Android 2.3.3 o superior

.NET 2008 Framework 3.5

Web Service ASMX SOAP

SQL Server 2005 Express

**Capitulo 3**

**Propuesta:**

**3.1 LoguinUse :**

Para enfrentar la problemática explicada en capítulos anteriores se propone LoguinUse, una aplicación para dispositivos móviles Android, versión 2.3.3 o superior, esta plataforma provee un numero aceptable de usuarios, gracias a su popularidad y debido a que es una versión estable que se utiliza en la mayoría de los dispositivos. La información obtenida por dicha aplicación será comprimida y enviada a un servidor dedicado que provee un WebService SOAP. Este servidor, se desarrolla sobre la plataforma .NET Framework 3.5 (FW 3.5) y una base de datos relacional SQL Server 2005 Express. También se crea una aplicación de escritorio sobre FW 3.5 con la capacidad de filtrar y exportar los datos a archivos en formato ARF para su posterior estudio mediante métodos de Datamining con la herramienta WEKA.

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

**3.1.1 Restricciones de calidad**:

Como primera medida se desea un rendimiento óptimo a la hora de almacenar la información obtenida en el dispositivo. Los puntos que se atacan son principalmente la disminución del poder de procesamiento al mínimo posible y como segunda medida, disminuir el espacio de almacenamiento requerido para almacenar los documentos generados.

También, teniendo en cuanta 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 un buen soporte para cambios futuros, de manera tal que la aplicación pueda crecer sin mayores inconvenientes en caso de necesitar, por ejemplo, obtener más información referente a una nueva funcionalidad del aparato.

**3.1.2 Estrategia General:**

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

**3.1.3 Estrategia Dispositivo móvil:**

Teniendo en cuanta los costos de procesamiento utilizados por Android a la hora de crear objetos en memoria dentro de una aplicación, se optó por mantener alojados en memoria objetos que de otra manera no hubieran sido necesarios. Si bien esta estrategia requiere de un mayor uso de memoria RAM, también simplifica el acceso a los estados almacenados por la aplicación evitando la creación de objetos para la lectura de estados en cada evento monitoreado del sistema.

Como método de almacenamiento, se decidió persistir la información en archivos de texto plano, esta estrategia evita el procesamiento de estructuras como XML que si bien son más deseables a la hora de trabajar con ellas conllevan un costo superior al de un archivo plano y utilizan un mayor espacio de almacenamiento. Esta decisión implica un desarrollo más complejo del lado del servidor ya que se deben crear los algoritmos capases de interpretar los datos obtenidos.

**3.1.4 Estrategias Servidor**:

Se decidió utilizar SOAP WebService para realizar la comunicación de datos, debido a que la plataforma .NET nos provee una interface amigable a la hora de crear este tipo de servicios. El método empleado para la creación del servicio SOAP es code-first. Dicho método consiste en desarrollar los métodos de la interface del servicio para luego generar el servicio, en base a este, utilizando herramientas provistas por el fremawork .NET 3.5.

Este 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 capases de generar en archivos en formato ARF la información requerida para el estudio de los datos.

**3.1.5 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 cual se pueden realizar consultas rápidas y a medida a la hora de estudiar la información recolectada.

La base de datos contiene un gran volumen 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 es espacio utilizado por la base la datos relacional, se decidió compartir datos entre usuarios, de esta manera se evita la generación duplicada de información, de esta manera en el caso, por ejemplo, de que dos usuarios utilicen la misma red Wifi, la información de esta no será duplicada en el repositorio.

**Capitulo 4**

**Solución:**

**4.1 Aplicación Android:**

**4.1.1 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 a la hora de agregar información en caso de ser necesario, a su vez nos permiten minimizar el espacio requerido y recursos de CPU a la hora de analizar los datos.

En esta sección se explicaran los conceptos básicos utilizados así como ejemplos prácticos para su mejor comprensión.

**4.1.1.1 Type:**

Para una buena organización de la información obtenida es necesario definir una estructura simple y ordenada de datos. 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 de estos tipos de log son WIFI, BLUETHOOT, BATTERY, ACTIVITY, CONNECTION, LOCATION, etc. Un tipo de log se corresponde a una característica específica del dispositivo y contiene un número variable de propiedades relacionadas a esta.

**4.1.1.2 Property:**

Una vez definidos los tipos de logs, es necesario definir cuál es el conjunto de datos de interés a la hora de almacenar la información. El conjunto de propiedades definidas para un tipo de log identifica cuales son los valores a tener en cuenta en el momento de almacenar un evento específico dentro de la aplicación. Una propiedad es, por ejemplo, dentro del tipo de log WIFI, la dirección ip y el nombre de la red definidos como, IP y NAME respectivamente.

