Android 2.3.3 o superior

.NET 2008 Framework 3.5

Web Service ASMX SOAP

SQL Server 2005 Express

**Capitulo 3**

**Propuesta:**

Como se menciona en capítulos anteriores, el consumo energético, hoy en día, es uno de los problemas más importantes a enfrentar en los dispositivos móviles. Por esta razón nuestro trabajo tiene el objetivo de optimizar el rendimiento de la batería. Actualmente, con el auge de la información y la comunicación se encuentran disponibles una gran cantidad de datos y funciones que pueden ser muy útiles a la hora de estudiar el comportamiento de los diferentes usuarios, ya sea que estén utilizando, o no, su dispositivos. Información como la localización, o la red inalámbrica a la cual un aparato se encuentra conectado, permiten estudiar en que contextos se malgasta el potencial energético. Eventos relacionados con cambios de estados en el dispositivo, nos brindan información acerca de cuáles son los horarios pico en los cuales se utiliza un aparato y cuál es la posible relación entre estas variaciones de estado.

En este trabajo se propone LoguinUse, una aplicación destinada a recopilar información relacionada al estado y el contexto en el cual un dispositivo es utilizado, para luego, en base a estos datos, realizar una investigación con el propósito de encontrar patrones comportamentales en los cuales se detecte un mal uso de la energía. LoguinUse es una aplicación para dispositivos móviles con sistema operativo Android. La plataforma se eligió debido a que cuenta con una gran popularidad, por lo que cubre un gran número de usuarios. La aplicación recopilará diversa información sobre el uso que los usuarios le dan a sus dispositivos móviles, por ejemplo lugar del uso, conexiones WiFi, conexiones Bluetooth, etc. La información recopilada por la aplicación será comprimida y enviada a un servidor dedicado, desarrollado sobre la plataforma .NET que provee un WebService SOAP y una posee una base de datos relacional SQL Server. Si bien el objetivo final sería que el servidor procese los datos, como prueba de concepto se piensa utilizar la herramienta WEKA para probar distintas técnicas de aprendizaje de maquina para detectar patrones en los datos recopilados. Para realizar las pruebas de concepto, se desarrollo una aplicación de escritorio sobre .NET con la capacidad de filtrar y exportar la información desde la base de datos a archivos en formato ARFF para su posterior estudio con la herramienta WEKA.

En este capítulo se detallaran los diferentes aspectos tenidos en cuenta para el desarrollo del sistema, asi como las restricciones de calidad en las cuales se base el diseño y estrategias generales utilizadas en cada componente del sistema.

**1.1 Restricciones de calidad**:

Como primera medida se desea que la ejecución de la aplicación no interfiera en las medidas obtenidas relacionadas el rendimiento de la batería, por lo tanto se pone especial énfasis en un rendimiento óptimo a la hora de obtener y almacenar la información del dispositivo. Por este motivo es que se debe disminuir lo máximo posible el procesamiento realizado por el dispositivo durante la captura de los datos, y reducir además el espacio necesario para almacenar dichos datos. También, teniendo en cuenta el rápido avance tecnológico, en lo que a dispositivos móviles respecta, y las cada vez más numerosas características de estos, se busca un bajo nivel de cohesividad entre componentes y gran flexibilidad para cambios futuros, de manera tal que la aplicación pueda crecer sin mayores inconvenientes en caso de necesitar, por ejemplo, obtener información adicional referente a una nueva funcionalidad del dispositivo.

**1.2 Estrategia General:**

Dada la problemática planteada, se toman en consideración algunas restricciones de calidad con respecto al diseño del sistema que apuntan a un buen desempeño, sobre todo de la aplicación móvil, pero sin desatender la aplicación servidor. Debido a esto se propone un sistema cliente-servidor en el cual la mayor carga de procesamiento se encuentra del lado del servidor. De esta manera se limita a la aplicación cliente a la obtención de datos del usuario y evitando toda lógica adicional de procesamiento de datos. El servidor será capaz de almacenar y procesar todos los datos obtenidos.

