SISTEMA DE TELECONTROL MÓVIL BASADO EN OPEN IEC 61850

DOMICIANO RINCÓN NIÑO

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERÍA
INGENIERÍA ELECTRÓNICA
SANTIAGO DE CALI
2014

SISTEMA DE TELECONTROL MÓVIL BASADO EN OPEN IEC 61850

DOMICIANO RINCÓN NIÑO

Trabajo de grado presentado para optar al título de Ingeniero Electrónico

Director

Fabio German Guerrero Moreno M.Sc

Codirector

Asfur Barandica López M.Sc

UNIVERSIDAD DEL VALLE

FACULTAD DE INGENIERÍA

INGENIERÍA ELECTRÓNICA

SANTIAGO DE CALI

2014

DEDICATORIA

A mi padre por su apoyo incondicional, por su esfuerzo de procurar siempre mi bienestar y por legarme el deseo de tener una profesión.
A mi hermana por ser mi ejemplo enseñándome el camino que ha recorrido.
A mi familia por ser el soporte tanto en mi proceso académico como en mi vida.
A mis compañero y amigos por los momentos compartidos.

AGRADECIMIENTOS

Agradezco a los profesores Fabio Guerrero y Asfur Barandica por la disposición mostrada desde el principio a dirigir mi proyecto de grado, por su dedicación, sus consejos, su paciencia y el conocimiento aportado que me facilitaron la realización del mismo.

A los profesores que hicieron parte de mi proceso académico por sus valiosas lecciones en el ámbito profesional y personal.

Agradezco también a mis compañeros por brindar una atmosfera de amistad y hermandad en medio de la dificultad de las obligaciones académicas.

A mi familia porque todo lo que soy se lo debo a ellos.

CONTENIDO

RESUMEN

El presente documento del trabajo de grado titulado “Sistema móvil de telecontrol basado en OPEN IEC 61850” tiene como objetivo general del trabajo de grado es desarrollar una aplicación de telecontrol en un dispositivo móvil, para un dispositivo electrónico inteligente (IED) usando OPEN61850.

Debe permitir entablar una comunicación inalámbrica con equipos de control sobre la planta que componen una red de comunicaciones basada en la norma de automatización de subestaciones eléctricas llamado IEC 61850.

Para lograrlo se usa el proyecto abierto OPEN IEC 61850, desarrollado por Fraunhofer ISE, que implementa en clases de JAVA el modelo de comunicación virtual presente en la norma.

El proyecto de grado surge como consecuencia de la necesidad en la industria eléctrica de un monitoreo no presencial de las subestaciones eléctricas que mantenga al operario al pendiente de la actividad de la planta de modo que la información de su estado sea distribuida y conocida desde una ubicación aleatoria y que permita además tomar acciones de control en función a la información recibida.

Con esto se evita la posición fija de las estación de mando al interior de la subestación ya que dejarla sola por atender otros eventos como una necesidad fisiológica, significa un aumento en el tiempo de respuesta si se llega a generar un evento crítico en la planta.

Para lograr implementarlo se propone los siguientes objetivos específicos:

- ✓ Familiarizarse con el proyecto abierto OPEN IEC 61850
- ✓ Crear una representación virtual de los datos recibidos y de los nodos lógicos del IED.
- ✓ Desarrollar una aplicación web que permita interactuar con un dispositivo electrónico inteligente.
- ✓ Diseñar e implementar la aplicación para ser ejecutada desde un dispositivo móvil inteligente.

El documento del trabajo de grado muestra el desarrollo y cumplimiento de los objetivos específicos capítulo a capítulo, que están dispuestos de la siguiente forma:

Inicialmente se contextualiza al lector en temas trascendentales del trabajo de grado en el capítulo de introducción. En él se trata la norma IEC 61850, OPEN IEC 61850 y el sistema operativo para dispositivos móviles Android sobre el cual funciona la aplicación de telecontrol brindando la información más relevante enfocada al contenido de la aplicación y sus capacidades. Adempás en el Anexo 1 se encuentra un capítulo sobre SCL (Substation Configuration Language) donde

se muestra la configuración y creación de modelos de IED mediante este lenguaje. Si el lector no está familiarizado con el tema no se debería obviar.
El segundo capítulo llamado OPEN IEC 61850 se realiza concretamente para cumplir con el primer objetivo específico exponiendo la anatomía del proyecto abierto catalogando por función las clases que lo componen y haciendo un paralelo con el contenido de la norma.
El tercer capítulo expone la metodología usada para el desarrollo del sistema móvil de telecontrol y los módulos que comprende la aplicación para cumplir con el segundo, tercer y cuarto objetivo específico. Esto se logra usando el sistema operativo Android y sus elementos de interacción con el usuario para atribuirles funciones de OPEN IEC 61850 lo cual dota a la aplicación de los medios para lograr interacción con los IED que componen la red de comunicación industrial basada en IEC 61850.
Seguidamente se presentan los resultados para lo cual se hacen pruebas de la aplicación en funcionamiento con IED simulados y reales. Se expone además las limitaciones de la aplicación y los logros en el contexto de la norma.
Finalmente se presentan las conclusiones de trabajo de grado, sintetizando los logros y dificultades durante el desarrollo, mencionando las limitaciones de las herramientas utilizadas y el trabajo futuro del sistema de telecontrol.

1. INTRODUCCIÓN

El proyecto de grado está fundamentado en cuatro temas de suma trascendencia los cuales son introducidos a continuación para contextualizar teóricamente el proyecto de grado.

1.1 NORMA IEC 61850

IEC 61850 es una norma de comunicación industrial para la automatización de subestaciones eléctricas en la que se modela la estructura de estas mediante un lenguaje propio (SCL), se define la interacción entre los elementos en la red de comunicación y los servicios que cada uno ofrece a la subestación. Estos servicios pueden ser de protección, control, medición, monitoreo, etc. según las funciones que ofrezca cada equipo que la compone.

Se describe en varios protocolos de nivel de aplicación, como se verá más adelante, algunos de ellos están enfocados a la inmediatez, otros a la seguridad e integridad de los datos. Esta diferencia permite el uso de protocolos intermedios como IP, TCP, UDP soportados sobre Ethernet lo que supone gran versatilidad en el uso de la norma.

Fue creada con el fin de estandarizar y unificar las comunicaciones industriales de las subestaciones eléctricas, por lo cual introduce el concepto de interoperabilidad, lo que permite que dispositivos de distinto fabricante coexistan funcionalmente en una red IEC 61850 gracias a formatos y mecanismos de mensajería propios de la red.

Los equipos involucrados con la planta, que ofrecen servicios a la subestación y a su vez efectúan intercambio de información a través de la red se denominan IED (Intelligent Electronic Device). La norma modela representaciones virtuales de los IED permitiendo hacer la descripción de su funcionamiento con el mismo lenguaje SCL. Los IED pueden tener desde un simple disyuntor que permita la desconexión de un transformador, hasta un complejo sistema de protección que prevenga daños por temperatura, corriente o voltaje elevados.

IEC 61850 consta de una serie de documentos escritos que describen cada requisito físico y virtual que debe tener una red IEC 61850. En total son diez partes que presentan entre otros los requisitos generales, el lenguaje de configuración de subestación (SCL) y la estructura física y virtual de comunicación.

1.1.1 Modelo OSI

IEC 61850 se describe en cuatro distintas pilas OSI como se muestra en la figura 1 cada una encabezada por un protocolo del nivel de aplicación.

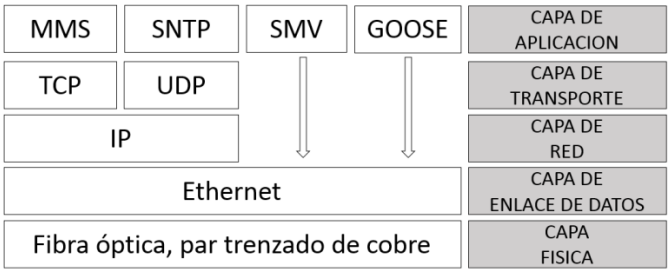


Figura 1. Modelo OSI de la norma IEC 61850. (Figura tomada de “Implementation Issues with IEC 61850 Based Substation Automation Systems”)

(A) Capa física

IEC 61850 no exige el uso de una tecnología específica La longitud del cableado debe ser razonable y presentar el mínimo de fallas en la transmisión abogando por un medio limpio y libre de interferencias. Por esto se tiene recomienda el uso de fibra óptica (FX o 10-base FL) o el par trenzado de cobre (10-base T).

(B) Capa enlace de datos

Para satisfacer el requisito del uso de Ethernet el formato de la trama puede estar basado en IEEE 802.3 o en IEEE 802.1Q que incluyen campos para definir prioridades en la trama a enviar y para el uso de VLAN útil para los protocolos de encapsulamiento directo (SMV y GOOSE).

A continuación se describen los protocolos de nivel de aplicación:

(C) GOOSE (Generic Object Oriented Substation Event)

Es un mecanismo que permite el envío de un mensaje a varios o a todos los integrantes de una red industrial de subestación basada en IEC 61850. La dispersión del mensaje es de temporización crítica, lo que significa que tiene un tiempo mínimo de entrega. El mensaje se trasmite a una dirección MAC multicast o broadcast.

GOOSE puede contener comandos, alarmas, indicadores o mensajes. El objetivo de este mecanismo es dar a la operatividad de la planta la característica de tiempo real aprovechando los beneficios de la tecnología Ethernet.

En estos casos la trama de la capa se basa en IEEE 802.1Q que permite seleccionar el grado de prioridad de cada mensaje y la VLAN por donde transita. El mensaje GOOSE se encapsula directamente en la trama Ethernet y se envía con el fin de reducir tiempos de procesamiento y transmisión para minimizar el tiempo de entrega.

(D) SMV (Sample Measurement Value)

Al igual que un mensaje GOOSE, un mensaje SMV se encapsula directamente en la trama 802.1Q y consiste el envío de valores análogos de los transductores de un IED a otros IED o a los HMI que compongan la red. Se usa también una alta prioridad en el envío de estos mensajes para conocer el comportamiento de la planta en tiempo real.

(E) SNTP

Se usa el SNTP (Simple Network Time Protocol) como alternativa para el envío de mensajes sincronizando los relojes de dos o más sistemas finales sobre una red

basada en IEC 61850. Se usa para funciones críticas en el tiempo, esencialmente útil en relevo de tiempos de dos sistemas separados, por ejemplo la desconexión secuencial de la red eléctrica primaria con dos IED con funciones de conmutación con accionamiento a distancia.
SNTP es un protocolo que se transporta mediante UDP dados los requerimientos de tiempo

1.1.2. MMS (Manufacturing Message Specification)

Este tipo de mensajes son orientados a la conexión, por esta razón se soporta en TCP/IP. Cada miembro de la red entonces se contacta por medio de su IP.
MMS es un estándar internacional descrito en la norma ISO 9506 diseñado para ambientes industriales. Consiste en un sistema de mensajería para el intercambio de datos entre dos dispositivos o dos aplicaciones computacionales sobre una infraestructura de red basado en servicios. Este intercambio tiene como objetivo el control o la supervisión de un equipo en un ambiente de producción en tiempo real y dada la condición de estándar, MMS tiene la posibilidad de usarse en dos dispositivos de distinto fabricante que implementen el estándar.
Es un protocolo de nivel de aplicación y está basado en la arquitectura cliente/servidor. Estructuralmente hablando, se define un objeto principal llamado VMD (Virtual Manufacturing Device) que presenta los parámetros para identificarse en la red y poder ser contactado. Este objeto contiene una representación virtual de un dispositivo real mediante la descripción en objetos definidos en el estándar. Se definen también los servicios que presta el servidor. MMS no especifica el lenguaje de programación que deba usarse para la definición los objetos y servicios.

(A) OBJETOS MMS

Los objetos MMS se usan para la descripción virtual de un equipo real. Los más relevantes son:
VDM, El dispositivo real representado en un objeto virtual.
Domain, Representa un recurso del VDM.
Program Invocation, La puesta en marcha de varios Domain
Varible, Un elemento con un tipo de dato especifico como entero, booleano, etc.
Type, Descripción de un formato de datos
Named Variable List, Lista de variables con nombre propio.
Semaphore, Objeto que regula el acceso a un recurso compartido.
Operation Station, Display y teclado para que un operador controle el VDM.
Event Condition, Un objeto que representa el estado de un evento
Event Action, La acción que se toma cuando un Event Condition cambia
Event Enrollment, Decide a quien dentro de la red notifica sobre el cambio de un Event Condition
Journal, Reporte de eventos y variables en el tiempo.
File, Un archivo en un servidor de archivos

(B) SERVICIOS MMS

Los objetos mostrados anteriormente tienen cada uno una serie de servicios asociados. Los servicios más importantes se muestran a continuación:
Read. El servidor debe responder con el valor de la variable cuestionada.

Write. El servidor debe cambiar el valor de la variable mencionada por el cliente.
InformationReport. Reporta el valor de las variables al cliente de forma no solicitada. Útil para el manejo de alarmas o mensajes preventivos.
GetVariableAccessAttributes. El servidor responde con la dirección de acceso y valor de la variable solicitada por cliente.
GetNameList. El servidor responde con las direcciones de acceso de la lista de variables solicitada por el cliente.
DefineNamedVariable. Permite al cliente crear dentro del VDM una variable.
DeleteVariableAccess. Permite eliminar una variable si es que es de tipo borrable.
CreateNamedVariableList. Le permite al cliente crear una Named Variable List.
GetNamedVariableListAttributes. El servidor responde al cliente con los valores de la NamedVariableList.
DeleteNamedVariableList. Le permite al cliente borrar una NamedVariableList.

(C) ESTRUCTURA MMS

La figura 2 ilustra la estructura MMS ilustrando la ubicación de los objetos y el funcionamiento de los servicios.

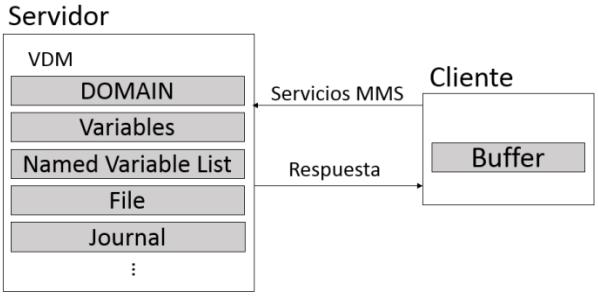


Figura 2. Estructura MMS en la interacción Cliente-Servidor

Como se observa en la figura 2 un cliente MMS puede no tener objetos MMS pero es preferible que contenga al menos un elemento de memoria para aprovechar la información que le envía el servidor. Luego del VDM, el domain es el objeto más importante y los otros objetos MMS sirven para la descripción del recurso que representa. La relación entre objetos MMS generalmente es de tipo jerárquica de modo que las variables y archivos están contenidos en un domain por ejemplo, aunque no es exigencia del estándar.

1.1.3 ACSI (Abstract Communication Service Interface)

La norma IEC 61850 ha adoptado a MMS y ha creado sus propios objetos y servicios pero están basados en los ya descritos en MMS. A este conjunto modificado se le ha nombrado ACSI (Abstract Communication Service Interface).
La norma además ha definido una base de tipos de datos que los objetos puedan acoger para su representación. Números enteros, cadenas de caracteres, booleanos, etc. son algunos ejemplos de tipos que las variables MMS pueden tomar para que permitan la lectura o manipulación de los eventos que acontecen en el dispositivo real a través de variables asociadas a estos eventos que están contenidas en el VDM. Por ejemplo la medición del voltaje de la etapa primaria de

un transformador monofásico mediante un IED, debe tener alguna variable asociada de tipo numérico, como punto flotante de doble precisión para representar este voltaje.
Los tipos de datos que presenta la norma IEC 61850 están definidas en otro estándar ISO, el ANS.1/VER (ISO 8824).
Al ser una implementación de MMS, los objetos y servicios ACSI corresponden a objetos y servicios MMS. En la tabla 1 se presentan las equivalencias entre los dos modelos de comunicación industrial.
Los objetos ACSI hacen referencia al modelo virtual de IED del que más adelante se hablara.

Objeto ACSI	Equivalente MMS
SERVER	VDM
LogicalDevice	Domain
LogicalNode	NamedVariable
DataObject	NamedVariable
DataSet	NamedVariableList
ReportControlBlock	NamedVariable
Report	Journal
Files	Files

Tabla 1. Comparación entre objetos ACSI y objetos MMS

En la tabla 2 se muestran las equivalencia de los servicios que presta el VDM entre ACSI y MMS

Servicio ACSI	Equivalente MMS
LogicalDeviceDirectory	GetNameList
GetAllDataValues	Read
GetDataValues	Read
SetDataValues	Write
GetDataDirectory	GetNameList
GetDataDefinition	GetVariableAccessAttributes
GetDataSetValues	Read
SetDataSetValues	Write
CreateDataSet	CreateNamedVariableList
DeleteDataSet	DeleteNamedVariableList
GetDataSetDirectory	GetNameList
Report (Buffered y Unbuffered)	InformationReport
GetBRCBValues/GetURCBValues	Read
SetBRCBValues/SetURCBValues	Write
Select	Read/Write
Cancel	Write
Operate	Write

Tabla 2. Comparación entre servicios ACSI y servicios MMS

1.1.2 Tipos de mensaje

La norma define siete tipos de mensaje que, según los requerimientos de tiempo, corresponden a uno de los cuatro mapeos que se enuncian en la figura 1 lo cual se muestra en la tabla 3:

Mensaje	Descripción	Protocolo de aplicación
Type 1	GOOSE (De tiempo Critico)	GOOSE
Type 2	Medium Speed Message	MMS
Type 3	Low Speed Message	MMS
Type 4	Raw data sample (De tiempo Critico)	SMV
Type 5	File Transfer Function	MMS
Type 6	Time Synchronization Message	SNTP
Type 7	Command Message with access control	MMS

Tabla 3. Tipos de mensaje de la norma IEC 61850

1.1.3. Estructura de una red IEC 61850

Una red de comunicaciones IEC 61850 es una red de área local (LAN) que permite el intercambio de información entre IED y otros equipos de red presentes en la planta. La red LAN puede hacer uso de enrutadores y switchs para formar topologías de estrella, bus o anillo, esto con el fin de distribuir la información de forma eficiente a las posibles conexiones con los HMI de una estación de ingeniería o un Gateway hacia la internet pública.

La norma plantea una estructura de tres niveles para la localización de todos los componentes de una red IEC 61850:

(A) Nivel de estación

El nivel donde se encuentra la estación de ingeniería. En este nivel se emiten comandos a la planta o se miden variables físicas en la misma desde equipos de cómputo con la capacidad de almacenar la información proveniente.

(B) Nivel de Bahía

Este nivel corresponde a los IED que se usen para tomar control de la planta. Generalmente se encuentran cerca de la planta pero refugiados en un armario.

(C) Nivel de proceso

En este nivel se encuentran los equipos que actúan directamente en la planta como Switchs o Sensores. La figura 3 muestra la relación entre niveles.

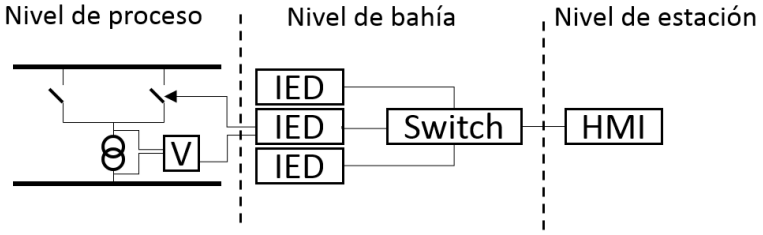


Figura 3. Relación entre los niveles de IEC 61850

Por otra parte por medio del internet se puede llegar a tener varias subestaciones conectadas por medio de un Gateway hacia la red y por medio de un NAT para la identificación de cada elemento de la LAN para equipos por fuera de ella como se muestra en la figura 4.

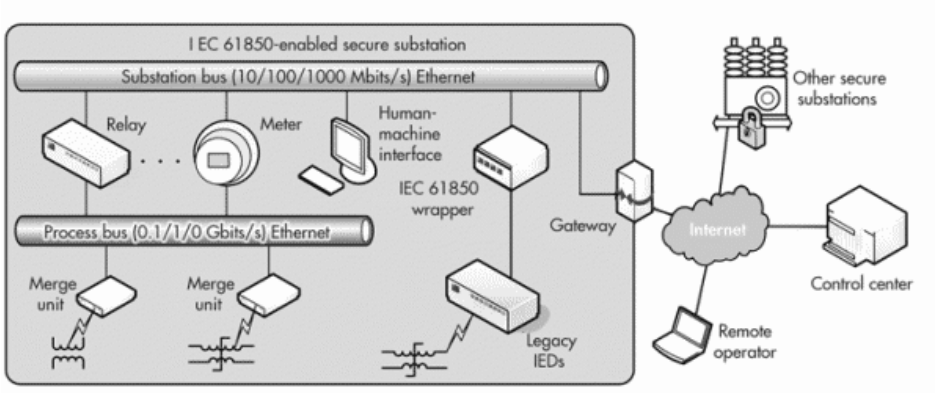


Figura 4. Ejemplo de arquitectura de red en IEC 61850. Figura tomada de la página web Nettedautomation en el enlace de IEC 61850.

Finalmente, algunos IED antiguos que no implementen la norma IEC 61850 pueden ser añadidos a la red mediante un IEC 61850 wrapper.

1.1.3.1. Requisitos de una arquitectura de red según el protocolo

Como se mostró anteriormente, la norma IEC 61850 presenta cuatro tipos de pilas OSI cada una con una capa de aplicación diferente. Es justamente esta diferencia la que permite determinar las exigencias de una arquitectura u otra.

En el caso de GOOSE y SMV la topología de red puede ser creada gracias al uso de Switchs (Capa 2). Cada IED dentro de la red se identifica únicamente con su dirección MAC.

En el caso de SNTP y MMS la arquitectura de red debe tener por obligación enrutadores (Capa 3) que soporten el enrutamiento IP ya que sus pilas OSI están basadas en protocolos de aplicación que se soportan en el protocolo IP. De esta forma cada equipo en la red adopta una dirección IP dentro del rango de las IP privadas.

1.1.4 Intelligent Electronic Device (IED)

Son equipos industriales que operan con las plantas directamente y pueden ser conectados a la red industrial. Presentan dos características fundamentales:

Son equipos microprocesados, lo que significa que su control es de tipo digital y la otra característica es que tiene una tarjeta de comunicaciones que le permite la recepción o el envío de datos involucrados en el procesamiento.

De este modo el IED tiene la facultad de ejecutar acciones de forma autónoma o porque ha recibido la orden desde un equipo remoto. También tiene la posibilidad de muestrear variables involucradas en el proceso eléctrico y enviarlas por su interface de comunicación hacia la red.

Los IED que implementan la norma IEC 61850 tienen la posibilidad de comunicarse entre sí, sin importar quien los haya fabricado. Esto es gracias a que se estandariza la composición de los mensajes que puedan transmitirse por la red de la subestación. Para que esto sea posible el IED presenta un perfil virtual basado en objetos y descrito mediante SCL. En el capítulo 2 se profundiza la estructura de este perfil virtual.

Los IED tienen terminales para ser conectados a dispositivos del nivel de proceso como disyuntores o sensores de voltaje y corriente. Lo anterior le da la capacidad al IED de abrir o cerrar el circuito donde fue instalado el disyuntor o medir el voltaje o corriente de la línea a la que se conectaron a los sensores.

1.1.5 SCL

Es el lenguaje de configuración de subestación. Permite la descripción completa de la red IEC 61850 mediante scripts de tipo XML. Se puede escribir en varias partes: Encabezado, Subestación, Comunicación, IED y DataTypeTemplates.

Asimismo SCL tiene varios tipos de fichero según la información que contiene, se identifican mediante la extensión del archivo que generan y la diferencia radica en las partes de un archivo SCL descritas anteriormente. A continuación se presentan los tipos de fichero SCL con una breve explicación de su contenido.

(A) SSD

Este fichero contiene todas las partes de una descripción con SCL, estas son, la parte de Comunicaciones, subestación, IED, DataTypeTemplates y el encabezado.

(B) ICD

En este fichero contiene la parte de IED de un script SCL aunque para la definición de sus LD, LN, DO y DA se puede usar la parte de DataTypeTemplates.

(C) SCD

Contiene la parte de la subestación, correspondiente al diagrama unifilar de la subestación a describir mediante el uso de la etiqueta Substation.

SCL es un lenguaje importante a lo largo del desarrollo del proyecto razón por la que en el anexo se muestra con mayor detalle.

1.2. OPEN IEC 61850

Es un proyecto abierto licenciado con LGPL y es una implementación de MMS de IEC 61850. Consta de una librería escrita en JAVA que contienen objetos y servicios ACSI apta tanto para la constitución de servidores y clientes.

OPEN IEC 61850 recoge casi todos los servicios ACSI y permite modelar mediante objetos de JAVA un perfil virtual de IED. Se logra mediante clases que involucran los niveles jerárquicos que un IED posee según lo propuesto por la norma. Tiene algunas funciones interesantes como la lectura de archivos SCL para la transición entre un script .ICD y un objeto de JAVA, sockets para la conexión cliente-servidor, lectura y escritura de todos los tipos de datos que

presenta la norma y mapeo completo del modelo de un servidor al ser recibido por el cliente.
Contiene además el código fuente de un servidor o IED virtual primitivo que permite las ejecutar las funciones anteriores o ser modificado para fines de desarrollo.
Finalmente es desarrollado por ISE Fraunhofer un instituto de investigación alemán apoyado por el ministerio de economía y tecnología del mismo país.
En el capítulo 2 se hará una presentación completa del contenido del proyecto abierto así como su correspondencia con la norma IEC 61850.