**4.1.1.2 Value:**

Haciendo uso de los conceptos explicados hasta el momento podemos definir la estructura base para un evento especifico, en consecuencia un evento monitoreado por la aplicación almacenara un conjunto de datos definidos por el tipo del log al que pertenezca y el conjunto de propiedades que lo conformen.

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

**4.1.2 Componentes**:

La aplicación correspondiente al dispositivo móvil está compuesta por tres grupos de componentes encargados de realizar las funcionalidades más importantes de la aplicación, a continuación se describen cada uno de ellos:

**4.1.2.1 Servicio:**

Este componente es el encargado de monitorear el dispositivo constantemente, haciendo uso de los *Receivers* del sistema. Inicializado al encender el aparato, inicializa los *Receivers* comenzando el monitoreo del dispositivo. Existe la posibilidad de que el sistema operativo elimine el servicio en caso de necesitar recursos para otros procesos, en este caso el *Servicio* se reinicia automáticamente.

**4.1.2.2 Receivers:**

Heredando de una clase abstracta *GeneralLoggingReceiver* la cual provee la funcionalidad de escritura en el log, permiten el monitoreo de un *LogType*. Cada receiver registra un único *LogType* y durante su ejecución, al momento de detectar el evento correspondiente, obtiene un listado de pares Property-Value, del sistema, que luego serán almacenadas en un archivo. Dado que los dispositivos incluyen nuevos features a medida que avanza la tecnología, esta herencia permite la inclusión de nuevos *LogTypes* y su correspondiente monitoreo.

Los *Receivers* utilizan el comportamiento provisto por las clases BroadcastReceiver y los *IntentFilter*. Un roadcastReceiver es un proceso que corre en segundo plano y puede ser activado por llamadas a través de Intents. Para lograr esto se utilizan IntentFiltes con los cuales filtrar las llamadas del sistema y solo atender a las que correspondan.

El sistema Android lanza eventos, en forma de *Intents*, cuando ocurre un acontecimiento importante, en este caso un cambio de estado en el aparato. Cada Receiver, durante su creación, se suscribe al evento correspondiente. De esta manera se evita la existencia de un proceso demonio encargado de monitorear 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 feature.

**4.1.2.3 Commands y CommandManager:**

Dentro de la aplicación existe el componente CommandManager, en el cual se mantiene un registro del estado actual del dispositivo. Este estado nos permite saber en cualquier momento cual es el último estado del 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 sin persistidos en el archivo del log, en este momento también se actualiza el 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 en el estado general.

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);

En el ejemplo se puede ver que solo en el caso de que la variable “change” sea positiva se guarda la información.

Este mecanismo provee la flexibilidad para agregar nuevos LogTypes e incrementar el número de properties dentro de un LogType, 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 Rules. Cada Rule contiene un sub-vector compuesto con 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.