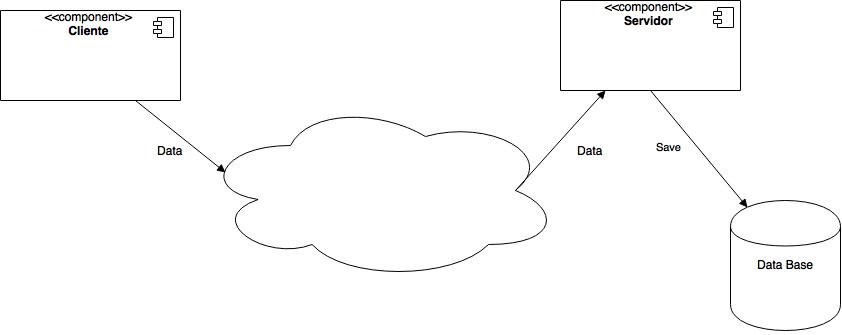


Diagrama ClienteServidor

**1.3 Estrategia Dispositivo móvil:**

Las estrategias utilizadas en la aplicación móvil están orientadas a minimizar el tiempo de procesamiento y almacenamiento con el objetivo de minimizar el consumo de energía y evitar que el registro de la información afecte indirectamente los datos obtenidos.

Teniendo en cuenta los costos de procesamiento utilizados por Android, a la hora de crear objetos en memoria dentro de una aplicación, se optó por diseñar una estructura en memoria en la cual se centraliza la información relevante del estado actual del dispositivo y a la que se puede acceder desde los diferentes componentes de la aplicación. Esta estructura se actualiza automáticamente en cada evento del sistema por lo tanto siempre se podrá encontrar en ella el estado actual del sistema. Si bien esta estrategia requiere de un mayor uso de memoria RAM este no es un recurso prioritario, y al simplificar considerablemente el acceso a los estados almacenados, evita una lógica adicional a la hora de buscar información relacionada al estado del dispositivo ahorrando tiempo de procesamiento.

Como método de almacenamiento se decidió persistir la información en archivos de texto plano, evitando el procesamiento de estructuras más complejas como XML, que si bien son más deseables a la hora de trabajar con ellas, conllevan un mayor costo de procesamiento para leer y escribir en ellas, comparados con el necesario para leer y escribir en un archivo plano. Además utilizan un mayor espacio de almacenamiento. Aunque esta decisión implica un desarrollo más complejo del lado del servidor, el cual debe ser capaz de interpretar los datos recibidos, el foco está puesto en minimizar el procesamiento en la aplicación móvil y no en el servidor.

**1.4 Estrategias Servidor**:

Se decidió utilizar SOAP WebService para realizar la comunicación de datos simplemente por una cuestión de practicidad, debido a que la plataforma .NET nos provee una interfaz amigable a la hora de crear este tipo de servicios. El método empleado para la creación del servicio SOAP es code-first. Este permite la definición de un servicio a partir del código, por lo que no es necesario tener ningun conocimiento sobre WSDL. Se desarrollan los métodos de la interface del servicio para luego generar el servicio, utilizando las herramientas provistas por .NET.

El servidor soporta la mayor carga de procesamiento del sistema, ya que se deben parsear los archivos enviados por cada usuario, para luego almacenarlos de manera organizada.

A su vez el WebService proveerá los métodos capaces de exportar la información desde la base de datos, a archivos en formato ARF para su posterior estudio.

**1.5 Estrategia Base de Datos:**

La información se almacena en una base de datos relacional SQL Server Express 2005, la cual consta de un motor muy poderoso y optimizado con el que se pueden realizar consultas rápidas, lo cual es un beneficio a la hora de extraer los datos para su estudio.

La base de datos contiene grandes volúmenes de información, los cuales podrán ser consultados mediante la utilización de filtros de la manera más adecuada a la hora de exportar los archivos ARF.

Con el objetivo de minimizar el espacio utilizado por la base la datos relacional, se decidió compartir datos entre usuarios, de esta manera se evita la generación duplicada de información, para que ante un caso, por ejemplo, de que dos usuarios utilicen la misma red Wifi, la información de esta se almacene una única vez.

**Capitulo 4**

**Solución:**

**1 Aplicación Android:**

**1.1 Conceptos básicos:**

Dentro del sistema se hace uso de tres conceptos básicos, a la hora de almacenar los diferentes tipos de indicadores de actividades del usuario. Estos conceptos simplifican la lectura de los datos recolectados y permiten cierta flexibilidad en caso de que sea necesario agregar información. A su vez permiten minimizar el espacio de almacenamiento requerido y recursos de CPU al momento de analizarlos datos.

A continuación se explicaran dichos conceptos básicos, así como también se mostrarán ejemplos prácticos para su mejor comprensión.