1.3 ANDROID

Es un sistema operativo dedicado a dispositivos móviles inteligentes con pantalla táctil. El proyecto Android es de código abierto y uno de los más usados a lo largo del mundo. Está basado en el kérnel de Linux y las aplicaciones que corren sobre el sistema operativo se soportan sobre una máquina virtual llamada Dalvik aunque dichas aplicaciones sean escritas en JAVA.
Al ser un proyecto abierto, el desarrollo de aplicaciones para éste sistema operativo es gratuito. Incluso Google, el patrocinador de Android, cuenta con una página web para desarrolladores de aplicaciones del S.O.
El sistema operativo se actualiza periódicamente añadiendo mejoras, aplicaciones, etc. Cada versión de Android cuenta con su respectiva API para el desarrollo de aplicaciones para determinada versión. Las API permiten al desarrollador el control de los periféricos de entrada y salida del dispositivo móvil como la pantalla táctil, el acelerómetro, WI-FI, Bluetooth, la tarjeta de sonido, la cámara o el micrófono así como también está compuesta de clases que permiten el control de flujo de una aplicación, ejecución de cálculos o creación de GUI para crear interacción con el usuario. Se puede identificar por su distintivo logo que aparece en la figura 5.

Figura 5. Logo del S.O Android

1.3.1. Desarrollo de aplicaciones

El proyecto Android recomienda el uso del framework de programación Eclipse. Es necesario un plugin para el software llamado ADT que añade al programa un apartado para la creación, depuración y ejecución de proyectos de aplicaciones Android en dispositivos emulados (mediante el complemento AVD) y reales.
El lenguaje de programación de desarrollo es JAVA, aunque se pueden ejecutar funciones de clases escritas en otros lenguajes mediante el uso de JNI (Java

Native Interface). Android permite el uso de objetos básicos de JAVA como tipos de datos, sockets, buffers, arrays, hilos, etc. y sus métodos respectivos. No es posible usar paquetes para GUI como AWT o Swing ya que la interacción con el usuario es diferente. El S.O tiene sus propios paquetes para la creación de interfaces de usuario permitiendo incluir UIElements (Elementos de Interfaz de usuario) en la pantalla usándolos como herramientas para la interacción con el usuario.

1.3.2 Funcionamiento y características

Dadas las limitaciones Hardware el S.O necesita de la liberación continúa de memoria para desechar procesos que la ocupan y tener la posibilidad de ejecutar nuevos. La liberación de memoria elimina procesos del sistema operativo y se hace con una prioridad definida:

1. Proceso vacío. Tiene caché reservado para la próxima ejecución de la aplicación que lo reservó.
2. Proceso en segundo plano. En desuso, minimizado.
3. Servicios. Procesos como control en segundo plano del GPS, WI-FI, Reproductor de música y demás servicios.
4. Proceso Visible. Una aplicación en uso que tenga dos procesos, uno de ellos estará en primer plano y otro pausado en segundo plano. Proceso visible hace referencia al último.
5. Proceso en primer plano. Está en uso y además ocupa la pantalla y el foco del dispositivo.

Una aplicación Android debe tener un proceso principal necesariamente con GUI para su interacción con el usuario. Un proceso así recibe el nombre de Actividad (Activity). Desde ella se pueden activar demás procesos sin interfaz, como cálculos.

Las actividades Android cuentan un ciclo de vida dentro del sistema operativo que el desarrollador debe tener en cuenta para tener control del estado de la aplicación principalmente con el objetivo de evitar su cierre imprevisto:

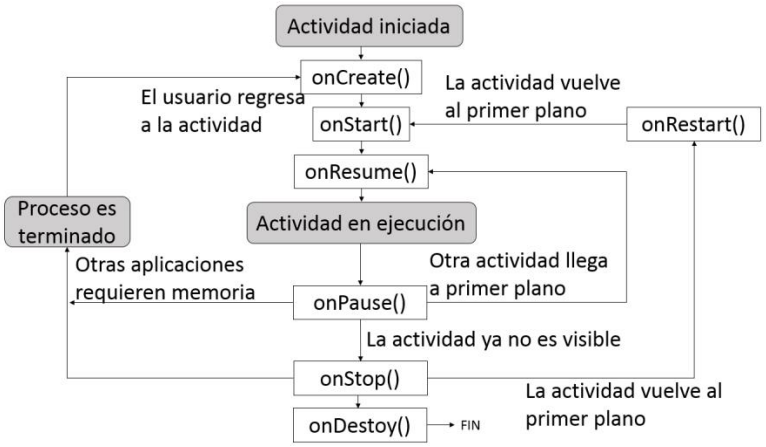


Figura 6. Ciclo de vida de una aplicación Android

Según la figura 6 al lanzar una aplicación se inicializan e instancian todos los elementos que componen la actividad principal (onCreate()), se informa que la aplicación está próxima a ser ejecutada (onStart()), luego es mostrada en pantalla (onResume()).
Una aplicación puede tener varias actividades y si otra es lanzada las dos pueden ser visibles para el usuario pero solo la que es lanzada recientemente es la que tendrá el foco, la otra estará pausada (onPause()). En este estado se puede regresar a estar visible y con foco (onResume()).
Si una actividad no tiene foco y no es visible entrará en estado de parada (onStop()). La actividad en onStop() puede ser destruida (onDestroy()) o traer de vuelta a la pantalla lo que implica reiniciar el proceso (onRestart()).
Si el sistema operativo requiere memoria, según la prioridad de procesos en Android, el estado más vulnerable es onStop() (Proceso en segundo plano), seguido de onPause() (Proceso Visible).
Un proyecto de Android puede contener varias actividades y varias clases. Cada actividad tiene asociado un archivo de apariencia escrito en XML.
Los elementos gráficos se especifican con etiquetas XML con las que se pueden incluir botones, cajas de texto, cajas de introducción de texto, etc. La configuración de la aplicación también se logra con un archivo XML llamada AndroidManifest.xml. A continuación se mencionan los componentes de un proyecto Android tal como se organiza en la carpeta del proyecto
Res – Es la carpeta de recursos de la aplicación y dentro se encuentra lo necesario para formar la apariencia de una aplicación. Dentro se encuentran las imágenes, layouts (Apariencia de una activity) y valores (Cadenas de texto y colores) que son los recursos de apariencia para ser usados en la aplicación.
Bin- Esta carpeta contiene el archivo .apk se es creado al compilar el proyecto y que los dispositivos Android pueden leer como instaladores de aplicación.
Src – Contiene las clases y actividades escritas en JAVA que rigen el funcionamiento de la aplicación.
Gen – Contiene un archivo llamado R.java el cual mapea todos los elementos definidos en la carpeta Layout y Drawable como identificadores de tipo entero para usarlos en las clases o actividades de la carpeta Src.
Libs – Almacena las librerías para poder ser usadas en el proyecto de aplicación en las clases o actividades de la carpeta Src. Las librerías deben ser guardas con extensión .jar.
AndroidManifest.xml – Define la estructura de los componentes del proyecto mediante la descripción de características como el nombre, la versión de API de Android, la versión mínima permitida para su ejecución, el tema visual, el icono, etc.

Además este archivo le concede de forma explícita a la aplicación los permisos de uso de los periféricos o de información privilegiada del dispositivo móvil.

1.3.3 Apariencia e interacción con el usuario

(A) Layout

Existen diferentes formas de organizar UIElements (Elementos gráficos de Android) dentro de la pantalla del dispositivo. La organización depende del layout utilizado. Principalmente existen dos formas de organización de elementos, estas son LinearLayout y RelativeLayout (véase figura 7). La primera organiza los elementos vertical u horizontalmente en fila, la segunda deja libre la ubicación de cada elemento según las coordenadas en la pantalla.

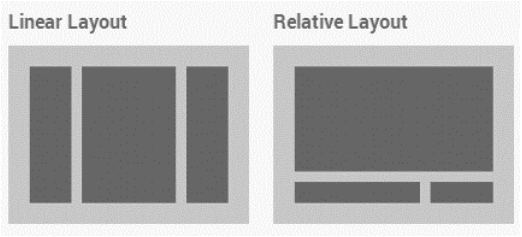


Figura 7. Organización de UIElements en Linear y Relative Layout

(B) UIElements de entrada

Button – Un botón virtual que sirve para activar una función especifica
EditTetx – Un cuadro de texto que permite el ingreso de datos.
Switch – Un interruptor virtual con dos posibles estados: Encendido o Apagado.

(C) UIElements de salida

TextBox – Un cuadro de texto que puede mostrar texto al usuario.
Toast – Un mensaje emergente de información al usuario mostrado temporalmente.

(D) Listeners

Se denomina *Listener* a la interrupción causada por el suceso de un evento como el toque de la pantalla. Cada UIElement presenta listeners para que ejecuten acciones luego de sucedido el evento.

2. OPEN IEC 61850

2.1. MODELO VIRTUAL DE UN IED

En la norma se modelan virtualmente los IED con objetos, no sujetos a algún lenguaje de programación especifico pero dada su naturaleza de objetos, se establece un orden jerárquico que sitúa a cada objeto dentro de otro más general.
El primer objeto corresponde al Logical Device (LD) y está asociado directamente al dispositivo físico (PD) que lo contiene. Cada LD contiene un conjunto de objetos llamados Logical Nodes (LN) los que a su vez contienen objetos de datos (Data Objects, DO). Los DO contienen Atributos de datos (Data Attributes, DA) que corresponden a valores de algún tipo de dato especificado por la norma ANS.1 aunque de ser necesario cada DA pueden ser contenedores de variables.
Cabe anotar que cada DA tiene asociado una etiqueta denominada Functional Constraint (FC) que clasifica la función de cada DA, así por ejemplo, gracias al FC se puede separar los DA de configuración, los DA de descripción o los DA de medición.
Open IEC 61850 usa el estándar e implementa los objetos descritos en ella con el lenguaje de programación JAVA. La norma y su implementación difieren en que la etiqueta FC cubre desde DO. En la figura 8 se muestran los organigramas mostrando la diferencia y el orden de los miembros en un modelo virtual de IED.

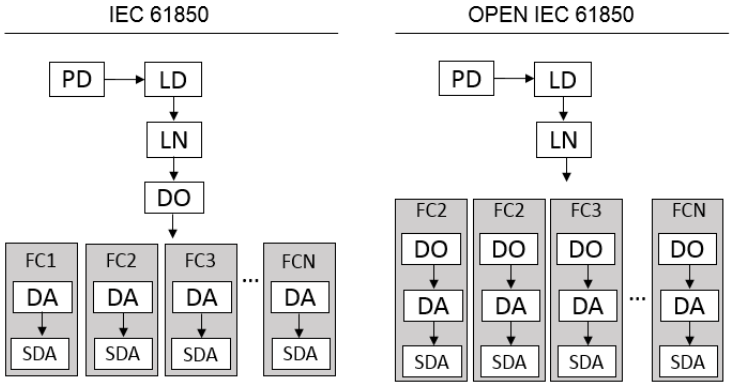


Figura 8. Modelo genérico de IED según la norma y su implementación

La diferencia no afecta el orden del modelo y es totalmente compatible. Esto se logra mediante varios DO con un mismo nombre dentro de un LN. Aunque su nombre sea igual tienen FC distintos lo que también puede verse como un solo DO con distintos FC como en el modelo original de la norma.
OPEN IEC 61850 denomina 'Nodo' a cualquier miembro del mapa sea del nivel jerárquico que sea. No se debe confundir con nodo lógico o LN que corresponde al tercer nivel del modelo luego de PD y LD.
El siguiente numeral describe las clases usadas en OPEN IEC 61850 para el modelamiento del modelo genérico de IED y sus objetos.

2.1.1 Modelamiento de objetos IEC 61850 con OPEN IEC 61850

A continuación se describen los distintos objetos que compone el proyecto OPEN IEC 61850 haciendo un paralelo con los descritos en la norma mostrando las correspondencias entre la norma y su implementación

2.1.1.1. Modelo Genérico de IED y clases de propósito general

Son clases generales que tienen que ver con otras más específicas en torno a la descripción de nodos del modelo genérico de IED.

(A) ServerModel

Esta es la clase con la que se modela de forma general un servidor IEC 61850 o IED. La composición jerárquica de este objeto corresponde al modelo genérico de servidor IEC 61850.

Se puede acceder a los *Dataset* (getDataSets()), o a los Bloques de Control de Reportes sin Buffer (getUrcbs()) configurados en el IED.

Esta clase también permite el acceso completo a todos los miembros de la estructura mediante la función findModelNode(ObjectReference ref, Fc fc).

Ref contiene la ruta o referencia y Fc describe el *functionalConstraint* del nodo. Si el nodo carece de Fc como es el caso de los LD y LN según lo muestra la figura 8 se puede poner un valor nulo. El método devuelve objetos de la clase *ModelNode*.

(B) ModelNode

Se trata de una clase genérica con la que se puede modelar todos los miembros del modelo jerárquico desde LD hasta los DA e inferiores.

Entre sus métodos se encuentran el acceso de sus objetos hijos y padres con la posibilidad de instanciar cada uno. Lo anterior sugiere que se pueden establecer relaciones de herencia entre todos los integrantes del modelo virtual de IED. La figura 9 ilustra cómo se puede acceder al nodo padre y a los nodos hijos de un nodo A dado ubicado en cualquier nivel jerárquico.

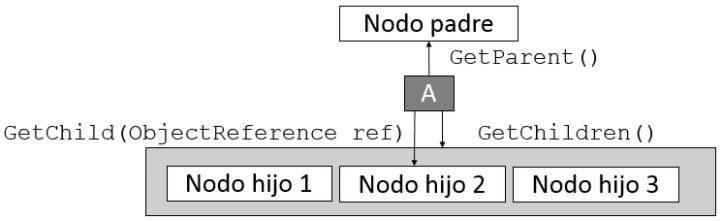


Figura 9. Acceso a nodos emparentados directamente

GetParent() devuelve un objeto de tipo ModelNode. *getChildren()* por su parte devolverá una colección JAVA (java.util.Collection) de objetos *ModelNode* con todos los nodos hijos de A. Finalmente el método *getChild(ObjectReference ref)* devuelve también un objeto de la clase ModelNode y se deberá conocer su referencia *ref* para poder acceder a él.

La referencia a cualquier nodo se representa con un objeto de la clase *ObjectReference* del cual se hablará más adelante. Como modelo de nodo, es

importante conocer su ruta dentro del modelo de árbol. ModelNode presenta el método *getReference()* que devuelve un objeto ObjectReference que contendrá su ruta. Si se requiere saber el nombre del nodo se puede hacer mediante el método *getName().*

Las siguientes clases extienden a ModelNode, por lo cual van a heredar sus métodos y atributos. La existencia de estas clases permite la instanciación de cada uno de los miembros por nivel del modelo genérico de IED proveniente de un archivo SCL o de la conexión con un IED mapeando cada etiqueta SCL en objetos JAVA.

(C) FcModelNode

Esta clase puede modelar miembros del esquema que tengan FC es decir del nivel de los DO, DA y subsiguientes si existen objetos pertenecientes a niveles más abajo de la jerarquía. Se diferencia a la anterior clase porque cada nodo no solo tiene un nombre y una referencia sino que ahora tiene un FC que se representa con un objeto de la clase Fc y de la cual se describirá más adelante.

(D) ObjectReference

Esta clase permite al usuario apuntar a un nodo determinado dentro del modelo genérico de IED. Todas las clases anteriores permiten ser cuestionadas sobre su referencia (getReference()) o sobre un nodo relacionado como su objeto padre o sus objetos hijos.

Los métodos de la clase permiten conocer el nombre del nodo al que referencian (getName()) y su ruta (toString()) pero se usa el mismo objeto de la clase como parámetro de entrada para apuntar al nodo referenciado en dicho objeto.

Este objeto consta de una cadena de texto con la dirección URI del nodo al que apunta la referencia y tiene la siguiente estructura:

NombreLD/NombreLN.NombreDO [FC].NombreDA.NombreSDA. … .NombreBDA

2.1.1.2. Clases correspondientes al modelo genérico de IED

Las siguientes son las clases que corresponden a los niveles jerárquicos de la estructura de un IED.

(A) LogicalDevice

El Logical Device (LD) corresponde a un *Domain* de la estructura MMS. Contiene todos los nodos lógicos configurados en el IED. Como *Domain* es el recurso principal del VDM conteniendo las funciones configuradas en el IED. Es factible incluir varios LD en un mismo IED gracias al lenguaje de configuración de IEC 61850 (SCL).

En Open IEC 61850 se implementa con la clase *LogicalDevice*. Para instanciar un LD se debe usar la referencia y la lista de LN que contiene.

(B) LogicalNode

Los nodos lógicos corresponden a aplicaciones reales. Cada nodo lógico contiene una función en específico y la unión de todas las funciones componen la función total del LD.

Hay distintos tipos de nodos lógicos, algunos reservados a la descripción, otros a la protección y otros a la medición. El nombre que recibe cada uno va de acuerdo a una nomenclatura que puede verse en la tabla 4.

Letra	Asociación
L	Sistema
P	Protección
R	Relacionado con la protección
C	Control
G	Genérico
I	Interfaces y archivado
A	Control Automático
M	Medición
S	Sensores y Monitoreo
X	Conmutación
T	Transformadores de instrumentación
Y	Transformadores de potencia
Z	Equipos de potencia

Tabla 4. Nomenclatura del nombre de los nodos lógicos

Las letras de tabla se usan en la primera letra del nombre del nodo para clasificarlo, las siguientes letras son algunas referencias de su función específica. Algunos ejemplos del uso de la nomenclatura en los nodos lógicos en las CDC pueden observarse en la tabla 5.

Nombre LN	Función
PDIF	Protección diferencial
RBRF	Breaker contra fallas
XCBR	Breaker
CSWI	Switch de control
PDIS	Protección a distancia
TVTR	Transformador de voltaje
TCTR	Transformador de corriente
MMXU	Unidad de medición
YPTR	Transformador de potencia
XSWI	Switch de conmutación

Tabla 5. Ejemplos de nombres de nodos lógicos

En el caso del segundo nodo por ejemplo, RBRF, la R corresponde a una función relacionada con la protección las siguientes letras significan **BR**eaker **F**ailure. Asimismo el séptimo, TCTR relaciona el nodo con un Transformador según la tabla, las siguientes letras significan **C**urrent **TR**ansformer.

En Open IEC 61850 se implementa con la clase *LogicalNodes*. Permite la instanciación de un LN usando la referencia y la lista de DO que tomará como sus hijos.
Permite también el acceso a los bloques de control de reporte que se le hayan configurado al nodo en el archivo .ICD.

(C) Fc

FunctionalConstraint es la clasificación funcional de un DA. Los clasifica en grupos de variables según su utilidad. Por ejemplo existas DA de bloqueo, de configuración, de medida o para la descripción.
Los FC descritos en la implementación son los de la tabla 6:

FUNCTIONAL CONSTRAINT					
BL	Blocking	**BR**	Buffered Reporting	**SG**	Setting group
CF	Configuration	**CO**	Control	**DC**	Description
OR	Operate received	**EX**	Extended definition	**SV**	Substitution
MX	Measurands, Analogue Values	**RP**	Unbuffered Reporting	**SE**	Setting group editable
SP	Setpoint	**SR**	Service response	**ST**	Status Information

Tabla 6. FunctionalConstraints descritos en OPEN IEC 61850

En OPEN IEC 61850 se implementa con la clase Fc. Esta clase tiene una enumeración de todos los FC.
En todos los nodos que tengan que ver con el FC, es decir desde DO hacia abajo en la jerarquía, se requiere esta clase. Se trata del Functional Constraint de la norma y presenta la misma nomenclatura de dos letras mayúsculas que esta.
Hacer una declaración e instancia de una variable tipo FC se hace de la siguiente forma:

```
Fc fcEjemplo;
fcEjemplo = Fc.MX;
```

La clase FcModelNode y por tanto sus subclases tienen un método llamado getFC() que devuelve un objeto de esta clase con información del FC del nodo cuestionado.
Los nodos que son niveles inferiores a LN tienen este parámetro y permite la clasificación por función de los DO. Recuérdese que sólo en OPEN IEC 61850 el Fc clasifica funcionalmente a los nodos desde DO. En la norma es desde DA hacia abajo

(D) FcDataObjects (DO)

Son las propiedades de los nodos lógicos. Algunos pueden referirse al nombre, otros al valor o al estado.
En Open IEC 61850 se implementa con FcDataObject. OPEN IEC 61850 lo adopta como un nodo con FC aunque en la norma no aparezca de esa forma como se vio en la figura 8. No obstante cuando se obtiene un modelo de IED desde un .ICD o

desde la conexión misma el objeto ServerModel tendrá tantos DO como el número de FC en total de todos los DA que contiene, como se ve en la figura 10.

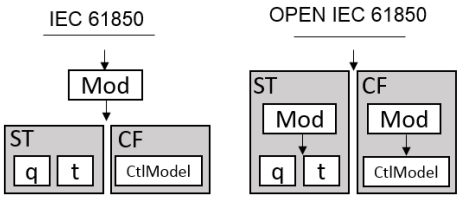


Figura 10. Diferencia entre el nivel DO de IEC 61850 y de OPEN IEC 61850

Lo anterior significa que habrán DO con nombre repetido pero distinto FC.
Para instanciar el DO se debe ingresar la referencia (ObjectReference), el FC (FC) y la lista de FcModelNode que contiene. Los objetos de esta lista deben estar en un nivel jerárquico inferior y corresponden al nivel de DA.

(E) DataAttributes (DA)

Como su nombre lo indica, describe los atributos de cada propiedad del nodo lógico. Se usan para la creación de variables presentes en el IED como variables de control o variables de medida cuyo valor representa el resultado de una medición en curso.
Los Data Attributes pueden ser de tres tipos mapeadas en las siguientes clases:

(F) BasicDataAttributes (BDA)

Representa una variable del modelo MMS con un valor y tipo de dato asociado.
En OPEN IEC 61850 se implementa con la clase BasicDataAttributes. Contiene subclases que dan soporte a todos los tipos de datos de la norma IEC 61850 de los cuales se hablará más adelante.
Sus métodos permiten obtener los parámetros configurados en el archivo SCL como el dchg, dupd, qchg (disparos para el sistema de reportes) y también su valor actual. Sin embargo cada una de sus subclases presenta métodos específicos para desempeñar esto último según el tipo de dato.

(G) ConstructedDataAttributes (Struct)

Es un contenedor que almacena varios DA. Por esta razón dan lugar a un nivel más abajo en la jerarquía para formar los SDA.
Los DA de tipo contenedor se instancian con la clase de OPEN IEC 61850 Constructed Data Attributes.

(H) Vector de BDA

Un *DataAttribute* es un arreglo de BDA del mismo tipo de dato. En OPEN IEC 61850 se implementa con una clase Homónima**.**

2.2 TIPOS DE DATOS

Todos los tipos de datos descritos en la norma se encuentran en OPEN IEC 61850 son subclases de la clase *BasicDataAttributes*.

2.2.1 Tipos de datos de propósito general

Las siguientes clases son estructuras de datos que sirven para la creación de otros tipos de datos presentes en la norma o para identificarlos.

(A) BdaType

Es una enumeración de todos los tipos de datos que puede presentar un dato. Son dos grupos: Tipos Básicos (Para variables dentro del modelo genérico):
Boolean, Check, DoublebitPos, Float32, Float64, Int16, Int16U, Int32, Int32U, Int64, Int8, Int8U, OctectString, Quality, Timestamp, UnicodeString, VisibleString.
Y tipos comunes ACSI (Para servicios del IED):
OptFlds, ReasonForInclusion, TapCommand, TriggerConditons.
Es útil para la identificación de tipos de variables que existen en un ServerModel. Todos los BasicDataAttributes tienen el método getBasicType() que devuelve un elemento de esta enumeración el cual tiene la información del tipo de BDA que es el objeto al que se cuestiona.

(B) BdaBitString

Este tipo de dato se basa en una sucesión de bits expresable con datos de tipo byte[] en JAVA. Por ejemplo un bitString llamado A puede ser:

```
A = {1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1}
```

BitString no tiene una extensión fija y es el tipo de dato base para sus subclases: *BdaCheck, BdaDoublebitPos, BdaOptFlds, BdaQuality, BdaReasonForInclusion, BdaTapCommand, BdaTriggerConditions* de las cuales se hablara más adelante.

2.2.2 Tipos básicos de BDA

A continuación se describen las clases con las que se implementas los BDA de la norma IEC 61850 o Atributos de datos básicos:

(A) BdaBoolean

Este tipo de dato puede tener solo dos valores: true y false. El equivalente en java es el tipo de dato *boolean*.

(B) BdaCheck

Este tipo de dato extiende a bitString y tiene como carga útil dos bits correspondientes a dos atributos booleanos:

Atributo	Escritura	Lectura
InterLockCheck	setInterlockCheck(boolean a)	getInterLockCheck().
SynchroCheck	setSynchroCheck(Boolean b)	getSynchroCheck().

