2.7. Modelo global de RBAC 20
2.8. Aproximación de RBAC 22
2.9. Esquema formal de RBAC 23
2.10. Ejemplo de creación de zonas espaciales para GeoRBAC . . . 25
2.11. Representación gráfica Primera Aproximación 30
2.12. Diferentes roles definidos 30
2.13. Acceso incorrecto 31

3.1. Disposición de barcos con sistema AIS 36
3.2. Escenario definitivo 37
3.3. Esquema global del sistema GeoRBAC 38
3.4. Arquitectura del Sistema Operativo Android 40
3.5. Gráfico comparativo de sistemas operativos móviles [5] 43
3.6. Cuota de mercado en dispositivos móviles [5] 44
3.7. Esquema conceptual MVC 45
3.8. Arquitectura Modelo-Vista-Controlador 46
3.9. Ciclo de vida de una Actividad en Android 49
3.10. Visión global de AndroidManifest 52
3.11. Ejemplo de AndroidManifest 53
3.12. Etiqueta manifest 53
3.13. Etiqueta uses-sdk 54
3.14. Etiqueta uses-permission 54
3.15. Etiqueta application 55
3.16. Etiqueta activity 56
3.17. Etiqueta intent-filter 57

3.18. Etiqueta caption 58
3.19. Acciones de una actividad de ejemplo 58
3.20. Ejemplo fichero XML de SharedPreferences 62
3.21. Ejemplo de mensaje Toast 63
3.22. Ciclo de vida de un Servicio de Android 65
3.23. Ejemplo de utilidad de hilos de ejecución [7] 67
3.24. Ciclo de vida de AsyncTask [7] 67
3.25. Ejemplo de Menú de Opciones 69
3.26. Ejemplos de AlertDialog 71
3.27. Consulta de proveedores de localización disponibles 72
3.28. Consulta de parámetros de los proveedores de la lista 73
3.29. Uso de la clase Criteria 74
3.30. Criba de proveedores 74
3.31. Secuencia de comunicación Cliente-Servidor HTTP 76
3.32. Primeros pasos petición HTTP 78
3.33. Petición GET HTTP 78
3.34. Respuesta a la Petición GET HTTP 78
3.35. Primeros pasos petición POST HTTP 79
3.36. Petición POST HTTP 79
3.37. Respuesta a la Petición POST HTTP 79
3.38. Google Cloud Messaging (GCM) 80
3.39. Esquema de funcionamiento de GCM [8] 82
3.40. Proceso de registro del dispositivo móvil como sensor en el SOS [8] 83
3.41. Obtención de la SIM del dispositivo 84
3.42. Declaración de SharedPreferences 85
3.43. Editor de SharedPreferences 85
3.44. Almacenamiento en SharedPreferences 85
3.45. Comprobación de registro 85
3.46. Toast Informativo 86
3.47. Creación de hilo 86
3.48. Creación de hilo 86
3.49. Proceso de RegisterTask 87
3.50. onPostExecute() 88
3.51. Interfaz de usuario 88
3.52. Diagrama de flujo de captación y distribución de posición geográfica 90
3.53. LocationManager y Criteria 90
3.54. Instanciación del SOS en la aplicación móvil 91
3.55. Método getRequestEntity() 92
3.56. Implementación performRequest() 92

3.57. Arranque del servicio 93
3.58. Implementación del Servicio 94
3.59. Configuración del modo preciso 95
3.60. Distribución de posición 96
3.61. Interfaz Gráfica de Usuario 97
3.62. Interfaz Gráfica de Usuario pantalla vertical 98
3.63. Menú de opciones 99
3.64. Configuración de tiempo de refresco 99
3.65. Configuración de tiempo de refresco 100
3.66. AlertDialog botón Acerca de... 100

5.1. Archivo AndroidManifest de la aplicación 105

6.1. Plantilla XML Parte 1 106
6.2. Plantilla XML Parte 2 107

7.1. Entorno de desarrollo Eclipse 109
7.2. Estructura de carpetas en un proyecto Android 110

Bibliografía

[1] D. Ferraiolo and R. Kuhn. *Role-based access controls. In In Proceedings of 15th NIST-NCSC National Computer Security Conference.* 1992.

[2] Barbara Catania Maria Luisa Damiani, Elisa Bertino and Paolo Perlasca. Geo-rbac: A spatially aware rbac. 2006.

[3] José Maria Ciampagna. Lbs – servicios basados en ubicación -. 2013.

[4] Jesús Tomás Gironés. *El gran libro de Android.* 2 edition, 2012.

[5] Statista: The Statistic Portal. Android's smartphone dominance is growing, http://www.statista.com/chart/1899/smartphone-market-share. 2013.

[6] Alejandro Alcalde. Fundamentos programación android: Conceptos básicos y componentes. 2014.

[7] Ramón Invarato Ricardo Moya. Asynctask en android, multitarea en android.

[8] Ravi Tamada. Android push notifications using google cloud messaging (gcm), php and mysql. 2013.

[9] Agustí Solanas y Jordi Castellà-Roca (FUOC. Fundación para la Universitat Oberta de Catalunya) Antoni Martínez-Ballesté. *Identificación, Autenticación y Control de Acceso.*

[10] Google Inc. *http://developer.android.com/reference.*

[11] Iván Posilio Gellida. El archivo androidmanifest.xml. 2013.

[12] Universitat Politècnica de València. *Vídeos en PoliTube sobre Android. http://politube.upv.es.*

[13] National Institute of Standards and Technology. Attributed bases access control(abac). 2013.

[14] Google Inc. *https://developer.android.com/google/gcm*.