**1.1.1 Type:**

Para una buena organización de la información obtenida es necesario definir una estructura simple y ordenada. De esta manera surge la noción de tipos de log: Type. Cada tipo de log engloba un conjunto de datos propios de la característica que monitorea. Algunos ejemplos de tipos de log pueden ser: WIFI, BLUETHOOT, BATTERY, ACTIVITY, CONNECTION, LOCATION, etc. Cada uno de estos se corresponde con una característica específica del dispositivo y contiene un número variable de propiedades relacionadas a esta.

**1.1.2 Property:**

Una vez definidos los tipos de logs, es necesario definir cuál es el conjunto de datos de interés que se quieren almacenar. El conjunto de propiedades definidas para cada tipo de log, identifica cuales son los valores más relevantes, relacionados con un evento específico dentro de la aplicación, que deben ser almacenados. Un ejemplo de propiedades, dentro del tipo de log WIFI, son la dirección ip y el nombre de la red, definidos como IP y NAME respectivamente.

**1.1.2 Value:**

Haciendo uso de los conceptos explicados hasta el momento, podemos definir la estructura base para un evento específico. Un evento monitoreado por la aplicación, almacenara un conjunto de datos definidos por el tipo del log al que pertenece y el conjunto de propiedades que lo conforman. Definimos Value como el valor almacenado en un momento dado, para una Property específica, incluida en el tipo de log monitoreado. Cabe destacar que una Property pude tener únicamente un Value en un momento dado.

A continuación se muestra un ejemplo de un evento monitoreado, con su estructura y valores.

[09:15:11]&[WIFI]&[STATE::1|SSID::”Ei200P”|MAC::C0:65:99:A7:BC:E0|IACCESS::1]

Como se puede observar en el ejemplo, adicionalmente a la estructura explicada con anterioridad, se almacena el momento exacto en el cual fue capturado el evento, con el fin de ordenar cronológicamente la información de ser necesario.

**1.2 Componentes**:

La aplicación móvil está compuesta por tres grupos de componentes, los cuales son los encargados de realizar las funcionalidades más importantes de la misma. A continuación se describen cada uno de ellos.

**1.2.1 Servicio:**

Este componente es el encargado de monitorear el dispositivo constantemente, haciendo uso de los Receivers (ver[sub:Receivers]) del sistema. Se instancia al encenderse el aparato, inicializa los Receivers y comienza el monitoreo del dispositivo. El servicio es el componente escencial de la aplicación, ya que conserva en memoria los diferentes objetos involucrados en el monitoreo y captura de la información del dispositivo. Para asegurar un monitoreo contstante, este servicio se incializa en lo que se llama modo foreground, lo que hace que siempre este en ejecución. De esta manera el sistema operativo, en caso de necesitar recursos, no considerará matar el servicio. Solo se puede detener de forma manual. Es por esta razón es que al iniciar el servicio, y siempre que el mismo se encuentre en ejecución, se muestra en la barra de estado del dispositivo una notificación de que dicho servicio se encuentra corriendo.

**1.2.2 Receivers:**

Heredando de la clase abstracta GeneralLoggingReceiver, la cual provee la funcionalidad de escritura en el log, estos componentes, permiten el monitoreo de un determinado Log-Type. Cada receiver registra un único Log-Type y durante su ejecución, al momento de detectar el evento correspondiente, obtiene un listado de pares Property-Value del sistema, que luego serán almacenados en un archivo. Dado que los dispositivos incluyen nuevas características con el rápido avance de la tecnología, esta herencia permite la inclusión de nuevos Log-Types y su correspondiente Receiver responsable del monitoreo.

Los Receivers utilizan el comportamiento provisto por las clases BroadcastReceiver y los IntentFilter, propias del sistema operativo Android. Un BroadcastReceiver es un proceso que corre en segundo plano y puede ser activado por llamadas a través de Intents. Para lograr esto se utilizan IntentFilters con los cuales se filtran las llamadas del sistema, para solo atender a las llamadas que correspondan y/o sean de interés.

El sistema Android lanza eventos, en forma de Intents, cuando ocurre un acontecimiento importante, un cambio de estado en el dispositivo. Cada Receiver, durante su creación, se suscribe al evento correspondiente. De esta manera se evita la existencia de uno o varios procesos activos, encargado de monitorear constantemente el estado del dispositivo cada intervalos fijos de tiempo, ahorrando una gran cantidad de energía.