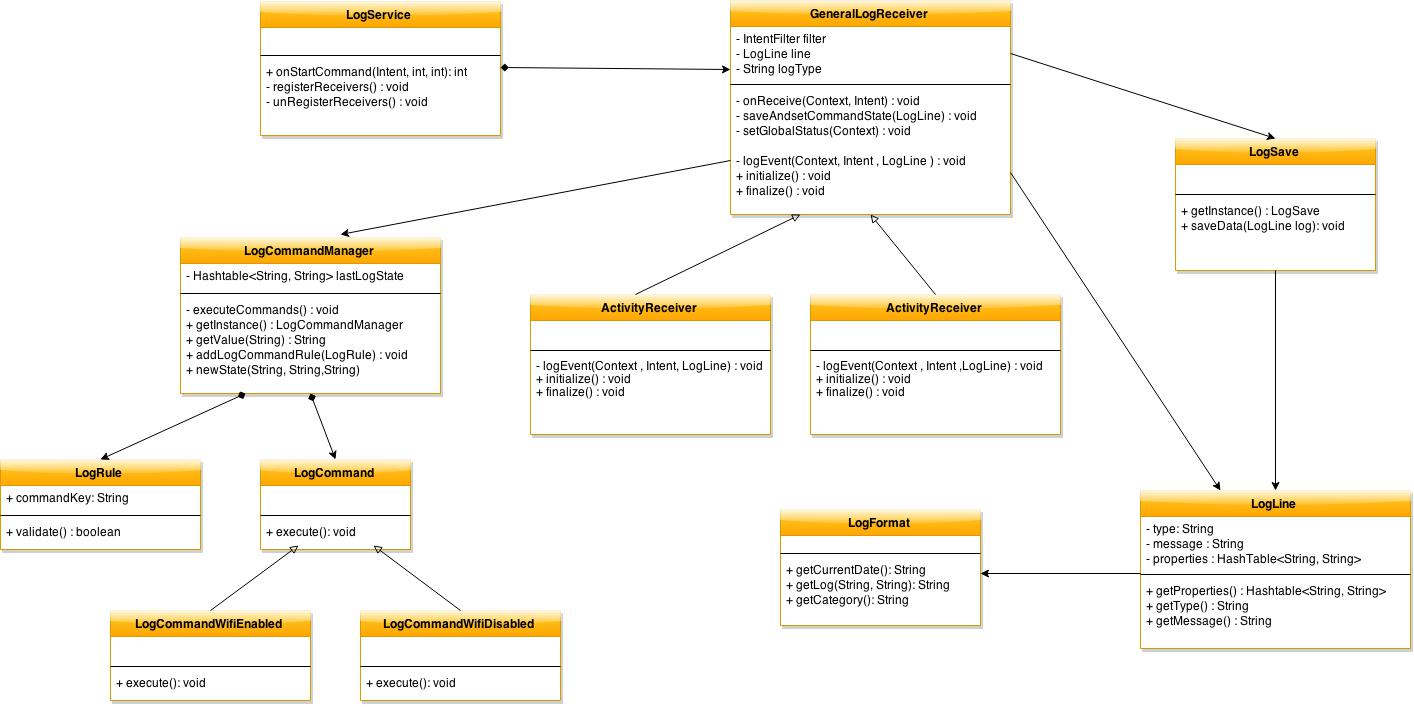


Diagrama de Clases: Aplicación Android

**4.1.2.4 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 dia 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 loguear la fecha en cada línea del archivo.

Con el objetivo de no alterar el uso de la energía a la hora de enviar los datos recolectados al servidor, la aplicación tiene configurado por defecto, enviar los datos solo si, el aparato está actualmente cargando la batería y tienen acceso a internet por Wifi, de esta manera ahorramos el consumo de batería en el envío de datos y no utilizamos la red 3G, con la cual existen más riesgos de errores y puede ocasionar gastos extras al usuario. Con el fin de enviar siempre archivos completos y no particionar los datos generados para un día específico, se decidió no enviar el archivo correspondiente al día en curso ya que este está 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, nuestro servidor .Net con la aplicación Android.

Este WebService nos provee los métodos para almacenar los datos recolectados por los usuarios en una base de datos relacional, desarrollada con SQLServer Express.

Como se explicó anteriormente, es el servidor el encargado de realizar las tereas de procesamiento más pesadas en cuanto a recursos.

**4.2.1 Componentes**:

El WebService está compuesto 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.

**4.2.1.1 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 el directorio el WebService envía una ACK al cliente indicando que el archivo fue recibido correctamente.

**4.2.1.2 Procesamiento de Datos:**

Una vez finalizada la comunicación y una vez descomprimidos los documentos, el servidor comienza el procesamiento de los datos recibidor 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 LogTypes y sus Properties para luego almacenar los datos en la base relacional.

Al momento de finalizar el procesamiento los documentos descomprimidos son eliminados y el sistema conserva el backup del paquete recibido.

**4.2.1.3 Almacenamiento de Datos:**

Los paquetes recibidos por el WebService son almacenados en un directorio creado por el mismo servicio, para cada usuario específico. 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. A su vez los datos procesados correctamente son almacenados en la base la base de datos, en la cual se registran la información relacionada a sus documentos de origen y el usuario responsable. 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.

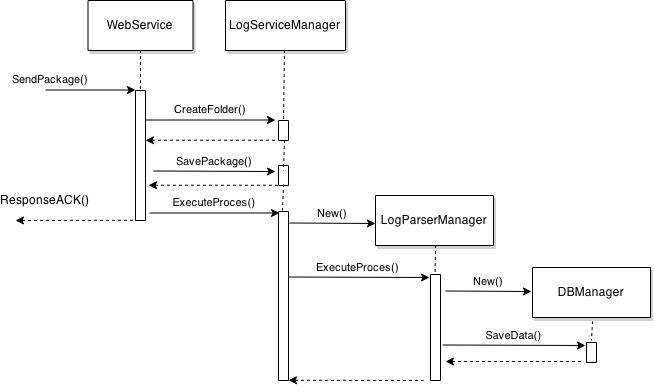


Diagrama de Secuencia, procesamiento del paquete de datos.

**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 utilizar 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: Se 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 identificar único del aparato.

File: en esta tabla se almacenan 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 locations 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, estos valores 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: se almacenan las condiciones de una regla específica. Cada una de estas indica un LogProperty y su PropertyValue necesarios para que la Condition pueda ser validada.