Tabla 7. Atributos de BdaCheck y sus métodos de escritura y lectura

(C) BdaDoubleBitPos

Este tipo extiende a BitString y la carga útil se encuentra en dos bits. Cada combinación de bits representa un valor referente al estado y posición de un interruptor de un IED.
Las cuatro posibles constantes están nombradas y son BAD_STATE, INTERMEDIATE_STATE, OFF y ON.

En java, los valores se encuentran definidos como datos estáticos en la subclase *BdaDoubleBitPos.DoubleBitPos* Así para definir un BdaDoubleBitPos se hace lo siguiente:

```
BdaDoubleBit ejemplo = DoubleBitPos.ON;
```

Tiene adicionalmente un método para escribirle un valor por defecto (setDefault()) de OFF.

(D) BdaQuality

La carga útil de este tipo de dato es de doce bits. Los dos primeros corresponden a constantes creadas especialmente para BdaQuality y es el atributo Validity. Puede tener cuatro valores nombrados en variables estáticas: GOOD, INVALID, QUESTIONABLE y RESERVED. Para hacer uso de estas constantes se debe usar la extensión Validity de la clase definiéndola de la siguiente forma:

```
Validity validez = Validity.GOOD;
```

Los siguientes diez atributos son booleanos y junto con la validez se presentan en la tabla 8.

Atributo	Escritura	Lectura
Validity	setValidity(BdaQuality.Validity a)	getValidity()
BadReference	setBadReference(boolean a)	getBadReference()
Failure	setFailure(boolean a)	getFailure()
Innacurate	setInnacurate(boolean a)	getInnacurate()
Inconsistent	setInconsistent(boolean a)	getInconsistent()
OldData	setOldData(boolean a)	getOldData()
OperatorBlocked	setOperatorBlocked(boolean a)	getOperatorBlocked()
Oscillatory	setOscillatory(boolean a)	getOscillatory()
OutOfRange	setOutOfRange(boolean a)	getOutOfRange()
Overflow	setOverflow(boolean a)	getOverflow()
Substituted	setSubstituted(boolean a)	getSubstituted()

Tabla 8. Atributos de BdaQuality y sus métodos de escritura y lectura

(E) BdaOctetString

Tipo de datos que representa una cadena de bytes. Cada byte tiene un valor decimal equivalente de 0 a 255. Un ejemplo de este tipo de dato se puede ver a continuación.

```
A = {0, 255, 177, 76, 91, 43, …}
```

El equivalente java es el tipo byte[]. La lectura de este tipo de dato se hace mediante el método getValue() que devuelve un byte. La escritura es gracias al método setValue(byte[] a).

La norma específica la longitud de este tipo de dato en 64 bytes.

(F) BdaTimestamp

Es la variable de tiempo de la norma IEC 61850. Se puede escribir en ella una fecha y hora determinada. La clase java.util.Date es la acorde para esto. La clase ofrece la posibilidad de ponerle a la variable la fecha actual (la fecha de cuando se

usa setCurrentTime()) o una fecha cualquiera que este descrita mediante java.util.Date (setDate(Date a)).

Sin embargo este no es el único atributo del que se compone este tipo de dato. Contiene además tres atributos booleanos:

Atributo	Lectura
clockFailure	getclockFailure()
clockNotSynchronized	getclockNotSynchronized()
LeapSecondsKnown	getLeapSecondsKnown()

Tabla 9. Escritura y lectura de Atributos booleanos de BdaTimestamp

Los atributos anteriores son de solo lectura al igual que los tres siguientes parámetros numéricos de este tipo de dato (Tabla 10).

Atributo	Lectura	Tipo
FractionOfSeconds	getFractionOfSecond()	Int
TimeAccuracy	getTimeAccuracy()	int
secondsSinceEpoch	getSecondsSinceEpoch()	long

Tabla 10. Escritura y lectura de atributos numéricos de BdaQuality

(G) TIPOS DE DATOS NUMERICOS

La norma presenta varios tipos de representación numérica y algunos de los tipos de datos concuerdan con tipos de datos de java. La tabla 11 muestra los tipos de datos y su rango.

TIPO DE DATO OPEN IEC 61850	RANGO	Descripción
BdaFloat32	[-3.4e38, -1.4e-45] U[1.4e-45, 3.4e38]	Numero en punto flotante de 32 bit que coincide con el tipo de java float**.**
BdaFloat64	[-1.78e308, -4.94e-324] U [4.94e-324, 1.78e308]	Numero en punto flotante de 64 bit que coincide con el tipo de java double**.**
BdaInt16	[0,65535]	Entero de 16-bit sin signo
BdaInt16U	[-32767,32767]	Entero de 16 bit con signo
BdaInt32	[0, 4294967295]	Entero de 32 bit sin signo
BdaInt32U	[-2147483648, 2147483648]	Entero de 32 bit con signo
BdaInt64	[0, 1.845 e 19]	Entero de 64 bit sin signo
BdaInt8	[0, 255]	Entero de 8 bit sin signo
BdaInt8U	[-127, 127]	Entero de 8 bit con signo

Tabla 11. Tipos de datos numéricos de OPEN IEC 61850

Para la lectura de las variables numéricas es necesario definir variables JAVA teniendo en cuenta su equivalente en JAVA.

TIPO DE DATO OPEN IEC 61850	Método de lectura general de tipo byte[]	Método de lectura especifico	Tipo de dato java
BdaFloat32	getValue()	getFloat()	float
BdaFloat64	getValue()	getDouble()	double
BdaInt16		getValue()	short
BdaInt16U		getValue()	int
BdaInt32		getValue()	int
BdaInt32U		getValue()	long
BdaInt64		getValue()	long
BdaInt8		getValue()	byte
BdaInt8U		getValue()	short

Tabla 12. Métodos de lectura para tipos de datos numéricos

Los tipos de datos BdaFloat32, BdaFloat64 son los únicos que se pueden leer en formato de un arreglo de bytes, pero también tienen la opción de leerse directamente como tipos de datos primitivos de java.
Por otra parte para definir y modificar el valor de las variables de tipo numérico se usan los métodos de la tabla 13 que concuerdan con la equivalencia java de una variable IEC 61850.

TIPO DE DATO OPEN IEC 61850	Método de escritura general de tipo byte[]	Método de escritura especifico
BdaFloat32	setValue(byte[] a)	setFloat(float b)
BdaFloat64	setValue(byte[] a)	setDouble(double b)
BdaInt16		getValue(short b)
BdaInt16U		getValue(int b)
BdaInt32		getValue(int b)
BdaInt32U		getValue(long b)
BdaInt64		getValue(long b)
BdaInt8		getValue(byte b)
BdaInt8U		getValue(short b)

Tabla 13. Métodos de escritura para tipos de datos numéricos

Los BdaFloat32 y BdaFloat64 tienen dos métodos de escritura. El especifico con tipos de variables primitivas de java y los generales donde han de describirse mediante cadenas de bytes.

(H) BdaVisibleString

Equivalente a un String de java. Es útil para descripciones como el fabricante, la versión y otros parámetros de texto que describan el IED.
La norma IEC 61850 fija la longitud de este tipo básico de dato en 32, 64 y 255 caracteres.

(I) BdaUnicodeString

Equivalente a un String de java en formato Unicode. Aunque se puedan declarar UnicodeString de distinta longitud, la norma IEC 61850 fija la longitud de este tipo de dato en 255 caracteres.

Es útil para descripciones con letras en otros idiomas, firmas digitales, etc.

2.2.3 Tipos de datos para servicios ACSI

Estos tipos de datos son usados para servicios ACSI en variables involucradas en el control de reportes y corresponden a las siguientes clases:

(A) BdaReasonForInclusion

La carga útil de este tipo es de dato es seis bits y cada uno representa los siguientes atributos.

Atributo	Escritura	Lectura
ApplicationTrigger	setApplicationTrigger(boolean a)	isApplicationTrigger()
DataChange	setDataChange(boolean a)	isDataChange()
DataUpdate	setDataUpdate(boolean a)	isDataUpdate()
GeneralInterrogation	setGeneralInterrogation(boolean a)	isGeneralInterrogation()
Integrity	setIntegrity(boolean a)	isIntegrity()
QualityChange	setQualityChange(boolean a)	isQualityChange()

Tabla 14. Escritura y lectura de los atributos de BdaReasonForInclusion

(B) BdaTapCommand

Este tipo extiende a BitString y la carga útil se encuentra en dos bits. Cada combinación de bits representa un valor constante. Los valores posibles para una variable TapCommand son cuatro según los dos bit reservados para este campo, estos son: HIGHER, LOWER, RESERVED, STOP.

En java, los valores se encuentran definidos como constantes en la subclase BdaTapCommand**.**TapCommand. Así para definir un TapCommand se hace lo siguiente:

```
TapCommand ejemplo = TapCommand.LOWER;
```

Tiene adicionalmente un método para escribirle un valor por defecto (setDefault()) de STOP.

(C) BdaTriggerConditions

La carga útil de este tipo de dato es de cinco bits. Cada bit corresponde a uno de los atributos que son presentados en la tabla 15

Atributo	Escritura	Lectura
DataChange	setDataChange(boolean a)	isDataChange()
DataUpdate	setDataUpdate(boolean a)	isDataUpdate()
GeneralInterrogation	setGeneralInterrogation(boolean a)	isGeneralInterrogation()
Integrity	setIntegrity(boolean a)	isIntegrity()
QualityChange	setQualityChange(boolean a)	isQualityChange()

Tabla 15. Escritura y lectura de los atributos de BdaTriggerConditions

(D) BdaOptFlds

La carga útil de este tipo de dato es de ocho bits. Cada bit representa un atributo y todos son presentados en la tabla 16 con sus métodos de lectura y escritura

Atributo	Escritura	Lectura
BufferedOverflow	setBufferedOverflow(boolean a)	isBufferedOverflow()
ConfigRevision	setConfigRevision(boolean a)	isConfigRevision()
DataReference	setDataReference(boolean a)	isDataReference()
EntryId	setEntryId(boolean a)	isEntryId()
ReasonForInclusion	setReasonForInclusion(boolean a)	isReasonForInclusion()
ReportTimestamp	setReportTimestamp(boolean a)	isReportTimestamp()
Segmentation	setSegmentation(boolean a)	isSegmentation()
SequenceNumber	setSequenceNumber(boolean a)	isSequenceNumber()

Tabla 16. Escritura y lectura de los atributos de BdaOptFlds

2.3. SERVICIOS IMPLEMENTADOS EN OPEN IEC 61850

En el capítulo pasado se describieron los objetos IEC 61850 que están dentro de la norma. En el presente capítulo se mirará con detalle las clases que componen los servicios IEC 61850 contenidos en las clases de la implementación OPEN IEC 61850.

2.3.1 Clases para la conexión

El proyecto OPEN IEC 61850 permite la creación tanto de clientes como servidores IEC 61850. Los grupos de clases son los siguientes:

2.3.1.1 Clientes

Las siguientes clases permiten la creación de objetos ACSI que soliciten servicios ACSI desde el lado cliente:

(A) ClientSap

Representa el servicio de acceso solo para los clientes IEC 61850. Para que un cliente pueda conectarse al servidor debe primero crear una instancia de esta clase, configurar el objeto y luego usar la clase de asociación *ClientAssociation*.

Entre los métodos de configuración del objeto se encuentra el poder escoger el máximo tamaño de PDU (setMaxMMSPDUsize()) que puede variar entre 64 y 65000 octetos y por defecto es 65000. Permite además la selección del tiempo de espera máximo de respuesta mediante la función setMessageFragmentTimeout(int timeout) que por defecto es de veinte segundos.

Se puede también configurar el tiempo de espera del mensaje fragmentado (setMessageFragmentTimeout()) que corresponde al tiempo que se debe esperar para recibir fragmentos de un mensaje una vez el primer byte es recibido.

Finalmente admite también la configuración del selector de transporte (T-SEL), en modo local y modo remoto.

(B) ClientAssociation

Esta clase complementa a la anterior ya que es la encargada del manejo de servicios desde el servidor hacia el cliente. Con este objeto se pueden ejecutar los servicios ACSI hasta ahora implementados en OPEN IEC 61850 y descritos en la clase *ServicesSupport* de la misma.

Para la conexión se requiere del objeto instanciado de la clase ClientSAP y de la clase ClientAsociation y se usa el método associate(InetAdress address, int port, String autenticacion) como se muestra a continuación.

```
ClientAssociation clienteAsociacionEjemplo;
clienteAsociacionEjemplo = ObjetoclientSAP.associate();
```

Para la desconexión del socket se debe usar el método disconect().

El parámetro de autenticación es una característica que OPEN IEC 61850 puede codificar pero no comprobar por el momento. Los IED con los que se use las funciones de la librería deben carecer de este parámetro para acceder a ellos.

Esta clase tiene implementados los servicios ACSI de IEC 61850 y permiten las operaciones de control y monitoreo, objetivos centrales del proyecto.

La tabla 17 presenta los servicios ACSI ejecutables por la norma y su método correspondiente de la clase ClientAssociation:

Servicio ACSI	Método de la clase ClientAssociation	Descripción
GetDataValues	getDataValues(FcModelNode FcNode)	Lectura de nodos
SetDataValues	setDataValues(FcModelNode FcNode)	Escritura de nodos
GetDataSetValues	getDataSetValues (DataSet dataSet)	Lectura de dataset
SetDataSetValues	setDataSetValues (DataSet dataSet)	Escritura de dataset
Select	select(FcModelNode FcNode)	Selección de nodo
Operate	operate(FcModelNode FcNode)	Operar nodo
CreateDataset	CreateDataset(List<FcModelNode>)	Crear dataset
DeleteDataset	DeleteDataset(List<FcModelNode>)	Eliminar dataset
GetURCBValues/ GetBRCBValues	GetRCBValues(Rcb Rcblock)	Leer los valores de un RCB
SetURCBValues/ SetBRCBValues/	SetRCBValues(Rcb Rcblock)	Escribir los valores de un RCB
GetDirectory/ GetDefinition	retreiveModel()	Retorna un todo el modelo del IED en un objeto serverModel

Tabla 17. Servicios ACSI presentes en OPEN IEC 61850

Operate y Select funcionan para seguridad SBO-with-normal-security (Select before Operate). Si se quiere controlar un nodo, se debe seleccionar primero antes de operarlo. SBO es creado en la norma para reservar un nodo para su control de modo que otros clientes no puedan modificarlo mientras se encuentra en operación.

(C) ServiceError

Cuando se intenta enviar un mensaje a un IED este puede fallar por distintas circunstancias. Las locales y de conexión con el host se pueden solventar con excepciones de JAVA como *IOException*. Pero las fallas producidas en el servidor se detectan con *ServiceError* que se lanza cuando el cliente intenta interactuar con él, pero este no contesta o no está operando en la red. Esto le permite al cliente darse cuenta del estado del servidor al que se intenta vincular.

2.3.1.2 Servidores

Las siguientes clases permiten la creación de objetos ACSI que brinden servicios ACSI desde el lado servidor:

(A) ServerSap

Es la clase análoga a ClientSap pero para la configuración y puesta en marcha del servidor. Tiene todas las opciones de configuración del ClientSap pero también tiene un método de gran importancia como lo es *getSapsFromSclFile*. Este método permite cargar un archivo .ICD en el servidor que operará al poner en marcha el servidor mediante el método *startListening()*.

El uso de *getSapsFromSclFile* puede o no tener éxito según el archivo .ICD. Si es correcto, el objeto de ServerSap con el que se ha usado el método en cuestión creará el servidor con el modelo descrito. Si es erróneo será lanzada la excepción *SclParseException* que detecta el problema consignándolo en consola.

El serverSap permite la emulación de un SAP de un IED y con él se pueden hacer pruebas en red con clientes IEC 61850 pero tiene una limitación: Los *SAP-IED* creados con el proyecto abierto no están habilitados para prestar servicios de reporte ya que esta parte sigue en proceso de desarrollo.

2.3.2 Report Control Block y Datasets

Es un servicio prestado por los IED que consiste en un sistema de difusión de mensajes desde el servidor hacia los clientes conectados de valores de variables determinadas cuando su estado o valor ha cambiado y guarda estos eventos en un log llamado reporte.

Para su funcionamiento la norma usa tres bloques además del dispositivo lógico (LD) (Contenido en objetos de la clase ServerModel). Estos son:

(A) Dataset

Un Dataset es una lista de accesos directos a DO, DA y SDA contenidas en el LD. Son declaradas en el archivo .ICD dentro del nodo lógico principal LN0. La información que se produzca en estos será la que se difunda entre los clientes.

También se pueden crear DataSets una vez el IED está en servicio mediante el servicio ACSI llamado CreateDataset. Para esto, el cliente debe informarle al IED cuál será el o los nodos FC que quiere agregar al nuevo Dataset.

Asimismo el cliente puede eliminar los Dataset borrables mediante el servicio ACSI DeleteDataset. Los Datasets borrables son los que un cliente ha creado en el servidor mientras los no borrables son los declaradas en el script SCL.

OPEN IEC 61850 implementa lo anterior con la clase *Dataset*.

(B) ReportControlBlock

El bloque de control de reportes toma un Dataset creado en el mismo script SCL para controlar su difusión y almacenamiento temporal de entradas al reporte. Los RCB generan un reporte con un ID especificado con la cadena de texto *RptID*, un valor booleano para conocer si el reportaje de eventos está activado (*RptEna*).

Presenta otras variables para controlar el flujo de reportes para los clientes como el número de secuencia en el caso de existan más reportes en cola para determinar el orden en el que emite cada uno o el tamaño de buffer que limita la memoria del Report.

OPEN IEC 61850 implementa lo anterior con la clase RCB y dos subclases URCB y BRCB que difieren en que el primero no tiene buffer mientras el segundo sí.

(C) Report

Contiene las entradas de cada evento en las variables en observación. Cualquier nodo que se referencia en el Dataset debe tener uno o un conjunto de BDA que obtengan la información de los sensores. Cada BDA tiene tres eventos que ocasionan una nueva entrada en el reporte, estas son:

dchg - Cambio en los datos (Data Change)

qchg - Cambio en la calidad (Quality Change)

dupd - Actualización de datos (Data Update)

Los anteriores eventos se pueden asociar a los BDA en el script SCL usando los parámetros homónimos durante la creación del script SCL.

3. METODOLOGÍA

En este capítulo se aborda el desarrollo del trabajo de grado mencionando inicialmente las herramientas hardware y software usadas para el desarrollo de la aplicación. Seguidamente se describen las características de la aplicación de telecontrol móvil. Luego se hace una mención informativa de las clases que se usan en Android para realizar la aplicación tanto visual como funcionalmente. Se expone, además, el diseño de la interfaz gráfica de la aplicación donde se muestra el acceso a cada modo y la ubicación de los módulos que contienen cada uno de ellos. Finalmente se expone la funcionalidad de la aplicación describiendo primero los procedimientos básicos que realiza la aplicación móvil y el uso de servicios de IEC 61850 con el proyecto abierto para luego describir la función de cada módulo para dar cumplimiento a los objetivos planteados en el proyecto de grado.

3.1. CONDICIONES DE DESARROLLO

La aplicación es desarrollada en el dispositivo móvil Samsung Galaxy Tab 2 7.0 cuya pantalla tiene una resolución de efectiva de 1024x530 píxeles en un área de siete pulgadas de longitud esquina a esquina. Este dispositivo cuenta con el sistema operativo Android versión 4.2.2 llamada *Jelly Bean*. En hardware, cuenta además con un procesador de doble núcleo de 1GHz cada uno y una capacidad de 1GB de memoria RAM.

Por otra parte, el software en el que se desarrolla la aplicación es Eclipse Kepler con el plugin ADT para obtener la suite de desarrollo de aplicaciones Android.

3.2. CONFIGURACIÓN DEL ARCHIVO MANIFEST.XML

La aplicación, es desarrollada para ajustar únicamente con la resolución del dispositivo usado en el desarrollo (1024x530), es configurada desde el archivo Manifest.xml para que su orientación siempre sea “Apaisada”, o con los lados más largo de la pantalla ubicados horizontalmente y cuenta con una única Activity

En materia de permisos, se conoce que es una aplicación de red por lo que tiene el permiso *android.permission.INTERNET* además, como se verá más adelante, la aplicación se apoya en la memoria principal del dispositivo móvil para ejecutar algunas funciones por lo que también cuenta con el permiso *android.permission.WRITE_EXTERNAL_STORAGE*.

3.3. ADICIÓN DE OPEN IEC 61850 A LA APLICACIÓN ANDROID

Embeber la librería OPEN IEC 61850 se puede realizar de dos formas. La primera es crear una librería jar a partir de los archivos .java pertenecientes al proyecto abierto cuidándose de no incluir clases que involucren la interfaz gráfica de Windows como la clase Log, Java Swing o Java AWT. Al crear la librería jar se importa a la carpeta *libs* del proyecto Android.

La otra opción es incluir el .jar proporcionado por el creador de la librería incluido en la distribución del 3 de septiembre de 2013 que aunque incluye algunas clases que hacen uso de estas librerías que no puede ejecutar Android pueden incluirse en la carpeta libs y hacer uso de ella siempre asegurándose de no llamar un método que lo use.

3.4. CLASES DE ANDROID COMUNES EN LA APLICACIÓN

Se expone brevemente la utilidad de las siguientes clases de Android que integran la mayoría de funciones de la aplicación tanto visual como funcionalmente.

(A) ViewFlipper

Es un contenedor de Layouts de una aplicación Android que permite poner en pantalla, programáticamente, uno a la vez. Es útil para aplicaciones que presentan múltiples modos o vistas donde a cada uno le pertenecen elementos gráficos distintos.

En la figura 11 se ilustra el funcionamiento de un ViewFlipper

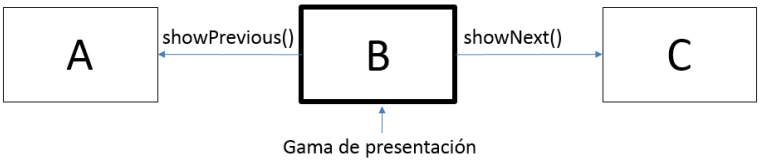


Figura 11. Acceso a los múltiples layouts de un ViewFlipper

ViewFlipper cargará automáticamente el primer layout declarado en el XML y se podrá cambiar de layout mediante los métodos showPrevious() y showNext() aunque también con setDisplayedChild(Int ID) se puede visualizar el layout identificado con la etiqueta ID.

(B) ArrayList

Es una clase que permite formar una estructura de datos vectoriales. Extiende a la clase java.util.List y es especialmente útil para almacenar en un vector los objetos de cualquier clase. La longitud del vector no es necesario definirla, inicialmente es cero y se incrementa en la medida en que se agregan elementos. También permite eliminar elementos de la lista.

El acceso a un objeto para su lectura se hace mediante el método get(int Index) donde index especifica la posición del objeto a leer. Para escribir un objeto existente en la lista en la posición *Index* se usa el método set(int Index). Finalmente para agregar y eliminar objetos en la posición *Index* se usa add(Clase objeto, Int Index) y remove(Int Index) respectivamente.

(C) ListView

Los *ListView* son elementos gráficos que sirven para ordenar en lista un grupo de elementos gráficos mediante el uso de *layouts* de XML. Sirve de interfaz visual

para un ArrayList ya que es capaz de representar visualmente los atributos de los objetos que contiene el ArrayList asociado.
Para esto es necesario un objeto de la clase *AdapterList* que le atribuye a cada ítem del ListView una misma apariencia basada en un *layout* XML cuyos UIElements son los que permitirán visualizar el atributo específico del objeto almacenado.
La figura 12 permite ver la utilidad de la clase.

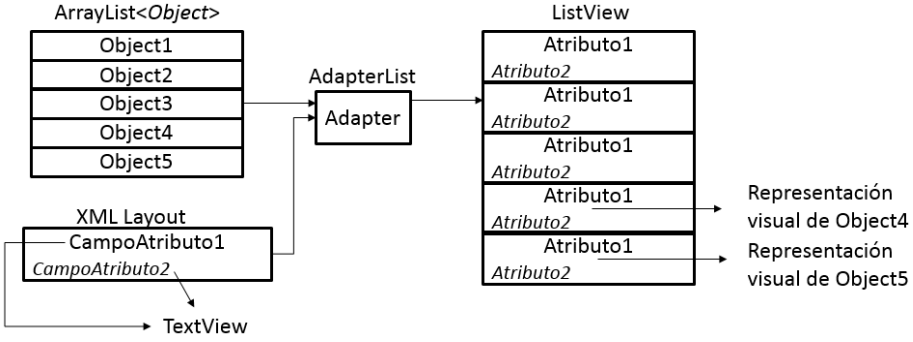


Figura 12. Composición de la interfaz ArrayList-ListView

El ListView tendrá la misma cantidad de objetos que el ArrayList. Cada vez que se efectúa un cambio en este último se requiere notificarle al adapter para que también se produzca el cambio en el ListView mediante el método *notifyDataChanged().*
La apariencia de cada ítem puede contener *TextView*, *Button*, y otros *UIElements*. A cada elemento que compone la apariencia se le puede asignar un atributo del objeto dentro del *ArrayList* lo que significa que el ArrayList-ListView es la interfaz ideal para mostrar en pantalla estructuras de datos vectoriales.
El usuario puede interactuar con esta lista tocando un elemento a través del *listener onItemClick* o tocándolo prolongadamente a través de *onLongClickListener* los cuales devuelven la posición tocada de la lista.