Cada Intent utilizado por los Receivers debe ser incluido en el archivo AndriodManifest.xml de la aplicación. De esta manera al momento de la instalación se obtienen los permisos de sistema operativo para acceder la información específica de cada característica que se desea monitorear.

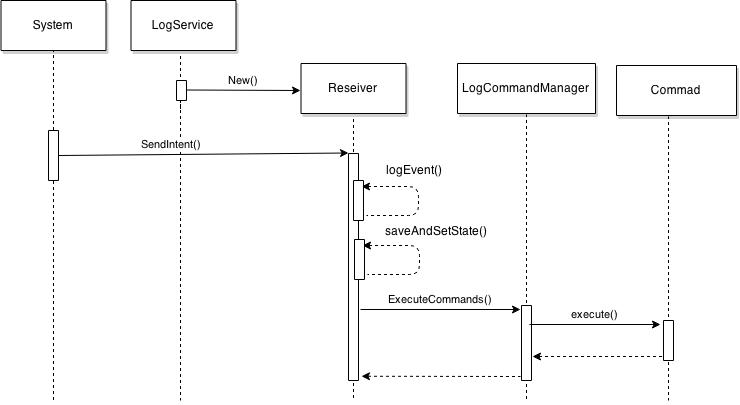


Diagrama de secuencia Log-Event se puede observar el mecanismo de monitoreo del sistema.

**1.2.3 Commands y CommandManager:**

Dentro de la aplicación existe el componente CommandManager, en el cual se mantiene un registro del estado actual del dispositivo. Esto permite saber en cualquier momento cual es el último estado en el que se encontraba el sistema. Este estado está representado por un vector de valores en un objeto del tipo HashTable. Cuando un nuevo evento es detectado por los Receivers del sistema, los datos son persistidos en el archivo de log, actualizando además dicho vector de estados.

Este componente también nos permite disminuir significativamente el espacio requerido por los archivos de logs. Los datos registrados en el log en cada evento específico, están formados por un sub-vector del vector de estados. De esta manera, solo se permite el logueo de la información si el nuevo sub-vector difiere del sub-vector correspondiente al último estado almacenado.

Hashtable<String, String> properties = l.getProperties();

Enumeration<String> enumKey = properties.keys();

**While** (enumKey.hasMoreElements ()) {

property = enumKey.nextElement

value = properties.get (property);

change |= LogCommandManager.*getInstance* ().newState (l.getType (), property, value);

}

**If** (change)

LogSave.*getInstance*().saveData (l);

El ejemplo anterior muestra que solo en el caso de que la variable “change” sea verdadera se registra la información. Este mecanismo provee flexibilidad para agregar nuevos Log-Types e incrementar el número de Properties dentro de un Log-Type, sin alterar el funcionamiento descripto.

Dentro del CommandManager también existen Commands. Cada Command hereda la interface y funcionalidad de un Command abstracto y es capaz de realizar una tarea específica, mediante el uso de su método execute. Estos objetos son utilizados para realizar tareas programadas dentro de la aplicación, como por ejemplo el envió de datos al servidor.

Es posible configurar cada Command mediante el uso de las reglas, Rules. Cada Rule contiene un sub-vector compuesto por pares Property-Value y una Key que indica el tipo de Command a ejecutar, de esta manera, cuando el sub-vector configurado coincide con el estado actual del dispositivo, el CommandManager ejecuta el Command correspondiente.