(D) Par Thread-Handler

Un hilo de clase Thread en Android no puede por sí sólo hacer cambios en la GUI de la aplicación. Para poder realizarlo se usa un Handler que debe ser instanciado en el hilo. Luego cada vez que se quiera efectuar un cambio en la GUI, el Thread debe enviarle un objeto de clase Message a su handler usando el método sendMessage(Message msj) y este hará un cambio a la GUI en función a msj. El handler por su parte ejecuta el método *handleMessage* cada vez que recibe un objeto enviado por su hilo.

(E) AsyncTask

Es una clase de Android que permite ejecutar una tarea en paralelo a la actividad principal teniendo la posibilidad, desde el hilo, de modificar la interfaz de usuario como parte del proceso en segundo plano. Es una forma fácil de manipular la GUI

sin pares *Thread-Handler*. El diagrama de cómo corre un proceso asíncrono se muestra en la figura 13.

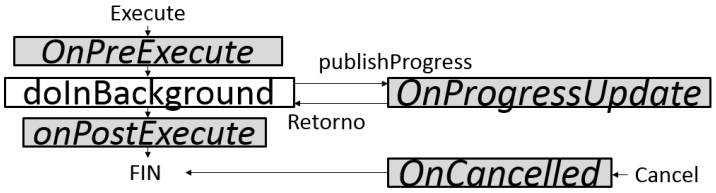


Figura 13. Diagrama de flujo de AsyncTask

Se muestra en cursiva los métodos de la clase que pueden modificar la interfaz gráfica. El método *doInBackground* no lo puede hacer. Para lograr exportar algún valor procesado dentro del método *doInBackground* se recurre al método *publishProgress* que enviará el valor para que sea leído por el método *OnProgressUpdate* y poder usarlo para la modificación de la *GUI* en función al valor exportado. Esta operación se puede hacer un número ilimitado de veces en el método doInBackground permitiendo cambiar la interfaz gráfica cada vez que el usuario lo necesite.
El método *OnPreExecute* permite introducir o preparar gráficamente la aplicación para informar que se va a iniciar un proceso y el método *OnPostExecute* permite informar sobre su finalización. En cualquier parte del proceso es posible cancelar la tarea solicitando externamente la cancelación del hilo, luego de esto se ejecutará *OnCancelled* que permite anunciar la interrupción del proceso.

3.5. DISEÑO VISUAL DE LA APLICACIÓN

La aplicación está compuesta por cuatro modos: El gestor de conexión, el navegador de nodos, Datasets y RCB y el panel de seguimiento. Para navegar entre ellos se usa un menú lateral. La figura 14 ilustra el modo como se compone la aplicación y el acceso a cada modo.

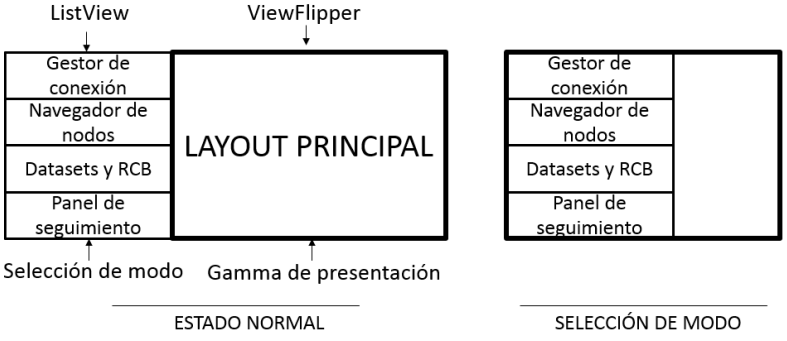


Figura 14. Composición de la aplicación y posiciones del menú lateral

La gama de presentación o lo que es mostrado por la pantalla del dispositivo es el *layout* principal. Este *layout* corresponde a un *ViewFlipper* que contiene los

distintos modos de la aplicación (cuatro RelativeLayout). Para navegar entre ellos, el *layout* principal tiene implementado el listener *onTouch()* para permitir arrastrar el menú con los cuatro modos a la gama de presentación para ser visible por el usuario. Una vez arrastrado el usuario escoge a qué modo puede ir gracias a que el ListView del menú implementa el *listener onItemClick.*

Ir a cada modo está regido por una serie de políticas que impiden según el estado de la aplicación ir de un modo a otro. Estos estados se verán en detalle más adelante y existen tres: 'No conectado a un IED', 'Conectado a un IED' y 'Archivo .ICD abierto'. Según estos estados los modos están restringidos según la siguiente lista:

- ✓ Modo Gestor de conexión: Es el modo inicial y en cualquier estado está disponible
- ✓ Modo Navegador de IED: Está disponible solo en los modos 'Conectado a un IED' y 'Archivo .ICD abierto'.
- ✓ Modo Datasets y RCB: Está disponible en los modos "Conectado a un IED" y "Archivo .ICD abierto".
- ✓ Modo Lista de seguimiento: Está disponible únicamente en el modo "Conectado a un IED".

Cada modo está compuesto por un *RelativeLayout* y la composición de todos se expone a continuación.

3.5.1 Modo Gestor de conexión

El modo gestor de conexión contiene UIElements que permiten la activación de los siguientes módulos:

- ✓ Administrador de perfiles de IED
- ✓ Módulo de conexión
- ✓ Información al usuario
- ✓ Apertura de archivos .ICD
- ✓ Configuración del cliente
- ✓ Configuración avanzada del cliente

La disposición de los UIElements que permiten al usuario el uso de estos módulos está ilustrada en la figura 15:

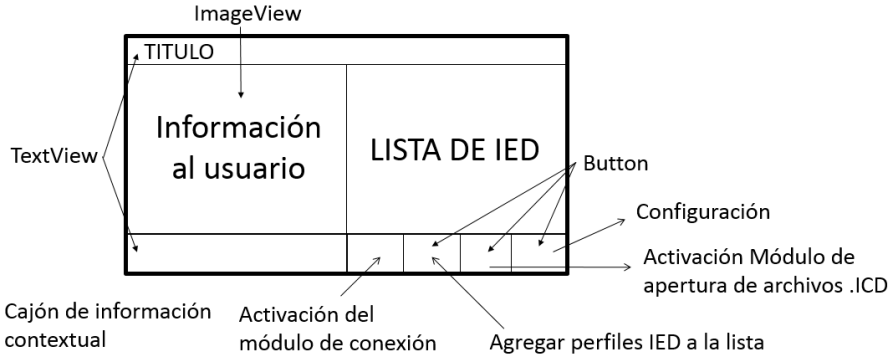


Figura 15. Composición del modo Gestor de Conexión

3.5.2. Modo Navegador de nodos

El modo de navegador de nodos permite ejecutar los siguientes módulos:

- ✓ Navegador de Nodos
- ✓ Módulo de adición a Dataset
- ✓ Operate y Select
- ✓ Asistente de lectura y escritura
- ✓ Elementos gráficos auxiliares (Dirección URI y Título)
- ✓ Organigrama
- ✓ Hilo de recepción continua

Cada módulo está organizado en el layout según la figura 16.

Figura 16. Composición del modo Navegador de Nodos

3.5.3 Modo Datasets y RCB

Este modo presenta dos módulos:

- ✓ Visor RCB
- ✓ Navegador de Datasets
- ✓ Asistente de lectura y escritura

Su respectivo *layout* es organizado de la siguiente manera.

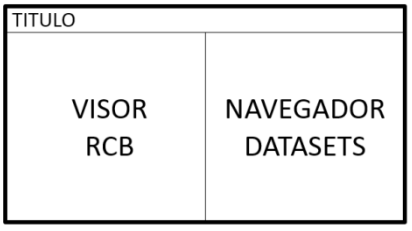


Figura 17. Composición del modo Datasets y RCB

3.5.4 Modo Panel de seguimiento

El modo de panel de seguimiento es un modo que reemplaza la barra del menú lateral por una lista de nodos a seguir. Si la variable en seguimiento es numérica el usuario podrá tocarla y cambiar el layout principal a un panel de graficación que muestra la dinámica de la variable. Este modo cuenta con los siguientes módulos:

- ✓ Panel de seguimiento
- ✓ Hilo de recepción continua

✓ Panel de graficación
✓ Envío de datos a servidor

3.6 PROCEDIMIENTOS BASICOS

Para el uso efectivo de OPEN IEC 61850 es necesario seguir sistemáticamente una secuencia de comandos determinada para llegar a los procedimientos descritos a continuación.

3.6.1 Procedimiento de conexión

Permite el vínculo con un IED a través de un socket creado en OPEN IEC 61850 correspondiente a la clase *ClientAssociation*. Se deben seguir los siguientes pasos para la construcción de un bloque de comunicación con el IED.

1. Se define un objeto de la clase *ClientSAP*.

```
ClientSap SapEjemplo = new ClientSap();
```

2. Opcionalmente se configura el objeto de ClientSap según los métodos de configuración.
3. Se crea un objeto ClientAsociation

```
ClientAsociation clienteEjemplo;
```

4. Finalmente se relacionan los dos objetos mediante el método *associate*.

```
clienteEjemplo=SapEjemplo.associate(DireccionIP, Puerto, null);
```

Esto ocasiona que el bloque de conexión constituido por los dos objetos envíe una petición de conexión a la dirección IP y puerto ingresados en el método. Si la conexión no tiene una respuesta dentro de un tiempo de espera, el enlace está caído o no hay un IED en la dirección IP y puerto al que fue enviada la petición, el procedimiento fracasa y el objeto de asociación queda nulo. De lo contrario, el objeto ClientAsociation será el objeto que enlaza el cliente y el IED hasta que se interrumpa la conexión o se haga una petición de desconexión.

La figura 18 muestra la disposición de cada objeto con respecto al IED y su papel en el enlace así como el modelo de caja negra del procedimiento.

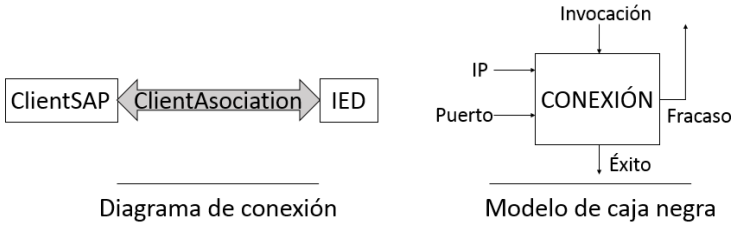


Figura 18. Modelo de conexión y caja negra del procedimiento

3.6.2 Procedimiento de lectura

Permite al cliente solicitar el valor de un BDA dentro del IED una vez se ha establecido la conexión. Esta petición se hace mediante el objeto que vincula al IED y el cliente que es el *ClientAssociation*. Se deben seguir los siguientes pasos para lograr lo anterior.

1. Referenciar un nodo dentro del modelo de IED obtenido con el servicio ACSI de *GetDataDirectory* y *GetAllDataValues* contenidos en el método *retreiveModel* de la clase *ClientAssociation*. Dado que este método devuelve un objeto de la clase *ServerModel* se puede usar el método de búsqueda de esta clase:

```
ModelNode a = serverModel.findModelNode(Ref, Fc);
```

El método permite encontrar nodos FC, es decir un nodo de DO hacia abajo en la jerarquía del modelo genérico de IED. Sin embargo se puede referenciar un LN o un LD con la misma función pero con un objeto nulo en el campo Fc.

2. Con el objeto *ModelNode* conseguido en el anterior paso se le solicita al IED el valor de esta variable mediante el servicio ACSI *getDataValues* a través del objeto ClientAssociation.

```
clienteEjemplo.getDataValues(a);
```

El IED responde con el valor del nodo y el de sus hijos, actualizando los atributos del objeto instanciado en el procedimiento. El IED puede no responder lo que hace que se lance una excepción anunciando el fracaso del procedimiento.

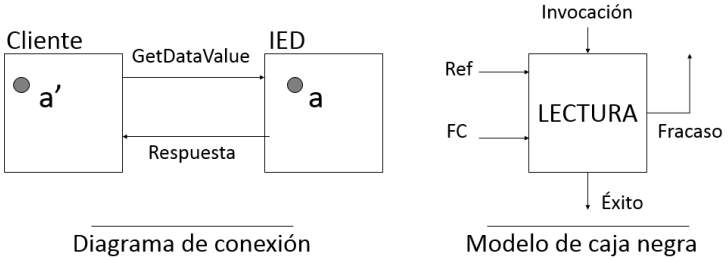


Figura 19. Lectura de variables y modelo de caja negra del procedimiento

En el diagrama *a'* es el objeto instanciado y elude a *a*, una variable existente en el IED que cambia como consecuencia del procedimiento que esté llevando a cabo. El cliente podrá saber el valor de *a* en el instante que ejecute el procedimiento de lectura. En ese momento *a'* queda con el valor de *a* y tendrán el mismo valor hasta que *a* vuelva a cambiar. Se debe, entonces, lanza el procedimiento cada vez que se desee conocer el último valor de la variable *a* y en general cualquier variable dentro del IED.

3.6.3 Procedimiento de escritura

Permite escribir un valor de una variable o grupo de variables en el IED. Para lograrlo se debe realizar lo siguiente:

1. Referenciar el nodo, se hace exactamente de la misma forma como se describe en el paso uno del procedimiento de lectura.

```
ModelNode a = serverModel.findModelNode(Ref, Fc);
```

2. Según el tipo de dato a escribir se usan sus métodos se escritura definidos. Estos métodos son presentados en el capítulo 2. Para el reconocimiento del tipo de dato a escribir se usan algoritmos auxiliares que son presentados más adelante.

3. Finalmente se usa el servicio ACSI *SetDataValues* a través del objeto *ClientAssociation*.

```
clienteEjemplo.getDataValues(a);
```

El IED recibe la petición de escritura y si el nodo referenciado es escribible se cambiará el valor de la variable del IED.

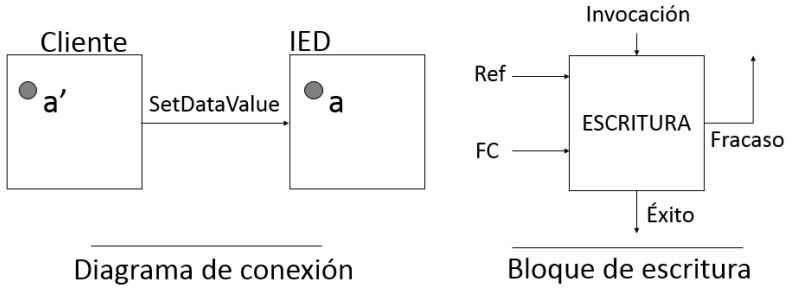


Figura 20. Escritura de variables y modelo de caja negra del procedimiento

Según muestra el diagrama, la variable *a'* es creada a partir de la referencia de una variable existente en el IED y es cambiada en el cliente. Estos cambios surten efecto en el IED cuando es enviada mediante el servicio *SetDataValue*.

3.6.4 Procedimiento de búsqueda

En OPEN IEC 61850 es recurrente tener colecciones de objetos que corresponden a los siguientes casos:

- ✓ Los hijos de un Nodo
- ✓ Los hijos de un Nodo FC
- ✓ Los Datasets configurados en el ServerModel
- ✓ Los RCB configurados en el ServerModel
- ✓ Los miembros de un Dataset

En general se puede obviar el uso de índice de iteración para bucles *for* cuando se trata con una Colección o Lista en JAVA. Se puede acceder a todos los pertenecientes a una Lista instanciado en cada iteración una variable del mismo tipo que los miembros de ella.

Para el caso de los hijos de un nodo se la forma del hacer la búsqueda es:

```
for(ModelNode hijo : Nodo.getChildren()){
       if(hijo.getName == Busqueda) ...;
}
```

Para el caso de la búsqueda de hijos de nodos FC

```
for(FcModelNode hijoFC : Nodo.getChildren()){
       if(urcb.getName() == Busqueda
              && hijoFC == FCBuscqueda) ...;
}
```

Los datasets se buscan de la siguiente forma:

```
for(Dataset dataset : ServerModel.getDataSets()){
       if(dataset.getReferenceString() == Busqueda) ...;
}
```

Para el caso de los miembros de los datasets:

```
for(FcModelNode miembro : Dataset.getMembers()){
       if(miembro.getName() == Busqueda) ...;
}
```

Finalmente los RCB del ServerModel se buscan con las siguientes líneas:

```
for(Urcb urcb : Dataset.getUrcbs()){
       if(urcb.getName() == Busqueda) ...;
}
```

Los puntos suspensivos son la parte en el algoritmo de búsqueda que permite ejercer una acción como asignarle el objeto encontrado a una variable.

3.6.5 Encontrar tipo de cualquier miembro de ModelServer

Encontrar el tipo de un nodo se puede hacer de dos distintas formas: usando la enumeración BdaType o con el método getClass(). Para el proyecto se ha decidido hacer el segundo puesto que la enumeración carece del tipo contenedor (ConstructedDataAttribute) y del tipo vector (Array) mientras que con las dos siguientes líneas no sólo se puede saber qué tipo de variable es sino que además a que clase pertenece:

```
String tipo = ""+Node.getClass();
tipo = tipo.substring(31);
```

La primera línea devuelve String del tipo: "class org.openmuc.openiec61850.*CLASE*" con la que se puede identificar cada clase de cualquier miembro del ServerModel. La segunda línea recorta el String dejándolo en "CLASE".

3.7 MÓDULOS

Todas las funciones de la aplicación se agrupan en módulos y cada módulo se encuentra en alguno de los siguientes modos de la aplicación.

3.7.1 Administrador de perfiles de IED

El modo de Gestor de conexión permite especificar la dirección IP del IED al que se quiere conectar, configurar el SAP y otros parámetros respecto al funcionamiento de la aplicación. El gestor de conexión consta de los siguientes módulos:

(A) Lista de IED

La lista de IEDs es un *ListView* que almacena objetos de la clase *IEDProfile*. La composición gráfica de este módulo es la mostrada en la figura 21:

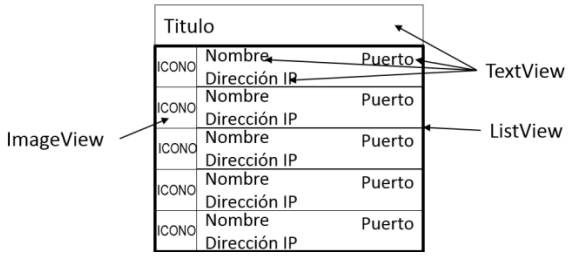


Figura 21. Composición gráfica del módulo Lista de IED

IEDProfile representa un IED y contiene la información necesaria para ser contactado por el cliente. También tiene información para identificar cada IED en un ambiente industrial. La tabla 18 muestra los atributos y métodos de un objeto IEDProfiles cualquiera:

ATRIBUTO	TIPO	Escritura	Lectura
nombre	String	void setName(String name)	String getName()
LAN	String	void setLAN(String fc)	String getLAN()
IP	String	void setIP(String ip)	String getIP()
Fabricante	String	void setFab(String fab)	String getFab()
Puerto	int	void setPort(Int Puerto)	Int getPort()
Id	long	void setID(Long id)	long getID()
Inicialización			
void setAll(String n, String lan, String ip, String fab, int port, long id)			

Tabla 18. Atributos y métodos de un objeto de la clase IEDProfile

Para añadir objetos de esta clase a la lista *IEDLV* se usa el botón *"Agregar perfiles a la lista"* presente en la Fig. 5.4.1.1. La acción de este botón carga un diálogo con un formulario que el usuario debe llenar para registrar un perfil IED e incluirlo en la lista. El formulario tiene la estructura mostrada en la figura 22:

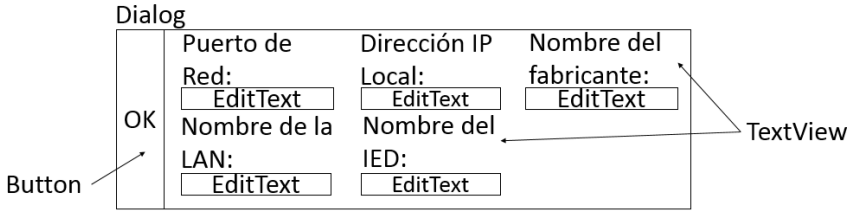


Figura 22. Dialogo de adición de perfil de IED

Los campos que se llenan en el dialogo concuerdan con los atributos de la clase *IEDProflie.*

El *ListView IEDLV* tiene implementados dos *listeners*:

OnItemClick - El usuario toca el IED de la lista al cual se quiere conectar. Cuando esto ocurre, la dirección IP y puerto del perfil tocado se le asignan al bloque de procedimiento de conexión y se activa el botón *"Activar módulo de conexión"*.

OnLongClickLister - El usuario toca prolongadamente el IED que quiere modificar o eliminar. Se abre un diálogo con las dos opciones.

Si el usuario escoge eliminar, se llama el método *remove(int index)* de *ListView* donde *index* es la posición del elemento a borrar. Si por el contrario, el usuario escoge modificar se abre un dialogo con los mismos campos que en la figura 22 para poder cambiarle los parámetros al IED tocado. Primero la aplicación borra el objeto y luego crea uno nuevo con la información suministrada en la posición tocada de modo que se mantenga igual el número de perfiles en la lista.

(B) ALMACENAMIENTO DE PERFILES DE IED

Se trata de un procedimiento para mantener los perfiles creados en la lista *IEDLV* a pesar de que la aplicación sea finalizada. Para lograrlo la clase *IEDProfile* implementa la clase *Serializable.* Esto permite guardar objetos de esta clase en un fichero dentro de la memoria del dispositivo. Se ha escogido la dirección URI "*Environment.getExternalStorageDirectory()/IED Profiles"* para guardar estos perfiles. Esta dirección URI corresponde a una carpeta raíz de la memoria principal del dispositivo móvil.

La creación de los archivos ocurre cada vez que se añade un perfil de IED a la lista o cuando se modifica. La exportación de objetos desde la aplicación se realiza mediante las clases *ObjectOutputStream* y *FileOutputStream* y gracias al método *writeObject* de la primera clase.

El manejo de los ítems dentro de la lista *IEDLV* se realiza mediante la clase *IEDListManager*. Sus algoritmos más importantes pueden verse en la figura 23.

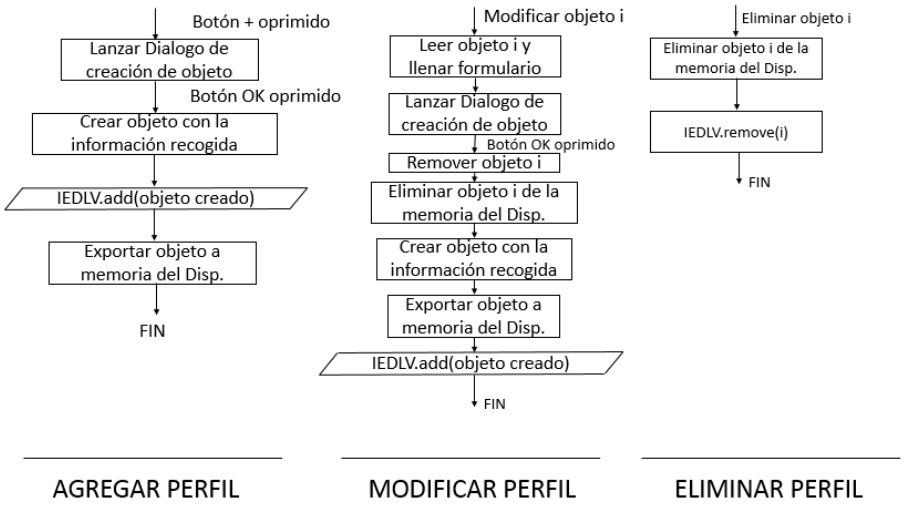


Figura 23. Procedimientos en la clase *IEDListManager*

Cabe anotar que lo que se agrega a IEDLV son objetos de la clase IEDProfile de los cuales serán visibles sólo los atributos *nombre*, *IP* y *Puerto.*

(C) Inicialización de la lista de perfiles IED

Al inicio de cada ejecución de la aplicación se debe inicializar la lista de ficheros contenidos en la carpeta *IED Profiles.* Se buscan entonces todos los ficheros y se cargan mediante las clases *FileInputStream* y *ObjectInputStream* y gracias al método *readObject* de la primera clase.

El algoritmo que se sigue es el que se muestra en la imagen.

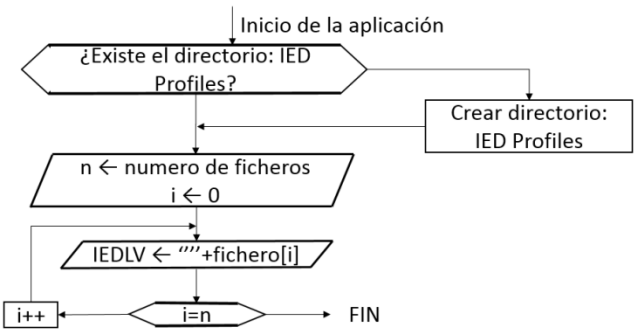


Figura 24. Inicialización de los objetos IEDProfile guardados

Donde IEDLV ← “”+fichero[i] significa adicionar un nuevo ítem en el ListView con el nombre del objeto guardado en el fichero i de la carpeta IED Profiles.