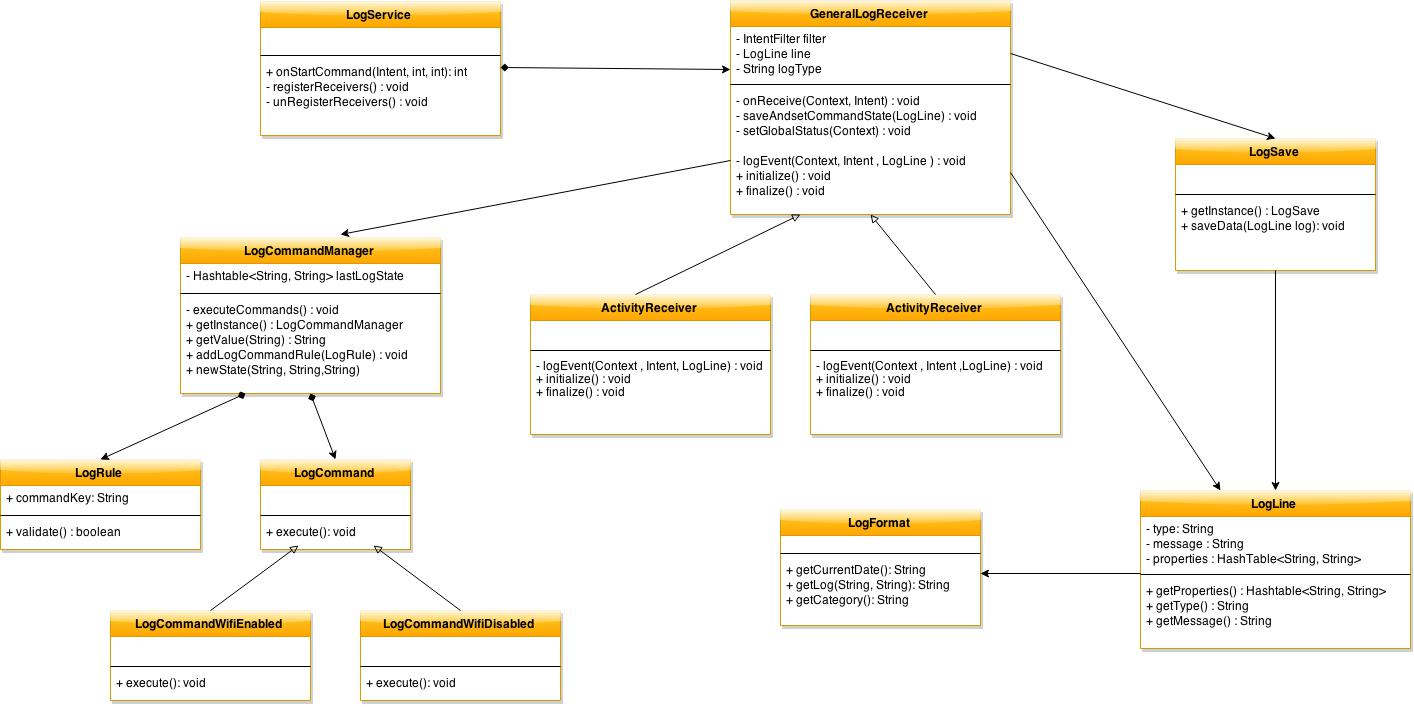


Diagrama de Clases: Aplicación Android

**1.2.4 Comunicación de Datos:**

Toda la información generada por la aplicación, será almacenada en archivos de texto plano, siguiendo el formato especifico explicado con anterioridad. Con el objetivo de almacenar la información de manera organizada, se decidió generar archivo por día dentro de un directorio creado por la aplicación. El nombre de cada archivo contiene la fecha en la cual fue generado y la información correspondiente. De esta manera también logramos reducir el espacio requerido, ya que no es necesario registrar la fecha en cada línea del archivo.

Con el objetivo de no influir en el consumo de enegía del dispositivo, a la hora de enviar los datos recolectados al servidor, la aplicación tiene configurado, por defecto, enviar los datos solo si el aparato se encuentra cargando la batería y tienen acceso a internet mediante Wifi. Así no solo no existe impacto en el consumo de batería durante el envío de datos, sino que además al evitar el uso de redes 3G, que son en general mucho mas lentas y menos estables que las redes Wifi, no existen riesgos de ocasionar gastos extras al usuario.

Para evitar particionar los datos recolectados, se decidió no enviar el archivo correspondiente al día en curso, debido a que es probable que continúe siendo utilizado. Una vez que se obtiene la lista de archivos a enviar, se genera un paquete comprimido y este es enviado al servidor.

**4.2 WebService:**

Para realizar la comunicación de datos entre el dispositivo móvil y el servidor se utiliza un WebService SOAP, el cual debido a su protocolo estandarizado nos permite comunicar sin problemas diferentes tecnologías, como son en este caso, un servidor desarrollado en la sobre la plataforma .Net con una aplicación Android. Este WebService nos provee los métodos para almacenar los datos recolectados por los usuarios en una base de datos relacional, que corre sobre un motor SQLServer Express.

Como se explicó anteriormente, es el servidor el encargado de realizar las tareas mas costosas en cuanto a recursos.

**2.1 Componentes**:

El WebService está constituído por un punto de acceso público, a través del cual los usuarios pueden enviar la información, y por diferentes componentes encargados de realizar las tareas de procesamiento y almacenamiento de los datos generados por los usuarios. A continuación se explicaran en detalle cada uno de estos componentes.