3.7.2 Módulo de conexión

El módulo de conexión se trata de un hilo asíncrono de la clase *AsyncTask.*

La ejecución de procesos relacionados con el uso de sockets en el sistema operativo Android tiene como principal regla la necesidad de usar sus métodos en un hilo diferente al de la actividad principal. Esto se debe a que el proceso estará congelado ante una eventual asociación tardía con un host externo. El objetivo es que la actividad principal siempre esté disponible a la interacción con el usuario y el argumento anterior se extiende también a procesos largos como procesos de búsqueda, de mapeo o de cálculo que puedan interferir con la fluidez de la aplicación.

El algoritmo simplificado del hilo de conexión es el que se muestra en la figura 25.

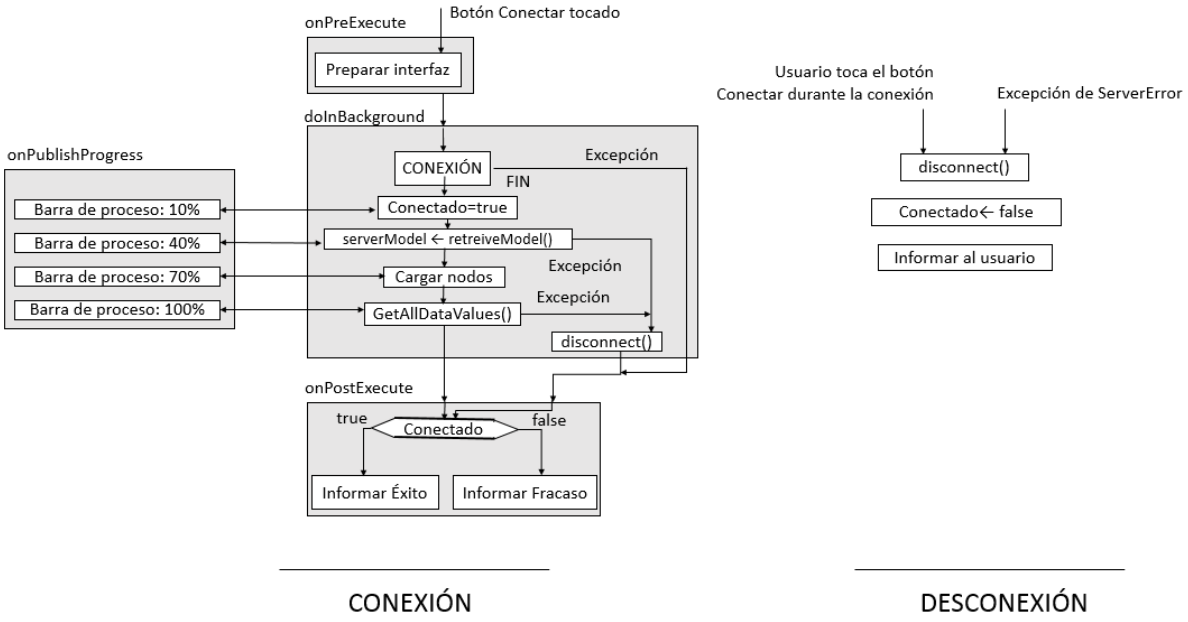


Figura 25. Algoritmo de conexión y el uso de Asynctask

Para accionar este método se debe seleccionar primero un IED de la lista *IEDLV* luego tocar el botón *conectar*.
Esto ocasiona que la aplicación se lo informe al usuario con el *módulo de información.* Luego se usa uno de los procedimientos básicos, el de *Conexión* usando como parámetros de entrada las variables globales *IedIp*, *IedPuerto.* Se usa una variable de control llamada *Conectado* que indica si el hilo tuvo éxito al conectarse al IED. En base al valor de esta variable se le informa al usuario el éxito o fracaso del proceso en el método *onPostExecute* de AsyncTask mediante el *módulo de información*.
Si se ha tenido éxito en la conexión se usará el servicio ACSI retreiveModel() para luego cargar los nodos en la lista *NODESLV* (Se hablará de ella en el módulo de *navegación de nodos*) y se usa el servicio GetAllDataValues() para cargar el valor de todas las variables al modelo conseguido. En cada parte del proceso se mantendrá informado el usuario el porcentaje de completado mediante una barra de proceso manejada desde el método *onPublishProgress*.
El módulo puede lanzar excepciones en los bloques de *Conexión* o en el uso de los servicios ACSI *retreiveModel()* y *GetAllDataValues().* En el primer caso se termina de inmediato el método doInBackground lanzando seguidamente el método *onPostExecute* mientras en los otros casos se cierra el socket primero con el método *close()* de *ClientAssociation* antes de finalizar el método para dejarlo cerrado.
Para desconectarlo simplemente se cierra el socket con el método disconnect(), Se cambia el valor de la bandera Conectado a false y se le informa al usaurio. Finalmente si ocurre una excepción de tipo ServiceError se cierra el socker automáticamente.

3.7.3 CONFIGURACIÓN DEL CLIENTE

Consiste en un botón que lanza un diálogo que permite modificar los parámetros que están dentro de la forma del dialogo mostrada en la figura 26.

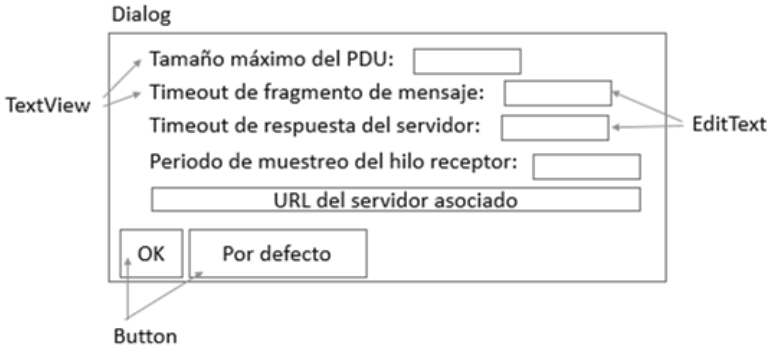


Figura 26. Composición gráfica del dialogo de configuración del cliente

Varios de estos parámetros son valores configurables en el objeto *ClientSAP.* El periodo de muestreo del hilo receptor, correspondiente a la variable entera

tiempoMuestro, se tratará más adelante en el módulo asistente de lectura y escritura.

3.7.4 CONFIGURACIÓN AVANZADA DEL CLIENTE

Este módulo consiste en un botón que lanza un diálogo con la opción de configurar los bytes de los selectores T-SEL que deben corresponder al T-SEL configurado en el archivo .ICD usado en los IED. Se tienen los valores estándar de este parámetro para sus dos partes (T-SEL Local y T-SEL Remote) y no deben ser cambiados a menos que sea estrictamente necesario. El dialogo tiene la forma mostrada en la figura 27.

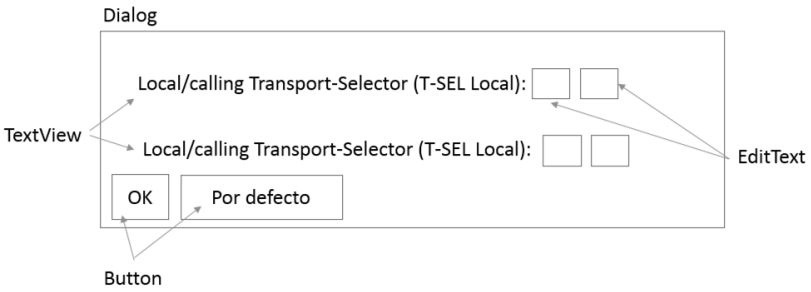


Figura 27. Composición gráfica del dialogo de configuración avanzada

Cada casilla representa un byte y los valores por defecto son:
T-SEL Local: [0 0]
T-SEL Remote: [0 1]

3.7.5 Módulo de apertura de archivos .ICD

Este módulo consiste en que el usuario pueda abrir un archivo .ICD y visualizar el modelo que se ha descrito en él. Esta operación es local y al igual que el módulo de conexión cargará los nodos encontrados en el archivo .ICD en la lista *NODESLV* (Se tratará en el módulo de *navegación de nodos*).
Mediante un botón se lanza un Diálogo que contiene una lista con los archivos .ICD presentes en la carpeta "*Environment.getExternalStorageDirectory()/ICDFiles*" lo que significa que es una carpeta alojada en la memoria principal del dispositivo móvil.
El proceso de cómo ocurre la acción de listar los archivos y la forma del dialogo se muestran en la figura 28.

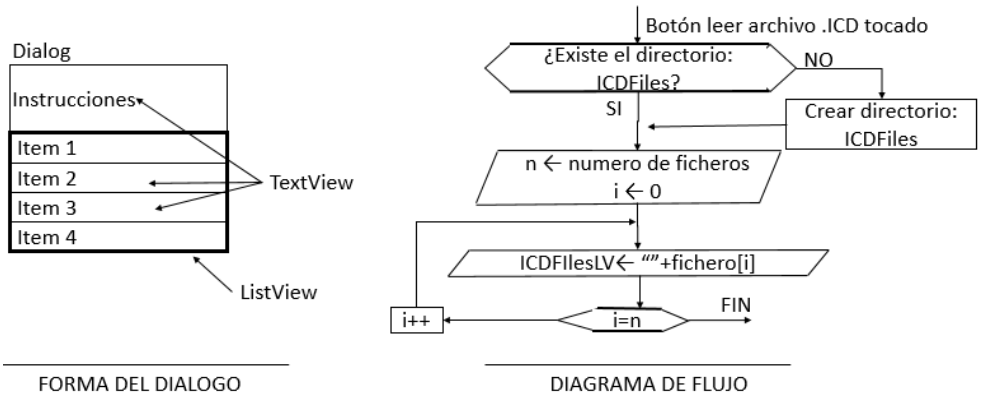


Figura 28. Dialogo de adición a Datasets y lectura de archivos .ICD

Donde *ICDFilesLV* es el *ListView* que contendrá los nombres de los archivos en la carpeta *ICDFiles*.
Esta lista tiene el listener onItemClick que permite al usuario abrir el script SCL tocado, el procedimiento de apertura es el mostrado en la figura 29.

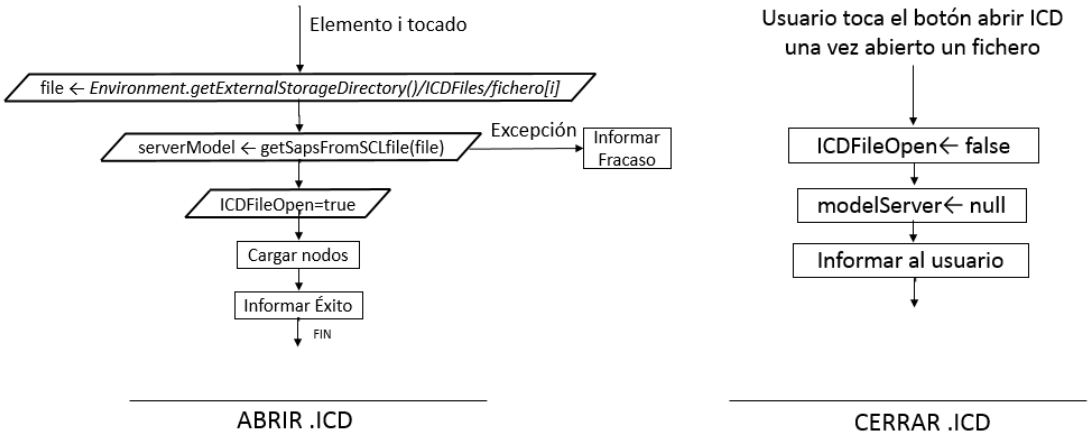


Figura 29. Diagrama de flujo para abrir un archivo SCL en el programa

Se usa el método *getSapsFromSCLFile(File file)* de la clase *ServerSAP* para lo cual se instancia un objeto de ella para poder usarlo. El método devuelve un objeto de la clase *ServerModel* y se le asigna a una variable global llamada *serverModel* que es usada también por el módulo de conexión y que contendrá el modelo del IED en la ejecución de la aplicación.
Dada la limitación que se trató anteriormente en la descripción de la clase ServerModel, no es posible ver los RCB configurados en el script SCL, sólo los nodos y los datasets.
Lanzar este método puede lanzar una excepción que si ocurre será informada al usuario mediante un *Toast*. Finalmente se usa la variable *ICDFileOpen* como bandera para saber si la aplicación tiene abierto un script SCL o no.

El usuario puede cerrar el archivo .ICD para usar el módulo de conexión para lo cual se restaura la bandera *ICDFileOpen* se borra el modelo en ServerModel y se le informa al usuario.

3.7.6 Módulo de información al usuario

Se usan dos elementos gráficos para darle información al usuario: Un ImageView que cambia periódicamente para presentar el programa y un TextView para informar el estado del mismo.

(A) PRESENTACIÓN DE GUIA AL USUARIO

Este breve procedimiento permite guiar al usuario en el uso de la aplicación explicando cada uno de los botones que esta tiene. La primera parte Consiste en una presentación de imágenes a modo de diapositiva cambiando cada cinco segundos. Para ello se usa el vector *info* con los id de las imágenes en la presentación. En Android estos id son números enteros que identifican cada recurso y para el cambio continuo de las imágenes en la diapositiva se usa un par Thread-Handler. Para mostrar las imágenes en pantalla se usa un *UIElement* llamado *ImageView*. El diagrama de flujo es el mostrado en la figura 30

.

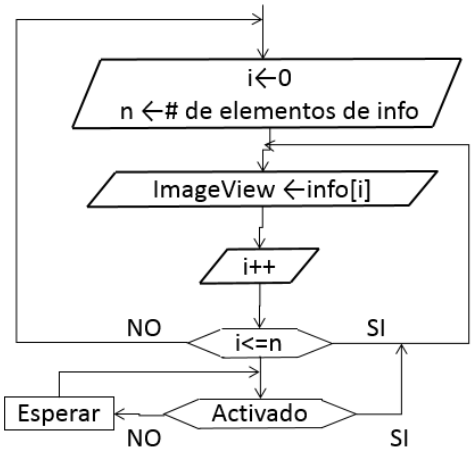


Figura 30. Control de la presentación en el ImageView

Donde *Activado* se trata de una variable para poder detener el hilo programáticamente. Para efectos de lectura el operador "←" significa mostrar en el ImageView la imagen info[i].

(B) CAJON DE INFORMACION CONTEXTUAL

Es un *TextView* en el que se informa al usuario estando en el modo de gestor de conexión, el resultado de los módulos de apertura de archivos .ICD y el módulo de conexión que como se vio tiene dos opciones al ser lanzado, el éxito o el fracaso. Cada cajón en los diagramas de flujo anteriores de informar al usuario el éxito o fracaso del procedimiento lo que se hará es poner un texto apropiado para los dos desenlaces en este TextView.

3.7.7 Navegador de nodos

El navegador de archivos permite explorar el modelo de un IED gravado en un objeto de clase *ModelServer*. El cliente puede hacerse a este objeto de dos formas:

- ✓ El modo local que permite transformar un archivo .ICD en un objeto *ModelServer*
- ✓ El modo de conexión. Tema central del proyecto y consiste en obtener el objeto *ModelServer* mediante los servicios ACSI de la norma IEC 61850.

En cualquiera de los dos procedimientos se asigna el objeto *ModelServer* conseguido a la variable global *modelServer*.

El navegador en si consiste en un conjunto de elementos gráficos que le permite al usuario moverse por los niveles jerárquicos de un IED, representado mediante el objeto *modelServer.* Cómo están dispuestos los UIElements en la pantalla según el diseño del navegador se puede observar en la pantalla de la figura 31.

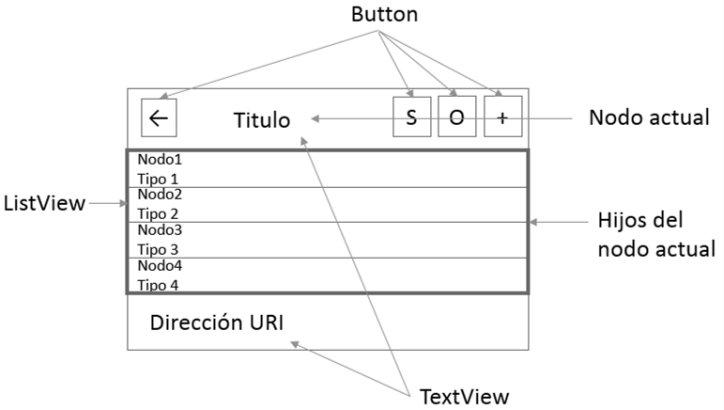


Figura 31. Composición del navegador de nodos

(A) OBJETO DE PERFIL DE NODO

Este objeto será el que se almacene en el ArrayList del ListView *NODESLV.* Pertenecen a la clase NodeProfile

ATRIBUTO	TIPO	Escritura	Lectura
nombre	String	void setName(String name)	String getName()
Fc	String	void setFc(String fc)	String getFc()
bdaType	String	void setBdaType(String type)	String getbdaType()
Path	String	void setPath(String path)	String getPath()
Id	Long	void setID(Long id)	Long getId()
Inicialización			
void setAll(String n, String fc, String type, String path, long id)			

Tabla 19. Atributos y métodos de los objetos de la clase NodeProfile

Cada ítem del ListView NODESLV tiene dos TextView. En el primero se mostrará el atributo *nombre* del objeto almacenado y en el segundo se mostrará el atributo *bdaType*. Se ha decidido poner en este último la siguiente información según el nivel jerárquico:

- ✓ Nivel de LD: Se escribe a la clase a la que pertenece. En todos los casos a la clase *LogicalDevice.*
- ✓ Nivel de LN: Tipo de nodo según la nomenclatura de nodos lógicos. Si el nodo se encuentra en un CDC se pondrá la clase específica. Por ejemplo para los nodos cuyo nombre empieza por la letra L se escribirá en el TextView que es un nodo del "sistema" o si el nombre del LN es MMXUn se escribirá que es una "unidad de medida".
- ✓ Nivel de DO: Vacío
- ✓ Nivel FC: Se pone un String que identifica el FC. Por ejemplo para MX se pondrá “Medición”.
- ✓ Nivel de DA e inferiores: Tipo de BDA o y el DA corresponde a un Array o un Contenedor. Se consigue gracias al método *getType()* de este nivel.

(B) ListView

Corresponde a la variable global *NODESLV* y en donde se mostrarán los hijos del nodo en donde se ubica el usuario en un instante dentro del modelo jerárquico del IED por lo que se tendrá un puntero de tipo *ModelNode* que indique en cual miembro se encuentra. Este puntero se ha nombrado como *NAV.* También se usa un puntero de tipo entero llamado *level* que indica el nivel jerárquico donde se encuentra el usuario e inicialmente con valor igual a uno.

La condición inicial de la lista *NODESLV* luego de usar el módulo de conexión o el módulo de apertura de archivo .ICD será alojar una lista con los hijos de la variable *ServerModel* que corresponde al nivel de LD. Lo anterior se logra con el algoritmo de la figura 32.

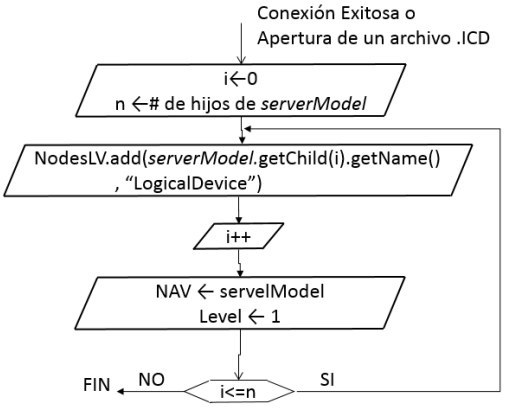


Figura 32. Inicialización de NODESLV luego de obtener un *serverModel*.

El bloque donde se usa el método add de ArrayList simboliza la creación de un objeto NodeProfile con los parámetros *nombre* y *fc* iguales a los dos parámetros (separados por comas) del bloque de asignación que está siendo agregado a la lista para ser visualizado.

(C) Ingreso a un nodo

Entrar a un nodo que presente *NODESLV* se realiza con el listener onClickListener. El diagrama de flujo se presenta en la figura 33.

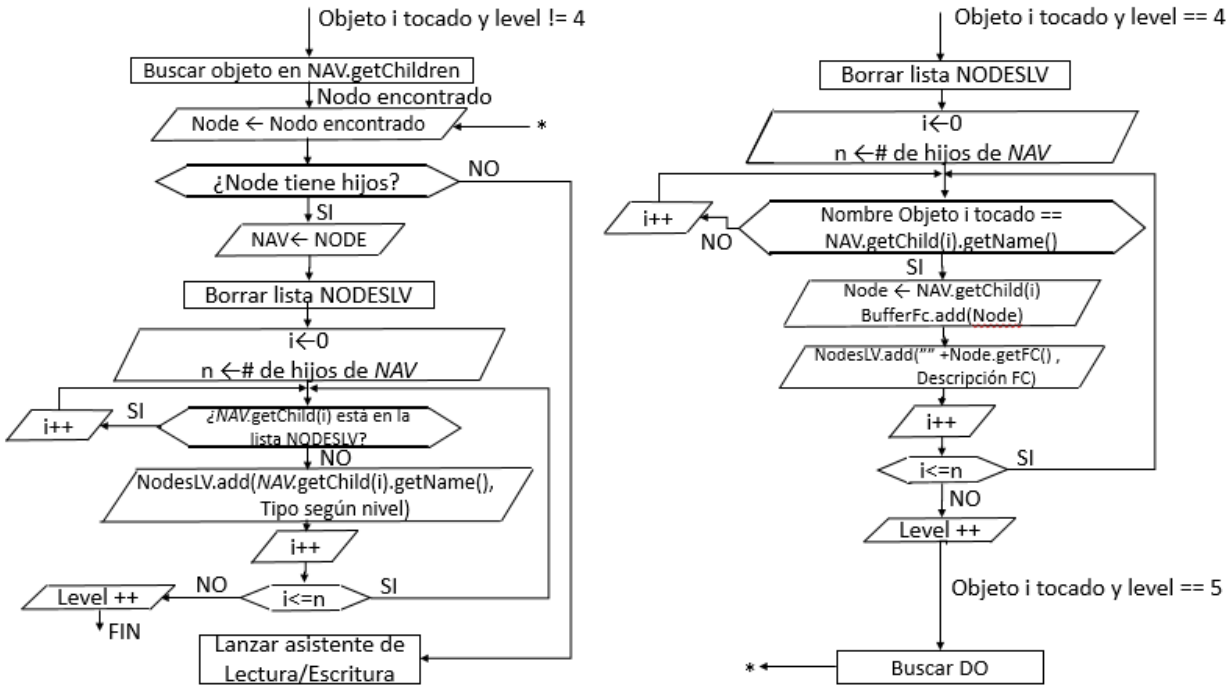


Figura 33. Algoritmo de navegación de nodos

Se usa el ListView para generar una vista que permita al usuario explorar la estructura jerárquica creada en los scripts SCL.

El diagrama de la derecha sugiere la creación de un nivel intermedio entre DO y DA para los FC de modo que cuando el usuario oprima sobre un DO se listen sus FC. Cabe anotar que es común encontrar dentro de un LN, DOs con el mismo nombre pero cada uno con diferente FC y el diagrama de la izquierda prevé esta situación excluyendo cualquier nodo con nombre repetido listando sólo uno.

La exclusión de estos otros nodos no tiene ninguna implicación de pérdida de información ya que el algoritmo de búsqueda solo compara nombres de DO hacia arriba y nombres y FC de DA hacia abajo.

Este bloque de “Buscar nodo FC en NAV.getChildren” corresponde al procedimiento básico de búsqueda de nodos FC por lo cual se compara tanto el nombre como el FC del nodo.

(D) Retroceso

Para devolverse o subir un nivel en el mapa jerárquico el usuario debe oprimir el botón atrás simbolizado en la figura 31 con el carácter "←". Al tocar este botón el procedimiento seguido por la aplicación es el mostrado en la figura 34.

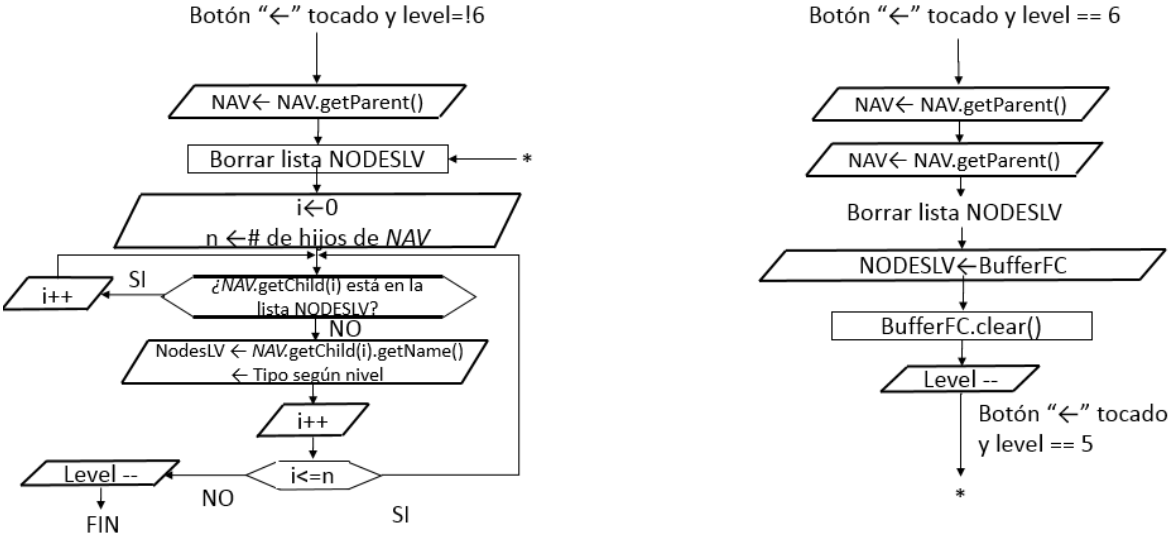


Figura 34. Algoritmo de Retroceso en la navegación

El objetivo del algoritmo es reescribir NAV con el valor del padre de NAV actual para luego poner en *NODESLV* el nombre y tipo de sus hijos.

El diagrama de flujo de la derecha mantiene coherente la inclusión del nivel FC que se ha querido agregar en el navegador de nodos cargando la lista de FC del DO actual que estaba guardada en el buffer BufferFC.

3.7.8 Módulo de adición a datasets

Se trata de un botón que lanza un Dialogo que permite la creación de un dataset en el servidor mediante el servicio ACSI *CreateDataset*. El botón es el que se representa en la figura 31 con el carácter "+". Este botón sólo aparece del nivel de los DA hacia abajo puesto que el servicio CreateDataset de la clase *ClientAssociation* sólo admite nodos Fc para integrar un dataset.

Al oprimir el botón "+" se mostrará un dialogo con la forma mostrada en la figura 35.

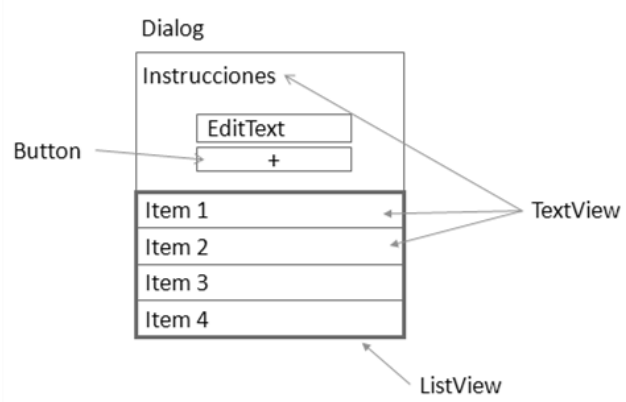


Figura 35. Dialogo para la adición/creación de dataset a partir de NAV

El Dialogo carga todos los datasets borrables en el ListView del dialogo listando cada elemento de *serverModel.getDatasets()*.

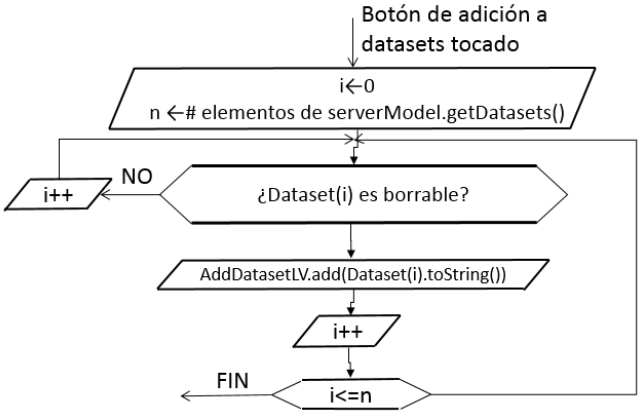


Figura 36. Diagrama de flujo para cargar los datasets borrables

Los datasets borrables son los que han sido creados mediante el servicio *CreateDataset*. En este dialogo, luego de la búsqueda, el usuario tiene dos opciones. Agregar el nodo NAV a un dataset borrable existente o crear uno nuevo a partir del puntero NAV.
Los algoritmos de agregar y adicionar son los que se muestran en la figura 37.

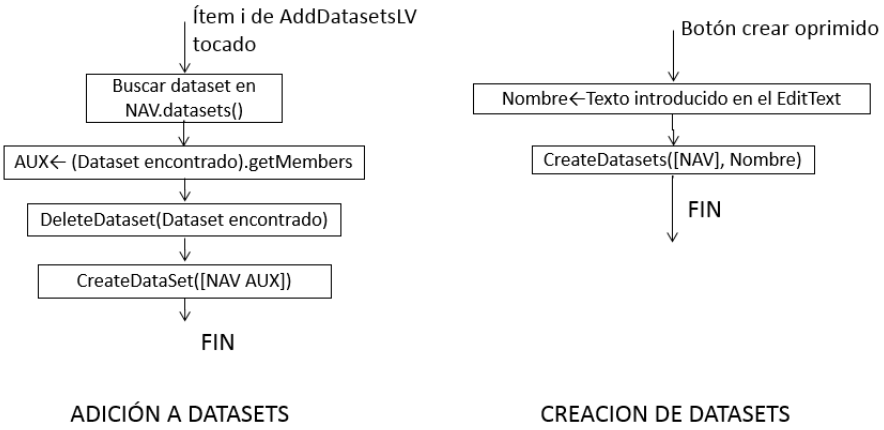


Figura 37. Adicionar NAV a un dataset existente o crearlo a partir de él

El método de búsqueda, al igual que en el navegador de nodos, se hace comparando los nombres de los datasets con la cadena de texto tomada del ListView.

3.7.9 Operate y Select

Operate y Select funcionan sólo si un DO contiene dos FC. El primero de control (CO) y el segundo de configuración (CF). Las condiciones para que un nodo sea de tipos SBO (Select before Operate) son:

En los DA de configuración debe existir un DA llamado ctlModel y éste tiene un valor de 2 (Este número hace parte de los Enums de la norma) significa que la escritura de este DO es de tipo SBO con seguridad normal.

En los DA de control también debe haber un DA llamado Oper.

Para su implementación se usa el navegador de nodos que comprueba las anteriores condiciones. Si es así se hace la petición del valor de la variable. Si este valor corresponde a 2 aparecerá en la barra de título el botón *Select* simbolizado en la figura 31 con el carácter “S”. Cuando el usuario oprime el botón se solicita el servicio ACSI Select. Si es exitoso se tendrá un tiempo límite para operar la variable cambiando valores del DO (El tiempo lo determina la variable sboTm perteneciente al nodo con FC de configuración). Para que los cambios tengan efecto se debe oprimir el botón "O" que aparece si el servicio *Select* fue exitoso, lo cual escribirá correctamente la variable.

Lo anterior se puede observar en el diagrama de la figura 38

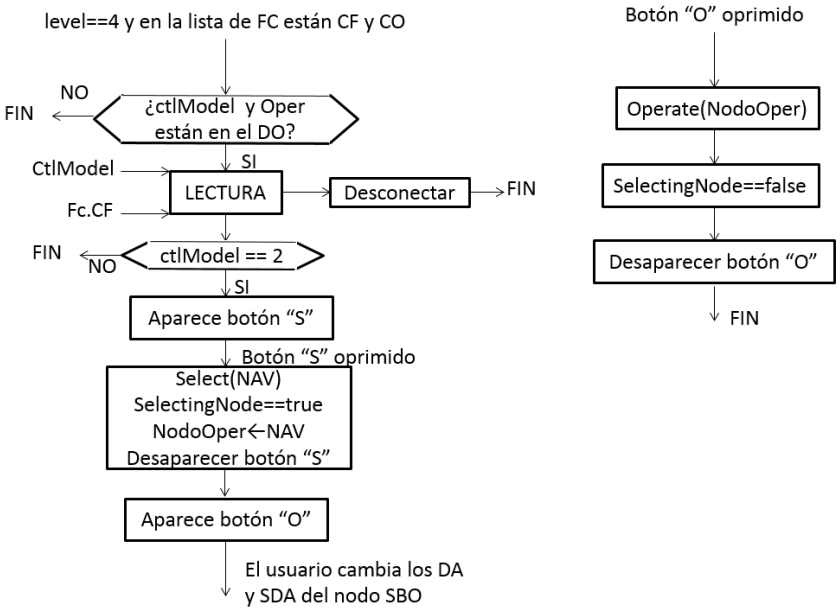


Figura 38. Algoritmo para ejecutar los servicios ACSI Select y Operate

3.7.10 Asistente de lectura y escritura

Se trata de un grupo de diálogos dedicados a la lectura y escritura de cada uno de los tipos de datos de OPEN IEC 61850. El objetivo es mostrar toda la información que contiene cada tipo de dato con *UIElements*. El asistente de lectura y escritura tiene la forma de la figura 39, general para todos:

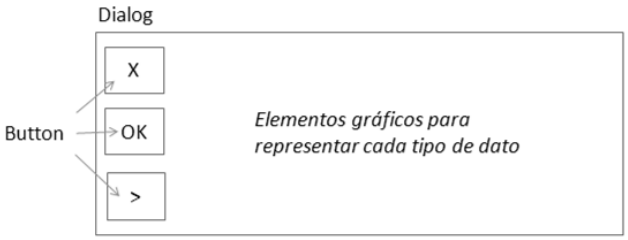


Figura 39. Composición gráfica general para los diálogos del asistente

Los botones usados ejecutan las siguientes funciones:

X - Cierra el diálogo

OK - Recoge la información de los elementos gráficos para escribir la variable con el valor puesto por el usuario en ellos.

> - El nodo abierto es agregado al panel de seguimiento (Se verá en detalle más adelante).

Cada diálogo tiene los elementos gráficos suficientes para representar cada tipo de dato. La forma como está constituido cada diálogo es la siguiente:

(A) NUMERICOS Y TEXTO: BdaInt8, BdaInt8U, BdaInt16, BdaInt16U, BdaInt32, BdaInt32U, BdaInt64, BdaFloat32, BdaFloat64. BdaUnicodeString, BdaVisibleString, BdaOctectString.

Estos tipos de dato solo presentan su valor numérico por lo cual basta con un EditText tanto para mostrar el dato leído como para poder ser escrito.

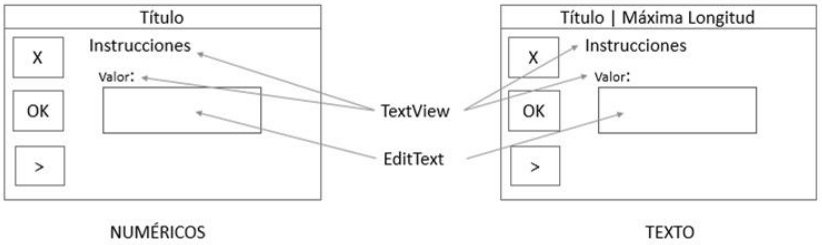


Figura 40. Dialogo para variables numéricas o de texto

(B) BdaBitString y BdaBoolean

BdaBitString se compone de cadenas de ceros y unos para lo cual se han puesto dos botones respectivos para realizar la descripción de la cadena. El Textview se llena en la medida que el usuario introduce ceros y unos. Para el BdaBoolean Se usa un UIElement llamado Switch que representa un valor binario. Se ha utilizado para que represente el valor booleano de este tipo de dato.

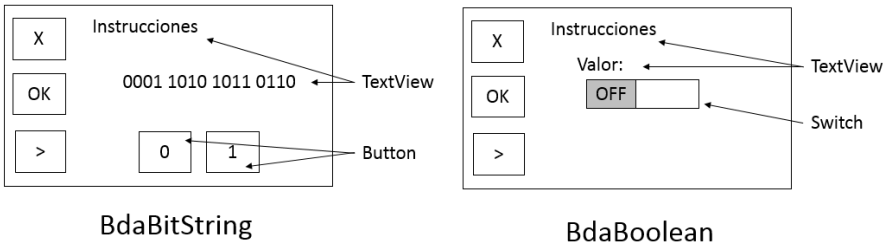


Figura 41. Dialogo para variables de tipo BdaBitString y BdaBoolean

(C) BdaCheck y BdaDoubleBitPos

BdaCheck tiene dos atributos booleanos por lo que han usado dos Switchs. Mientras que BdaDoubleBitPos puede tener cuatro valores: ON, OFF, BAD y INTERMEDIATE_STATE. Para esto se ha puesto un botón selector que mantiene solo uno de los Switches habilitados. Inicialmente se puede escoger con el switch de la izquierda los valores ON y OFF. Una vez se oprima el selector se inhabilita el switch de la izquierda y se habilita el de la derecha pudiendo elegir entre los valores BAD e INTERMEDIATE.

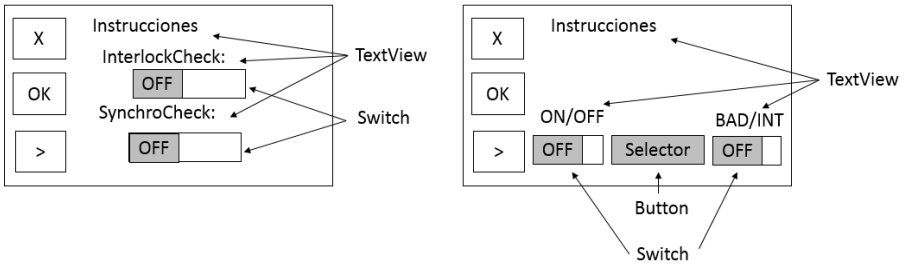


Figura 42. Dialogo para variables de tipo BdaCheck y BdaDoubleBitPos

(D) BdaTimestamp y BdaOptFlds

BdaTimestamp tiene un valor de java.util.Date. La fecha del objeto Date se ha representado con un elemento gráfico llamado TimePicker. Las horas, minutos y segundos se representan mediante tres TextView respectivamente.

BdaTimestamp presente además tres valores booleanos que se representan mediante Swtichs. El atributo TimeAccuracy está en un TextView mientras que FractionofSecond y LeapSecondsKnown están en un TextView y la razón es que el método setDate no incluye estos dos últimos atributos por lo que son de sólo lectura. Para este tipo de dato se ha incluido el botón de la figura que permite poner en el dialogo la hora del instante en la que fue oprimido.

Por otra parte BdaOptFlds presenta nueve valores booleanos representados son Switches.

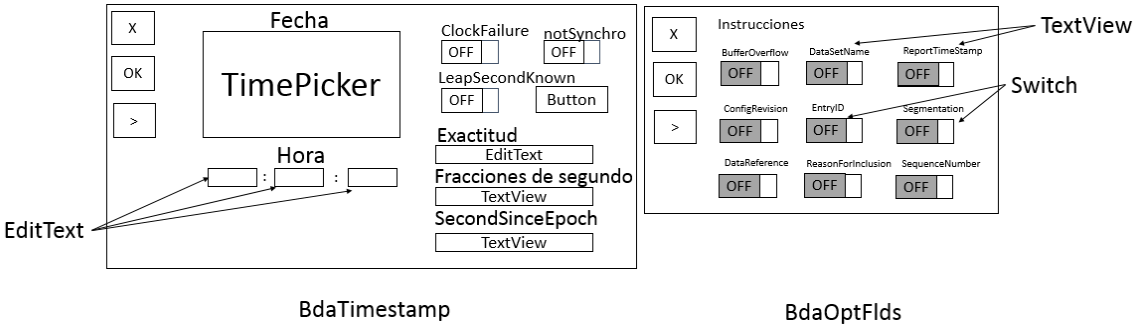


Figura 43. Dialogo para variables de tipo Timestamp y BdaOptFlds

(E) BdaReasonforInclusion, TriggerConditions, BdaTapCommand y BdaQuality

BdaReasonforInclusion y TriggerConditions comparten exactamente los mismos atributos. Son seis de tipo booleano y para representarlos y manipularlos se usan seis Switchs.

Por otra parte BdaTapCommand tiene cuatro posibles valores: HIGHER, LOWER, RESERVED, STOP. Para esto, se han puesto cuatro botones para su representación y manipulación. Al tocar un botón se indica al usuario cuál de los cuatro se oprimió inhabilitando el botón tocado y habilitando el resto.

Finalmente BdaQuality se compone de once valores booleanos y del atributo de Validez que puede tener cuatro valores: GOOD, INVALID, RESERVED y QUESTIONABLE. Se decide usar once switchs y cuatro botones para los cuatro posibles valores de la validez. Al hundir un botón se indica al usuario cuál de los cuatro se oprimió inhabilitando el botón tocado y habilitando el resto.

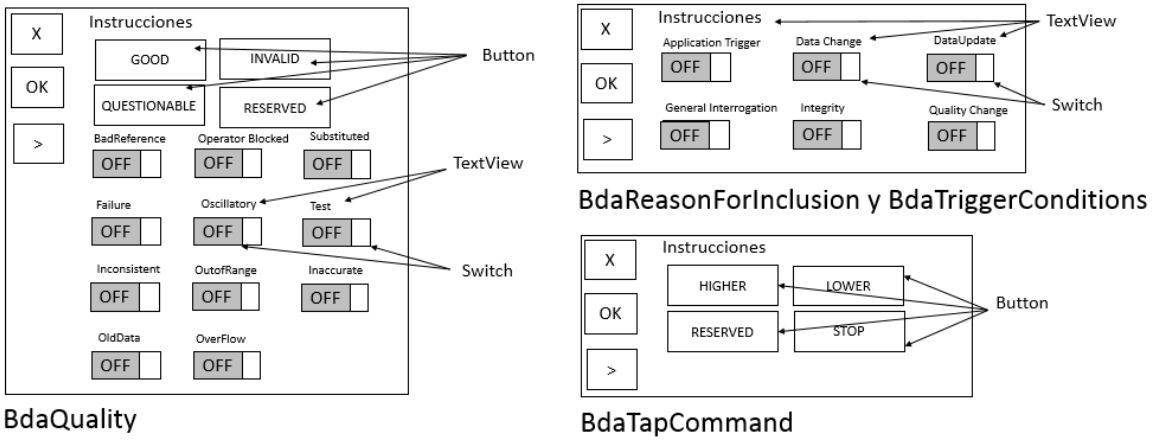


Figura 44. Dialogo para variables de tipo BdaQualiy, BdaReasonforInclusion, BdaTrigerConditions y BdaTapCommand

(F) Lectura y representación de los UIElements del asistente

La lectura de los UIElements usados en los diálogos, así como la representación de datos sobre ellos tienen método que se describen en la tabla 20.

UIElement	Tipo a representar	Representar dato	Leer Dato
Switch	Binario	void setChecked(Boolean valor)	Boolean isChecked()
TextView	Alfanumérico	void setText(String valor)	String getText(String valor)
EditText	Alfanumérico	void setText(String valor)	String getText(String valor)
TimePicker	Año, mes y dia	void updateDate(int ano, int mes, int diaDelMes)	Int getYear(); Int getMonth(); Int getDayofMonth()
Button	Lista de opciones	setEnabled(Boolean seleccion)	Boolean isEnabled()

Tabla 20. Lectura y representación de los UIElments del asistente

(G) Algoritmo para abrir BDAs con el asistente

El asistente se integra al navegador de nodos como se ve en la figura 33 lanzar el asistente implica leer la variable y según el tipo representar todos sus atributos mediante los diálogos del asistente antes de ser mostrado en pantalla. El diagrama de flujo de la figura 45 muestra el procedimiento que se sigue luego de

lanzar el asistente. Se debe recordar que en la variable auxiliar Node queda apuntado el nodo que se visualiza en el asistente.

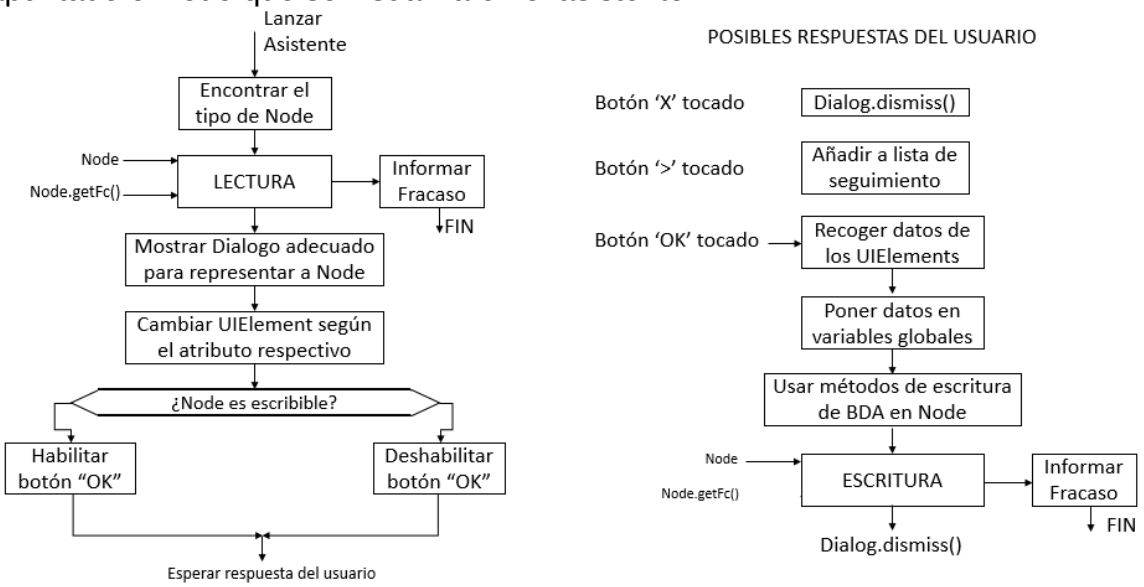


Figura 45. Abrir una variable en el asistente y respuestas del usuario

Si el procedimiento de lectura falla, el fracaso se informa por medio de Toast. Mostrar el Dialogo adecuado según el tipo de Node se hace gracias a que se encuentra primero el tipo de variable y mediante un selector se lanza el dialogo correcto. Se debe leer la variable y poner los atributos adquiridos en los UIElements para representar todos los atributos del dato para que sea legible al usuario.

El botón se escritura estará habilitado sólo si el nodo es escribible. Para saberlo se emplean las siguientes líneas de código.

```
FcModelNode modelNode = (FcModelNode) Node;
Fc constraint = modelNode.getFc();
return constraint != Fc.ST && constraint != Fc.MX && constraint!=null && constraint!=Fc.RP;
```

En la primera línea se realiza un *cast* de la variable para poder saber su Fc. La segunda lo adquiere en una variable llamada *constraint* y finalmente se retorna una operación binaria. La operación significa que el nodo es escribible si su Fc no corresponde a un Status (ST), Medición (MX), Reporte (RP) o Fc nulo de lo contrario no se podrá escribir la variable.

Después de ser visualizada en pantalla la variable, si es escribible, el usuario puede cambiar, mediante los mismo UIElements que se usan para la representación, el valor de la misma. Para esto se debe recoger la información que se ponga sobre el dialogo, cambiar el valor de las referencias de forma local apoyándose en las variables globales para finalmente ejecutar el procedimiento de escritura.

3.7.11 Elementos gráficos auxiliares

En el navegador de nodos habrá dos *TextView* que indiquen el nombre del nodo actual y su dirección URI respectivamente.

Para la indicación del nodo actual simplemente se pondrá en el título el nombre del nodo tocado, o sea el nombre del punto NAV.

La dirección URI se indicará con una cadena de String muy parecida a la que arrojan los objetos de la clase *ObjectReference* y tiene la siguiente forma

LDnombre/LNNombre.DONombre [Fc].DANombre. … .SDANombre

Esto indicará que los elementos mostrados en lista pertenecen al nodo cuyo nombre es el del título ubicado en la dirección del *TextView* inferior.

3.7.12 ORGANIGRAMA

Consiste en un grupo de botones que dibuja el organigrama del IED según el usuario navega por él. La disposición de los botones y la condición para que aparezca según la navegación es la que se muestra en la figura 46.

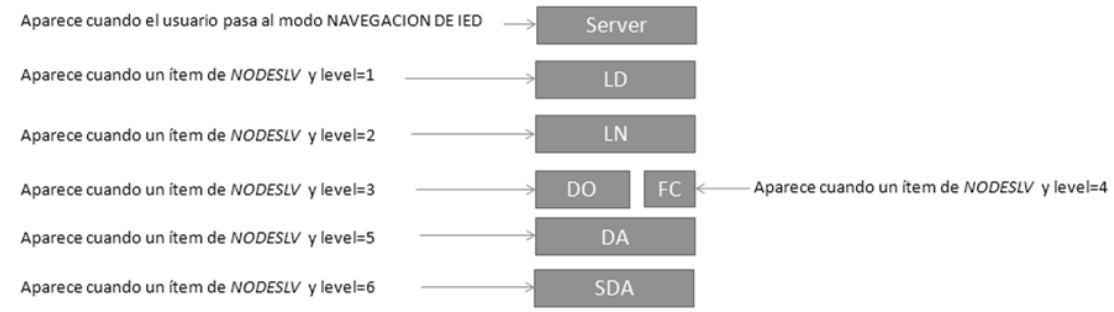


Figura 46. Organigrama y los botones que lo componen

Los botones tienen implementado el *onClickListener* de modo que cuando son tocados el puntero NAV se sitúa el nodo del mapa desapareciendo sus subordinados. Los nombres de los nodos aparecerán en el botón de su nivel respectivo.