**2.1.1 Recepción de Datos:**

Utilizando el punto de acceso el usuario envía un archivo comprimido. Dentro de este paquete de datos pueden existir una cantidad variable de documentos, uno por cada día de logueo, dependiendo de cuantos días han transcurrido desde la última conexión. Una vez almacenado, en un directorio local, el WebService envía una ACK al cliente indicando que el archivo fue recibido correctamente.

**2.1.2 Procesamiento de Datos:**

Una vez finalizada la comunicación y una vez descomprimidos los documentos, el servidor comienza el procesamiento de los datos recibidos en el paquete. El procesamiento de los datos se realiza de manera secuencial por orden cronológico. El sistema lee cada documento realizando un parseo de sus datos, obteniendo los Log-Types, sus Properties y Values, para luego almacenar los datos en la base de datos relacional. Cuando finaliza el procesamiento de los datos, los documentos son eliminados conservando un backup del paquete recibido.

**2.1.3 Almacenamiento de Datos:**

Los paquetes recibidos por el WebService son almacenados en un directorio para cada usuario específico, creado por el mismo servicio. En el caso de existir un error en el procesamiento de un documento, este no será eliminado del directorio. De esta manara es posible verificar fácilmente los datos procesados incorrectamente. Los datos procesados correctamente son almacenados en la base la base de datos, en la cual se registran la información relacionada a sus documentos y el usuario de origen. De esta manera es posible localizar para cada dato almacenado, cual fue su paquete de origen, así como su documento dentro de este y a que usuario pertenece.

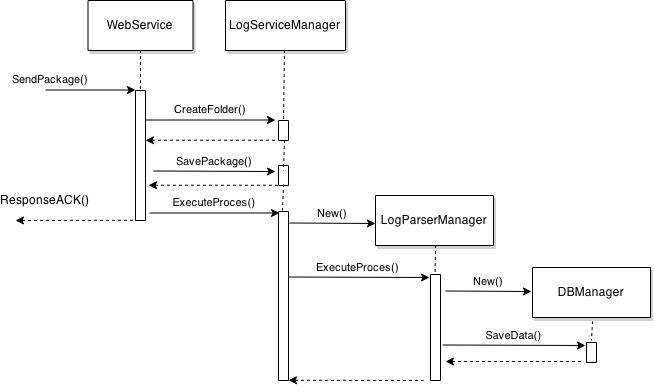


Diagrama de Secuencia, procesamiento del paquete de datos.

**4.3 Base de Datos:**

Debido a la gran cantidad de información obtenida por la aplicación y la necesidad de consultar de diferentes formas los datos obtenidos para la búsqueda de patrones, es indispensable la correcta organización de los datos y la utilización de una estrategia que nos permita obtener información de manera eficaz. Es por este motivo que se utiliza una base de datos relacional, controlada por el motor de bases de datos SQLServer Express.

**4.3.1 Estructura:**

Dentro de la base de datos, la información se encuentra organizada en diferentes tablas de manera tal que nos permita optimizar el espacio utilizado A continuación se describen cada una de las tablas utilizadas y de qué manera se utiliza su información.

User

Almacena la información de los usuarios registrados en el sistema. En base a los datos de esta tabla se puede obtener la fecha de la última conexión, la versión y modelo de dispositivo utilizado y un identificador único del aparato.

File

Almacdena los datos correspondientes a los documentos procesados por el servicio, se pueden obtener datos como el tamaño, nombre y a que usuario pertenece.

LocationGroup

Almacena los LocationGroups pertenecientes a cada usuario, ubicación del LocationGroup y la cantidad de localizaciones que contiene.

Type

Contiene el listado de LogTypes utilizado por la aplicación a la hora de registrar eventos en el dispositivo.

Property

Almacena un listado de cada posible LogProperty relacionada con su LogType correspondiente.

PropertyValue

Se encuentran los posibles valores para cada LogProperty, los cuales pueden ser compartidos por diferentes usuarios. Un ejemplo recurrente es el caso en el que dos usuarios se conectan a una misma red Wifi.

Rule

Contiene las reglas especificadas para cada usuario del sistema, así como un código que indica que comando se deberá ejecutar cuando esta se cumpla.

Condition

Almacena las condiciones de una regla específica. Cada una de estas indica un LogProperty y su PropertyValue necesarios para que la condición pueda ser validada.