La forma como funciona es ejecutando el método del botón atrás "←" tantas veces como se requiera hasta quedar fijado en el nivel que el usuario ha seleccionado mediante el toque a un botón del organigrama

```
Retrocesos=level-levelTocado;
for (int i=0 ; i<Retrocesos){
   NavegadorNodos_Atras();
}
```

Donde *levelTocado* será el nivel del botón del organigrama tocado según la siguiente lista:

- ✓ Botón Server: levelTocado=1
- ✓ Botón LD: levelTocado=2
- ✓ Botón LN: levelTocado=3

- ✓ Botón DO: levelTocado=4
- ✓ Botón FC: levelTocado=5
- ✓ Botón DA: levelTocado=6
- ✓ Botón SDA: levelTocado=7

3.7.13 Visor de RCB

Según lo expuesto en el capítulo sobre los servicios de la norma IEC 61850 que se encuentran en OPEN IEC 61850, los datasets y RCB configurados en un IED pueden ser vistos con el cliente mediante el uso de las clases Dataset y RCB del proyecto abierto.

Se ha decido crear un modo para estos dos entes teniendo en cuenta su estrecha relación.

Para los listar los RCB configurados en un IED se usará el ListView llamado *RCBLV* con ArrayList asociado *ListaRCB* y adaptador *AdapterRCB*.

(A) OBJETO

Al igual que en el navegador de nodos la lista *RCBLV* contendrá objetos de la clase *NodeProfile* descrito en la tabla 5.6.2.1.1. El Adaptador *AdapterRCB* dotará a cada elemento de la lista *RCBLV* de dos TextView organizados en dos renglones mostrando los atributos *Nombre* y *BDAType*, este último usado para informar el Dataset al que pertenecen los RCB listados.

(B) INICIALIZACION

Al seleccionar el modo “DATASETS Y RCB” comenzará la carga de los RCB en la lista *RCBLV*.

Para esto se usa el método de ServerModel getUrcbs() que devuelve un colección de todos los RCB configurados en el script SCL. La inclusión a la lista se hace como lo muestra el diagrama de la figura 47.

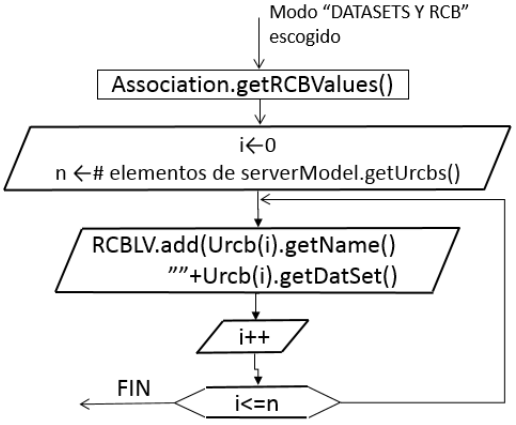


Figura 47. Inicialización de RCB en RCBLV

(C) CONTENIDO

El usuario puede acceder a cualquier RCB listado tocando el ítem gracias a la implementación del listener OnClickListener.

Cada RCB contiene una lista de BDAs con un nombre determinado, estos son:

- ✓ BufTm de tipo BdaInt32U. Mide el tiempo, en milisegundos, de duración de las notificaciones disparadas por las banderas dupd, qchg, dchg en un BRCB. El valor se lee con el método RCB.getBufTm().
- ✓ DatSet de tipo BdaVisibleString. Muestra el nombre del dataset que ha sido afiliado al presente RCB.
- ✓ ConfRev de tipo BdaInt32U. Cuenta las veces que un dataset afiliado a un RCB ha sido cambiado. Se lee mediante el método RCB.getConfRev().
- ✓ IntgPd de tipo BdaInt32U. Se lee mediante el método RCB.getIntgPd().
- ✓ OptFlds de tipo BdaOptFlds. Representa los campos opcionales que este RCB envía a los clientes afiliados al reporte. Este valor se lee mediante getOptFlds().
- ✓ RptEna de tipo BdaBoolean. Representa si los reportes para este RCB están habilitados o no. Se lee mediante getRptEna().
- ✓ RptId de tipo BdaVisibleString. Indica el ID del reporte emitido por el RCB. Se lee mediante el método getRptId().
- ✓ SqNum de tipo BdaInt8U. Indica el número de secuencia para todos los BRCB cuyo parámetro RptEna es verdadero.
- ✓ Resv de tipo BdaBoolean. Indica si el URCB está reservado por otro cliente.

Lo anterior significa que el organigrama del RCB sólo tiene tres posibles niveles según el organigrama de la figura 48:

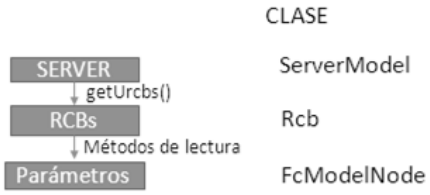


Figura 48. Organigrama de RCB

Estos parámetros pueden ser solamente leídos mediante el asistente de lectura y escritura. En el proyecto se decide excluir el servicio de reportes para el cliente IEC 61850 dadas las limitaciones de OPEN IEC 61850.

3.7.14 Navegador de datasets

Para los Datasets se usará el ListView llamado DATASETSLV con ArrayList asociado *ListaDATASETS* y adaptador *AdapterDatasets*.

(A) Objeto

Al igual que en el navegador de nodos y el visor de RCB, la lista *DatasetsLV* contendrá objetos de la clase *NodeProfile* descrito en la tabla 5.6.2.1.1. El Adaptador *AdapterDataset* dotará a cada elemento de la lista DATASET*LV* de dos TextView organizados en dos renglones mostrando los atributos *Nombre* y *BDAType*, este último usado para múltiples propósitos según el nivel:

✓ Nivel Server: Se informa la ruta donde está inscrito cada Dataset listado.
✓ Nivel Dataset: Se informa el FC de DA listado en el Dataset.
✓ Nivel DA e inferiores: Se informa el tipo de dato BDA

(B) Inicialización

Para el proceso de inicialización se usa un procedimiento parecido al módulo de adición a datasets pero esta vez se incluyen tanto los borrables (creados por los clientes) como los no borrables (creados en el script SCL).

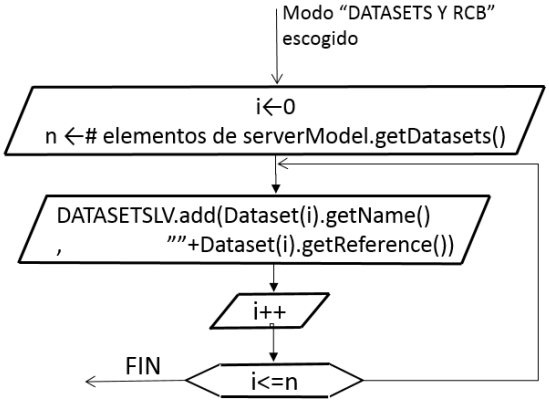


Figura 49. Inicialización de datasets en DATASETLV

Una vez inicializados el usuario podrá navegar por los nodos Fc que integren cada Dataset.

(C) Navegación

La navegación en el DATASETLV es similar al del NODESLV. La diferencia es que no es necesario lidiar con Fc o nombres de DO repetidos ya que los datasets contendrán DA e inferiores. En el objeto ServerModel los Dataset se organizan como se muestra en la figura 50

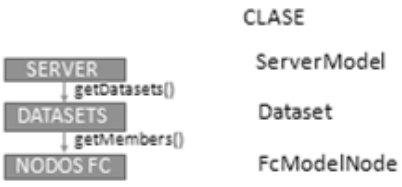


Figura 50. Organigrama de datasets

Cuando se ha llegado al nivel de NODOS FC se navega de la misma forma como en el navegador de nodos. El diagrama de flujo para ejecutar la navegación por estos niveles es el mostrado en la figura 51:

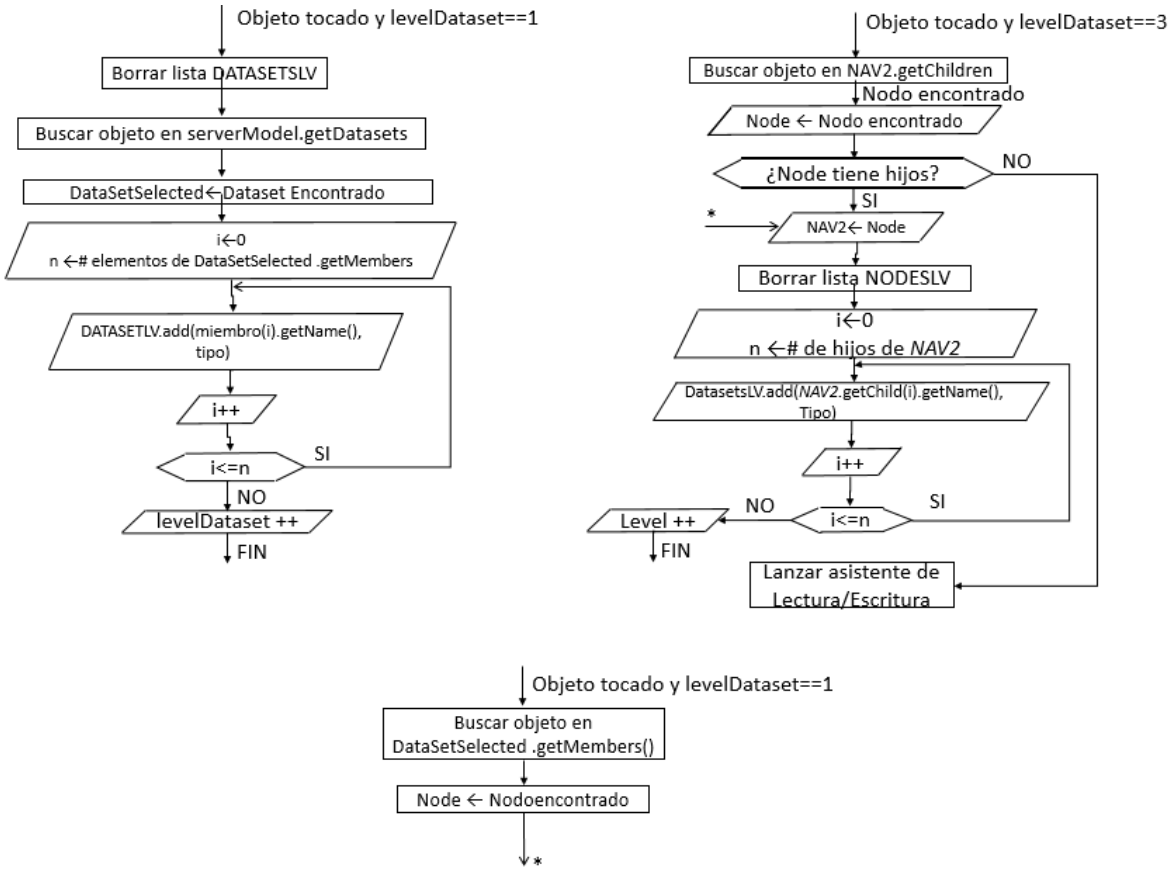


Figura 51. Algoritmo de navegación en Datasets

El algoritmo se desarrolla con tres punteros: El primero es el puntero de nivel que es un número entero indicando en qué nivel se encuentra el usuario en el DATASETLV siendo el inicial el número cero correspondiente al server, es llamado *levelDataset*. El puntero *DataSetSelected* (objeto de la clase Dataset) que indicará cuál de los datasets de la lista *ServerModel.getDatasets* ha sido seleccionado y responsable de apuntar a cual Dataset ha accedido el usuario.
El puntero *NAV2* (objeto de la clase FcModelNode) para apuntar a nodos del nivel dos hacia abajo.
En el diagrama se observa que luego que el usuario seleccione un Dataset de la lista inicial, se cargarán sus miembros en la lista, presentes en la colección *DatasetSelected.getMembers()*. Los miembros de un dataset corresponden a nodos Fc (Nivel NODOS FC) y se podrá bajar a niveles inferiores del organigrama hasta que uno de los nodos tocados no tenga hijos, esto significa que corresponde a un BDA por lo cual se debe iniciar el asistente de lectura y escritura.

(D) Retroceso

El botón de retroceso para subir a niveles en el organigrama está controlado mediante el algoritmo de la figura 52.

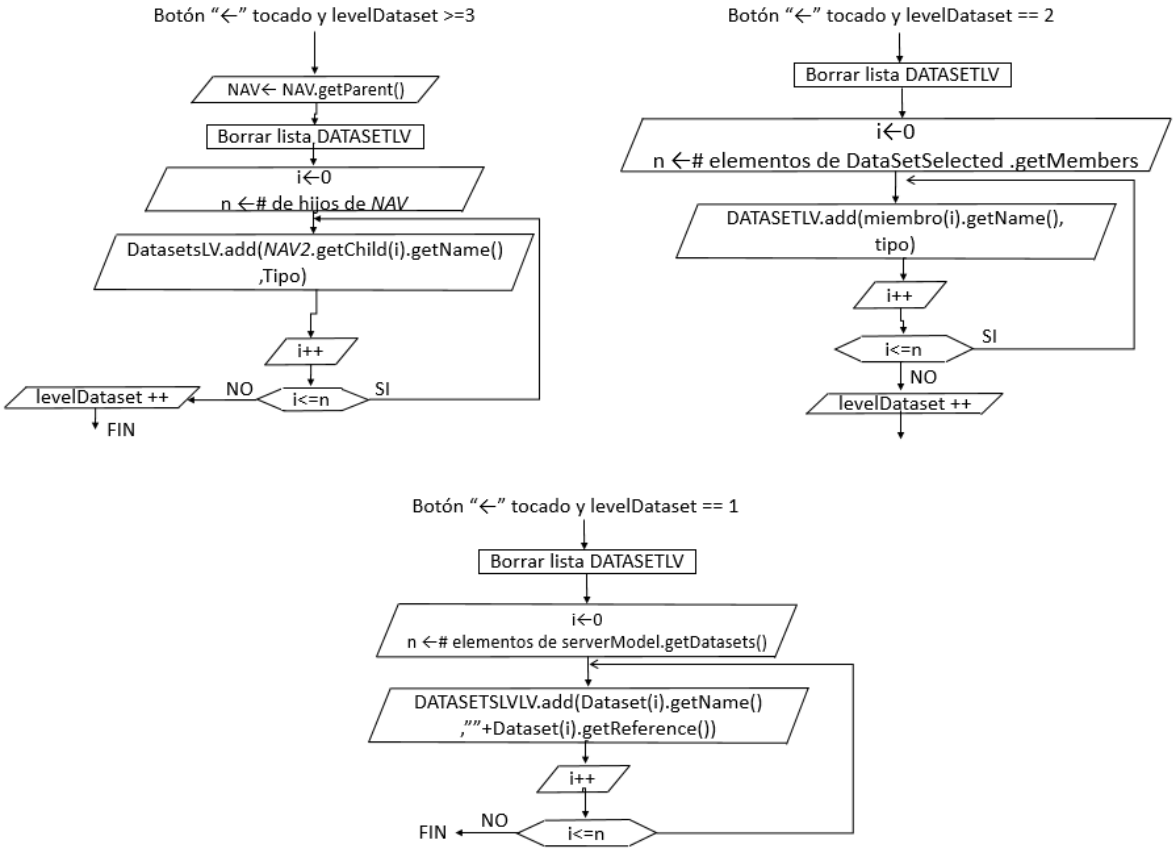


Figura 52. Algoritmo de navegación hacia atrás en Datasets

El algoritmo prevé la existencia de los niveles del organigrama de Datasets y la diferencia de clases entre ellos.

(E) ELIMACION DE DATASET

Para el proceso de eliminación de datasets se usa el listener onLongClickListener de modo que cuando el usuario toque prolongadamente un Dataset de DATASETSLV este sea eliminado si es que es borrable. De otro modo se le informará al usuario del fracaso de la operación con un Toast.

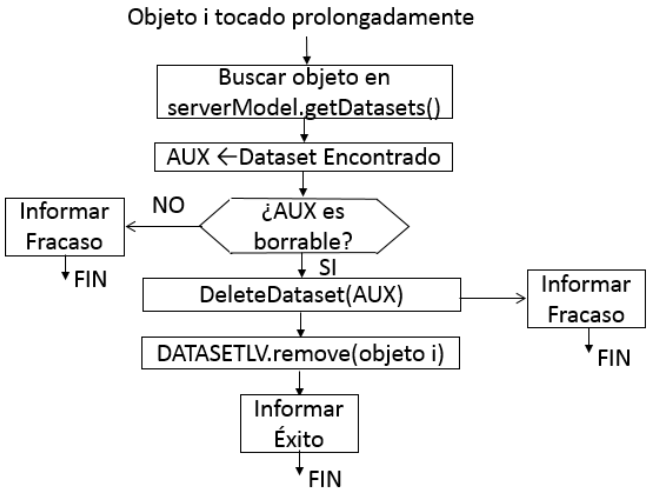


Figura 53. Algoritmo de eliminación en Datasets

Para eliminar el dataset, entonces, se usa el servicio ACSI DeleteDataset y de tener éxito, se elimina de la lista y del servidor y se le informará el éxito al usuario mediante un Toast y la eliminación del ítem de la lista.

3.7.15 Panel de seguimiento

El panel de seguimiento consiste en un modo en el que el usuario podrá visualizar la lectura de hasta diez variables de cualquier tipo de dato BDA. Las diez variables son organizadas en un ListView que presente en todos los modos. Mediante el hilo de recepción continua (Se describe en detalle más adelante) se actualiza el valor de las variables en lista cada determinado tiempo.

El panel de seguimiento está alojado en el Layout auxiliar reemplazando el menú por lo que se podrá visualizar la lista de variables en seguimiento en cualquier modo de la aplicación. La forma como está dispuesto el panel de seguimiento se puede observar en la figura 54.

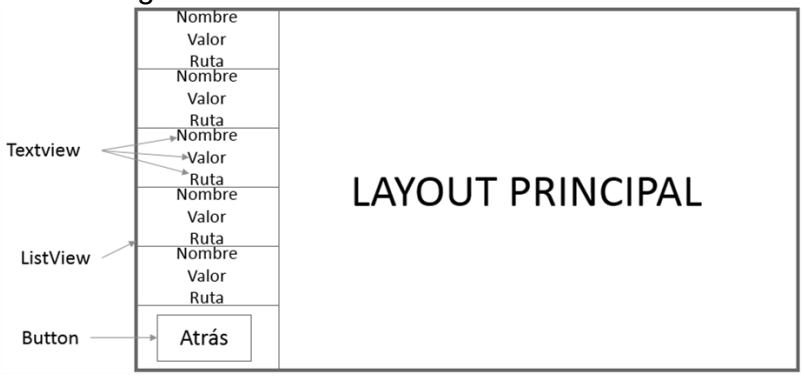


Figura 54. Composición gráfica del panel de seguimiento

Se decide crear una clase llamada ProbeProfile cuyos objetos sean los que se encuentren en la lista PROBESLV. Sus atributos y métodos se encuentran en la figura 21:

ATRIBUTO	TIPO	Escritura	Lectura
nombre	String	void setName(String nombre)	String getName()
valor	String	void setValor(String valor)	String getValor()
raiz	String	void setRoot(String root)	String getRoot()
Inicialización			
void setAll(String n, String fc, String type, String path, long id)			

Tabla 21. Atributos y métodos de los objetos de la clase ProbeProfile

El ArrayList asociado es llamado PROBESList y su adaptador tiene el nombre AdapterProbes. Este último dotará a los elementos de PROBESLV de tres TextView organizados en tres renglones. Cada TextView representará respectivamente todos los atributos de los objetos de ProbeProfile almacenados.
Este modo pensado para recibir cada determinado tiempo, definido por el usuario, datos actualizados del servidor. El encargado de controlar el flujo de datos de entrada e inclusión de variables a seguir es el hilo de recepción continua que se describe en seguida.

3.7.16 HILO DE RECEPCION CONTINUA

El hilo de recepción continua es una clase que permite actualizar el valor de una variable cada cierta cantidad de segundos especificada en la variable global *TiempodeMuestreo* para ser mostrada en la pantalla del dispositivo móvil.
Consiste en hacer el procedimiento de lectura en un hilo a parte del hilo principal preguntando por el valor actual al servidor de todas las variables que componen la lista de seguimiento.
El hilo es un bucle de iteraciones indefinidas pues está pensado para recibir un flujo de datos constante para que sea mostrado en la aplicación de forma dinámica, conforme lleguen los datos al cliente.
El algoritmo consta únicamente de doInBackground y OnProgressUpdate de la clase AsyncTask. En la figura 55 se muestra la tarea que se lleva en segundo plano y que compone el método doInBackground.

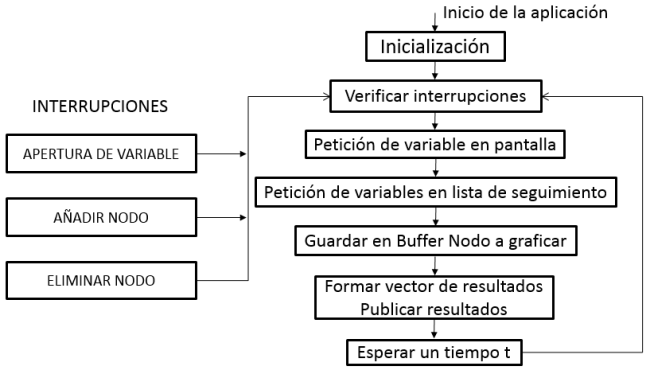


Figura 55. Procedimiento del hilo de recepción continua

Como se observa en el diagrama una de las interrupciones es abrir una variable en el asistente. Se trata de un complemento al asistente de lectura y escritura usando como motor el hilo de recepción continúa para observar el movimiento de la variable en el asistente y se expondrá en detalle más adelante.

(A) Inicialización

Se requiere de un *ArrayList* que permita albergar objetos de *FcModelNode* (Variable llamada *NodetoRead*) para la lectura en conjunto de todos los nodos puestos en esta lista mediante el bloque “Petición de variables en lista de seguimiento”. Inicialmente estará vacía y se podrán añadir elementos mediante el asistente de lectura/escritura de variables con el botón de seguimiento de nodo lo que se representa en el diagrama con la interrupción “Añadir Nodo”. Los objetos de la clase *FcModelNode* de esta lista deben concordar con el *Listview* que contiene objetos *ProbeProfile* que a su vez compone el panel de seguimiento.
La lista de variables a seguir se almacena en el ArrayList ListaPROBES correspondiente al ListView PROBESLV y con adaptador adapterPROBES. Este último dota a cada ítem de PROBESLV con tres renglones para los tres atributos del objeto ProbeProfile.

La lista de variables en seguimiento cuenta con un máximo de hasta diez variables y se usa una variable de tipo entero llamado *NumeroProbes* que indica el número actual de variables en seguimiento.
Mediante *OnProgressUpdate* desde la clase *ReceiverTask* se posibilita actualizar los elementos gráficos, sin interferir con la ejecución del programa, con los datos más recientes por lo que se requiere un vector de *String* llamado *Results* para almacenar todos los valores de las variables en *NodetoRead* en la publicación de los resultados en cada bucle mediante el método onPublishProgess().

(B) Interrupciones

Aunque las interrupciones cambian el valor de variables dentro de la clase *DataReceiver* al instante, el cambio tendrá efecto al siguiente ciclo del bucle debido al tiempo de espera entre cada ciclo.
Cuando se cierra el asistente, se cambia de nuevo la variable a *null*, para inhabilitar nuevamente el bloque “Petición de Variable en pantalla”.

(i) Añadir Elemento

Todas las variables que quieran ingresar al panel de seguimiento deben ser añadidas por el usuario desde el asistente de escritura/lectura. Según la figura 39 el botón “>” será el que permita añadir la variable visualizada en el asistente al panel de seguimiento. El método de adición es una interrupción al hilo de recepción continua y su procedimiento es el mostrado en la figura 56:

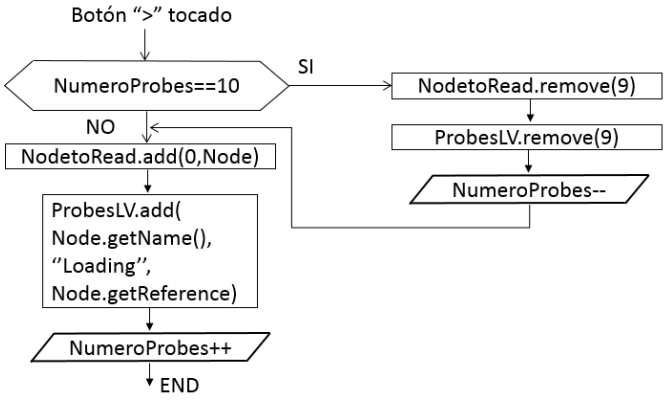


Figura 56. Añadir elemento a la lista de seguimiento

Las listas NodetoRead y ProbesLV están en modo FIFO: La sentencia NodetoRead.add(0,Node) indica que se introduce el nodo abierto en el asistente de lectura/escritura a la lista en la primera posición desplazando los otros elementos al índice inmediatamente superior y cuando se completan los 10 nodos en lista se remueve el último y se añade el entrante en la primera posición.

(ii) Eliminar Elemento

PROBESLV tiene implementado el listener onLongItemClick que eliminará el ítem que se toque prolongadamente siguiendo el diagrama de flujo de la figura 57.

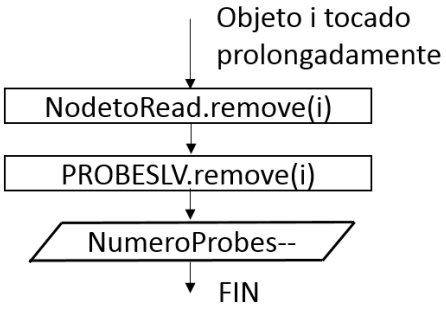


Figura 57. Eliminar elemento a la lista de seguimiento

Eventualmente, cuando se borren todos los nodos de *NodetoRead*, se inhabilitará el bloque de “Petición de variables en lista de seguimiento”

(iii) Abrir variable en el asistente

Esta interrupción ocurre cuando el usuario abre una variable en el asistente de lectura/escritura. Esto habilita el bloque “Petición de Variable en Pantalla”.

(C) PROCEDIMIENTO

(i) Bloque de petición de variables en seguimiento

Se hace un bucle con un número de iteraciones igual al número de elementos cargados en el *ArrayList NodetoRead* para leer secuencialmente los valores de las variables almacenadas. Todos los valores de los elementos son guardados en el vector *Results.*

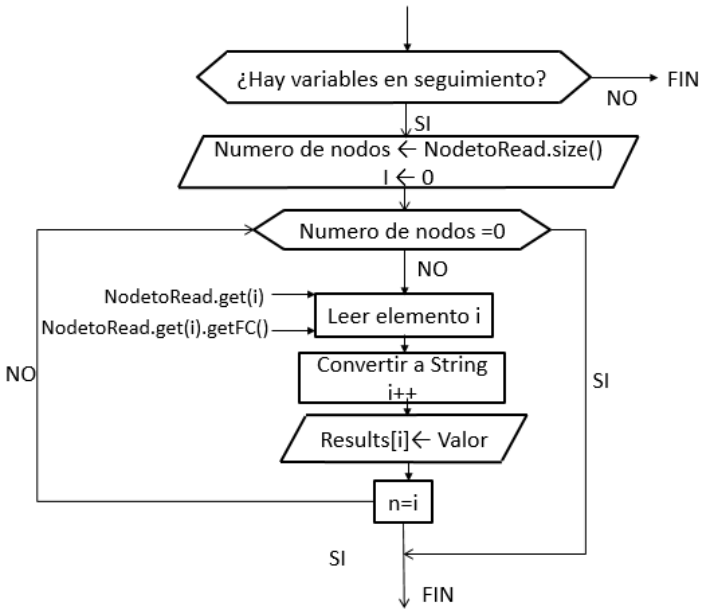


Figura 58. Algoritmo de lectura de lista de variables bajo seguimiento

(ii) Petición de variable en pantalla

Este bloque es un complemento al asistente de lectura y escritura, no tiene que ver con el modo de seguimiento de variables pero hace parte del hilo de recepción continua. Consiste en la petición del valor de la variable que ha sido abierta en asistente de conexión, donde el View involucrado es el que se encuentra dentro del diálogo y que muestra el valor de la variable.

Se requiere un *FcModelNode* aparte para realizar seguimiento en tiempo real a la variable que ha sido abierta en el asistente (llamada *NodoEnPantalla*). Esta variable estará inicialmente nula y para su actualización en tiempo real se debe usar el *View* del asistente que sirve de interfaz para su visualización. Por ejemplo, los tipos numéricos de datos, presentan *EditText* para la representación del valor de la variable. Para esto se decide instanciar un puntero que referencie el UIElement de representación (llamado ViewenPantalla) con el fin de poder manipularlo y modificarlo desde el AsyncTask con el método onPublishProgress.

En el caso del tipo *BdaTimestamp* se ha decido dar tres punteros correspondientes a los TextView de horas, minutos y segundos. (Llamados en el AsyncTask *ViewHoras, ViewMinutos, ViewSegundo*).

La inicialización de los *Views* con valor nulo se realiza para poder comparar en cada iteración del diagrama de flujo de la figura 55 si los punteros fueron asignados externamente.

El asistente se encarga de instanciar la variable *NodeenPantalla* para que corresponda a la variable abierta. Se lee, su valor se transforma a String. Esto se puede observar en el diagrama de la figura 59.

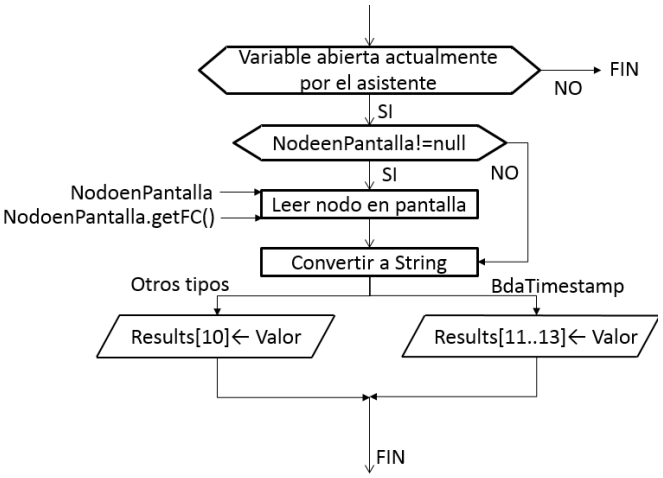


Figura 59. Algoritmo de lectura de lista de variables bajo seguimiento

No todos los tipos de datos tienen implementada esta característica debido a que no todos cambian con la misma frecuencia como lo hacen los tipos numéricos. Los siguientes son los tipos de datos con actualización automática al ser abiertos en el asistente de lectura y escritura: BdaFloat32, BdaFloat64, BdaInt16, BdaInt16U, BdaInt32, BdaInt32U, BdaInt64, BdaInt8, BdaInt8U, BdaTimestamp.

(iii) Guardar en buffer nodo a graficar

Este bloque es habilitado cuando el usuario toca una variable numérica dentro del panel de seguimiento**.** PROBESLV tiene implementado el listener onItemClick que abrirá el modo de PANEL DE GRAFICACIÓN. Cuando se activa este modo, el bloque “Guardar en Buffer Nodo a graficar” de la figura 55 guardará en un buffer llamado BufferVar del tipo ArrayList<Float> los valores que se vallan recibiendo para esta variable. El procedimiento puede verse en la figura 60.

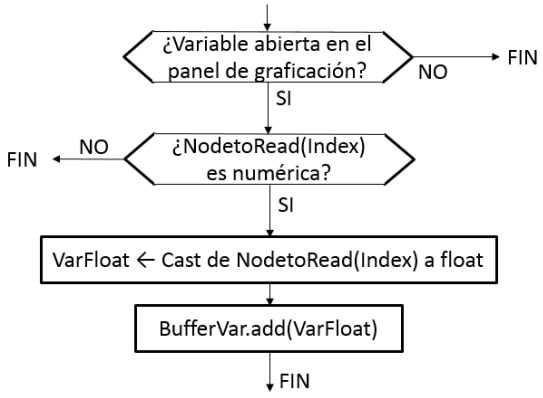


Figura 60. Buffer de puntos para graficar en el panel de graficación

(iv) Formar vector de resultados y publicación

El Vector *Results* tiene 13 posiciones dispuestas de la siguiente forma:

Results[0..9] : Almacenar los resultados convertidos en String que resultan del bloque ‘Petición de variables en seguimiento’.
Results[10] : Almacenar el resultado convertido en String que resulta del bloque ‘Petición de variable en pantalla’.
Results[11..13] : Almacenar el resultado convertido en String que resulta del bloque ‘Petición de variable en pantalla’ si la variable observada en el asistente es de tipo *BdaTimestamp*.
Luego de formar el vector se publica usando el método *publishProgress* de AsyncTask. Esto llama el método *OnProgressUpdate que* recibe el vector publicado para luego ser mostrado en pantalla. Se actualiza el valor de la variable del asistente y de las variables en el panel de seguimiento como se ilustra en la figura 61.

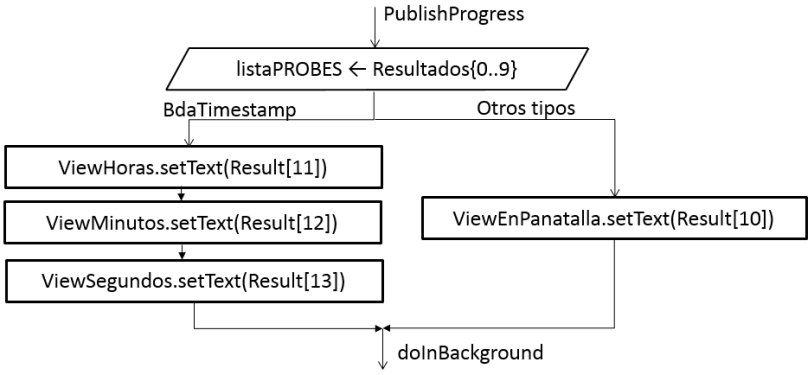


Figura 61. Publicación de resultados para actualizar los UIElements

En el que se tiene en cuenta la disposición del vector *Results* para que estos valores sean representados en el UIElement usado en el asistente de lectura/escritura.

3.7.17 Panel de graficación

Este panel mostrará el movimiento de la variable conforme vallan llegando los datos al cliente dibujando los puntos en un plano cartesiano mediante la clase de Android SurfaceView. Presenta la forma de la figura 62.

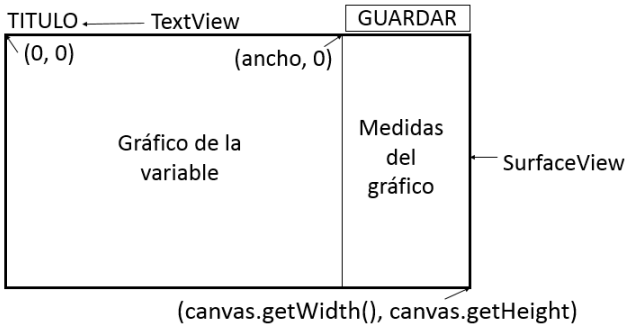


Figura 62. Composición gráfica del panel de graficación

Esta forma corresponde a un Layout perteneciente al ViewFlipper principal y sólo se mostrará cuando el usuario toque un ítem de variable numérica del panel de seguimiento.
El motor de redibujado del panel es un hilo de la clase Thread responsable de efectuar un cambio en el SurfaceView cada determinado tiempo. Este tiempo será el mismo *tiempoMuestreo* ya que como se ve en la figura 55 no existe forma que se reciba datos en instantes intermedios.
Cabe anotar que el panel sólo esbozará variables de tipo numérico y que el algoritmo guardará los datos a partir de que el usuario haya tocado el nodo numérico en el panel de seguimiento.
El hilo de redibujado tiene una inicialización para poner líneas horizontales y verticales, dibujar los cajones de medida del gráfico y el fondo para que quedar listo gráficamente para cuando comience a recibir los datos.
Luego de la inicialización el hilo quedará en espera hasta que el usuario seleccione una variable numérica del panel de seguimiento, en cada iteración el algoritmo ejecuta los comandos enunciados en la figura 63:

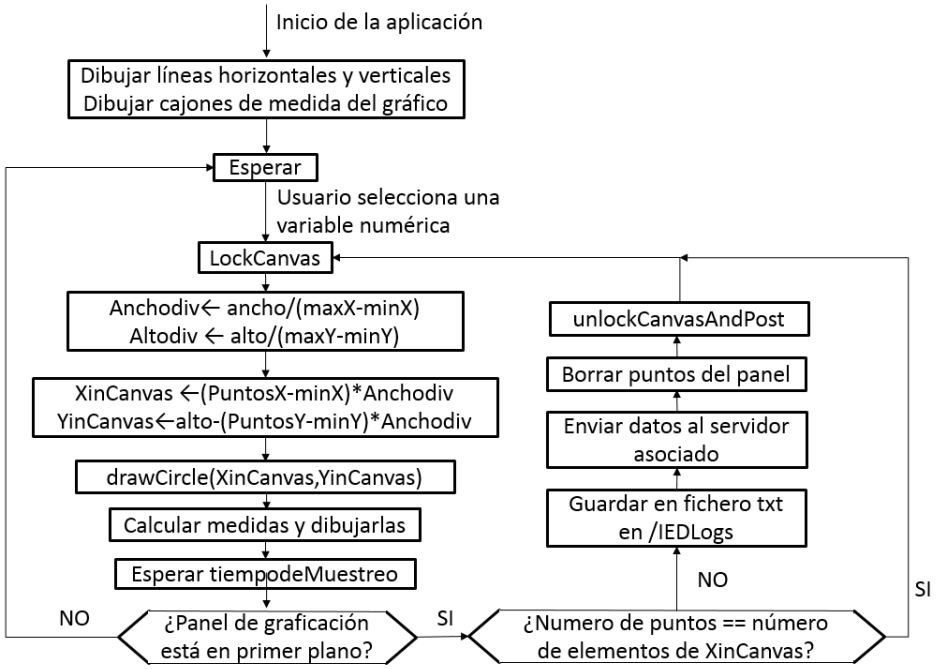


Figura 63. Motor de redibujado, creación y envío de logs al servidor

Donde *PuntosX* y *PuntosY* son puntos coordenados de tiempo y valor de la variable. *PuntosY* sale de *BufferVar*. *PuntosX* sale del tiempo de inicio toma de la muestra y a partir de este tiempo los siguientes tiempos estarán espaciados en el plano cada *tiempoMuestreo* segundos.

Se puede observar que en cada iteración se borran los puntos y se vuelven a dibujar con un tiempo de espera intermedio además que el graficó se escala automáticamente tanto en X como Y. Mediante unas etiquetas se informará al usuario algunos valores de los ejes.
El redibujado ocurre tan rápido como el dispositivo pueda procesar los bloques después de la espera hasta llegar de nuevo al dibujado.
Se podría pensar que la animación sufrirá de *flashing* pero la mayor ventaja de la clase SurfaceView es justamente la eliminación de este fenómeno para que el desarrollador no tenga que implementar técnicas como la de doble buffer para eliminarlo. Esta clase requiere que se haga una copia de lo que se ha dibujado actualmente y ponerla en un canvas, el usuario dibuja sobre él, modificándolo, y cuando termine llama el método unlockCanvasAndPost que guardará y visualizará los cambios hechos a la región donde se colocan los puntos.
Asimismo en cada iteración se refrescan las medidas del grafico que gracias al método drawString(String texto, float x, float y) permite dibujar en un Canvas una cadena de texto. Con esta herramienta se publicará en cada iteración medidas comunes de gráficos. Estas son:

- ✓ Valor Minimo: El minimo de los datos mostrados en el panel de graficación. Coincide con el límite inferior del grafico dada la escala automática del Panel
- ✓ Valor Máximo: El máximo de los datos mostrados en el panel de graficación. Coincide con el límite superior del gráfico.
- ✓ Valor Promedio
- ✓ Valor Actual
- ✓ Tiempo de inicio de la muestra: La hora y fecha en la que el usuario decidió abrir la variable en el panel de graficación.

El panel de graficación está programado para guardar un log cuando el buffer llege a -100- datos o cuando el usuario oprima el botón de guardado. El formato de este fichero es el siguiente:

```
t, Valor 1
t+tiempoMuestreo, Valor 2
...
t+n*tiempoMuestro, Valor n
```

La aplicación guardará el log en una carpeta llamada IEDLog ubicada en la memoria principal del dispositivo móvil. En el nombre del archivo log se tendrá información de la ruta de la variable y la fecha de inicio de la muestra. El archivo tendrá extensión *.txt*.

3.7.18 Envío de datos a servidor

Se realiza gracias a las clases HttpClient y HttpPost. Esta última clase contiene como parámetro de entrada la dirección URL del servidor la cual es brindada por el usuario en el modo de configuración del cliente para hacer un envío mediante POST. Se crea un buffer de salida hacia el servidor mediante la clase NameValuePair en la que se pueden emitir pares coordenados teniendo en cuenta que si se pone como *Nombre* en cada par el valor del tiempo en el instante que fue tomado, este tiempo será único. El diagrama es el de la figura 64.

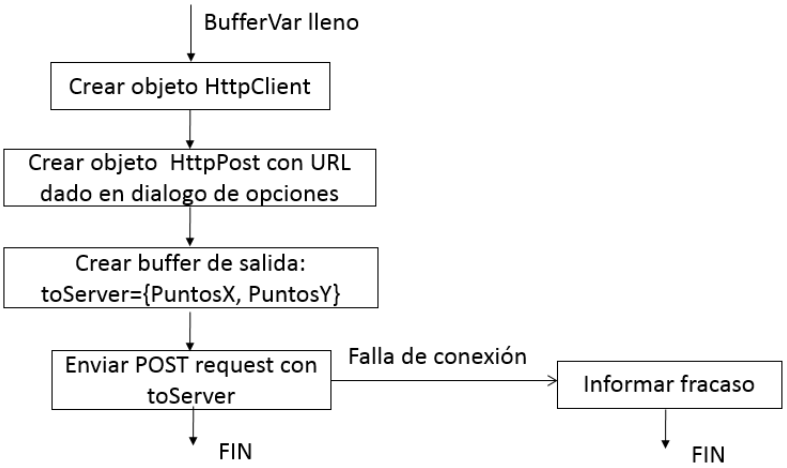


Figura 64. Algoritmo envío de POST request luego del buffer lleno

(A) Script para gestionar el POST request en el lado servidor

Es un sencillo script que recibirá los datos en el registro POST de forma automática y creará un archivo con el nombre especificado en el mensaje.

El procedimiento que realiza es el de recepción de pares nombre-valor que emite el módulo de envío al servidor. El mensaje enviado desde la aplicación móvil debe tener por obligación un par de la forma: “Nombre : *NombredelArchivo.txt*” con el fin de que desde la aplicación se nombre el archivo que el script en PHP creará en la carpeta pública del servidor.

El script es el siguiente:

```
<?php
   global $filename;
   $filename=$_POST['Name'];
   foreach ($_POST as $key => $value) {
   if($key!=="Name")
   file_put_contents($filename,"$key,$value"."\r\n",FILE_APPEND);
   }
?>
```

Se resalta el hecho de que PHP facilita la lectura de los pares nombre-valor. La escritura línea por línea del archivo con los valores tomados se realiza gracias al bucle y se compara en cada iteración que no se guarde ninguna línea cuyo nombre corresponde a “Nombre”.

4. RESULTADOS

En éste capítulo se realizan pruebas a la aplicación desarrollada probando primero los módulos principales alusivos a los objetivos planteados en la tesis de grado (mapeo, escritura y lectura de nodos). Se examina cualquier contratiempo y mientras se usa la aplicación así como el uso de los recursos del dispositivo. Se usa una aplicación Android llamada CPU monitor para ver el consumo de RAM y porcentaje de CPU usado por la aplicación. Asi mismo se miden tiempos tanto en la aplicación Android como en el servidor por medio de estampas de tiempo al inicio y final de los métodos o módulos.

Los módulos se prueban en los siguientes dispositivos móviles:

Hardware Usado	Samsung Galaxy Tab 2 7.0 Acer Iconia B1-A71
Sistema Operativo del cliente	**Samsung Galaxy Tab 2 7.0:** Android 4.4.2 KitKat **Acer Iconia B1-A71:** Android 4.2.2 Jelly Bean
Características del cliente	**Samsung Galaxy Tab 2 7.0:** Procesador doble núcleo 1GHz, 1GB RAM, 1024x530 resolución efectiva **Acer Iconia B1-A71:** Procesador doble núcleo 1.2GHz, 512MB RAM, 1024x530 resolución efectiva

La resolución efectiva se refiere a la porción de la pantalla que ocupa la aplicación. El resto lo ocupan las barras de notificación y navegación propias de Android.

La red 61850 usada a lo largo del proyecto para realizar pruebas de funcionamiento se ve en la figura 5.

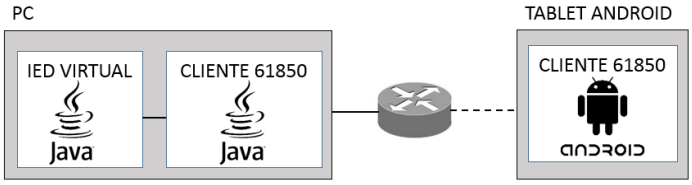


Figura 65. Topología de la red virtual de pruebas

La red está compuesta por un Router Access Point y dos sistemas finales. El PC hace las veces de IED mediante un servidor realizado en java en el proyecto abierto OPEN IEC 61850. El PC a su vez corre un cliente conectado al anterior servidor mediante la dirección loopback con el que se puede conectar al IED virtual.

EL Router AP enlaza el PC con una Tablet Android que se conecta mediante el uso del protocolo IEEE 802.11 (WI-FI). Dentro de la Tablet se encuentra un cliente 61850 que corre sobre el sistema operativo Android (Tema central del proyecto).

4.1. MÓDULO DE CONEXIÓN EN IED SIMULADO

Esta prueba a la aplicación valida el objetivo de poder establecer una conexión con el IED.

Condiciones previas	Un IED virtual OPEN IEC 61850 escuchando peticiones de clientes. Aplicaciones recién instaladas
Objetivo	Lograr conexión con el IED Virtual
Procedimiento	1. Crear un perfil virtual de IED con IP y puerto correctos 2. Configurar opciones de la aplicación por defecto 3. Lanzar el módulo de conexión
Resultado	En ambos dispositivos móviles la conexión es satisfactoria e informada mediante los módulos de información
Contratiempos	**Acer Iconia B1-A71:** algunas animaciones usadas en la aplicación se ralentizan.
Desempeño	30.5136 MB de RAM, 92% máximo en el uso de la CPU, 6% de uso promedio minimizado
Comentarios	El tiempo de conexión al IED Virtual es en promedio de 1.6 segundos. El uso de CPU es elevado y se debe a que Android pausa todas las demás aplicaciones mientras están minimizadas dándole todos los recursos a la aplicación en primer plano. Cuando se minimiza este porcentaje cae drásticamente.
Punto de vista de IEC 61850	Se logra una asociación cliente-IED Satisfactorio uso de los servicios ACSI *GetDataDirectory* y *GetDataDefinition y GetAllDataValues* Contacto en red con el objeto ACSI Server

4.2 MÓDULO DE NAVEGACIÓN DE NODOS

Esta prueba corresponde a la validación del mapeo e identificación de nodos del IED.

Condiciones previas	El dispositivo móvil está conectado al IED Virtual
Objetivo	1. Ir al modo Navegador de IED. 2. Navegar a través del mapa jerárquico del IED, del nivel más alto al más bajo del mapa y viceversa. 3. Tener una guía en la navegación mediante el organigrama, el título del nodo y la dirección URI
Procedimiento	1. Pulsar los ítems mostrados por el ListView 2. Verificar funcionamiento
Resultado	En ambos dispositivos móviles el navegador de nodos funciona correctamente logrando navegar desde el primero hasta el último nivel.
Contratiempos	**Acer Iconia B1-A71:** algunas animaciones usadas en la aplicación se ralentizan.
Desempeño	31.152 MB de RAM, 93% máximo en el uso de la CPU, 6% de

	uso promedio minimizado.
Comentarios	Verificación de la representación virtual de los objetos ACSI recibidos en el método retreiveModel.
Punto de vista de IEC 61850	Contacto, identificación y mapeo de los objetos ACSI SERVER, LogicalDevice, LogicalNode, DataObject y DataAttribute. Verificación y exploración de las relaciones de herencia e Identificación de los tipos de datos.

Figura 66. Captura del modo navegador de IED

4.3. ASISTENTE DE LECTURA Y ESCRITURA

Condiciones previas	El dispositivo móvil está conectado al IED Virtual y en pantalla se encuentra un nodo que no tiene hijos, es decir, un nodo BDA dentro del modo de navegador de IED. El hilo de recepción continua se encuentra con un tiempo de muestreo de un segundo.
Objetivo	1. Verificar que se logre la visualización del tipo de dato requerido en el asistente. 2. Lograr leer distintos tipos de dato según el BDA tocado en la GUI. 3. Debe reconocer si el nodo tocado es escribible. Si no lo es debe estar inhabilitado el botón “OK” en el asistente. 4. Volver a la interfaz principal cerrando el asistente.
Procedimiento	1. Tocar un ítem que represente un BDA. Los BDA bajo pruebas son: BdaCheck, BdaQuality, BdaTimestamp, BdaBoolean, BdaInt8, BdaInt8U, BdaInt16, BdaInt16U, BdaInt32, BdaInt32U, BdaInt64, BdaFloat32, BdaFloat64, BdaVisibleString (con longitud 32, 64 y 255), BdaUnicodeString, BdaOctectString, BdaDoubleBitPos.

	2. Observar el botón “OK” que dirá si el nodo es no escribible
Resultado	En ambos dispositivos móviles la conexión es satisfactoria e informada mediante los módulos de información. Se reciben correctamente los datos solicitados en el proceso de lectura previo.
Contratiempos	Para las variables de sólo lectura, el primer dato mostrado contiene la palabra “Cargando” transitoriamente hasta que el tiempo de retardo entre iteraciones del hilo de recepción continua se agote y actualice este campo en el vector de resultados.
Desempeño	31.517 MB de RAM, 91% en el uso de la CPU para variable de lectura y escritura.
Comentarios	Recepción de datos en el proceso de lectura y su correspondiente representación virtual. Funcionamiento correcto del hilo de recepción continua.
Punto de vista de IEC 61850	Satisfactorio uso del servicio ACSI *GetDataValues.*

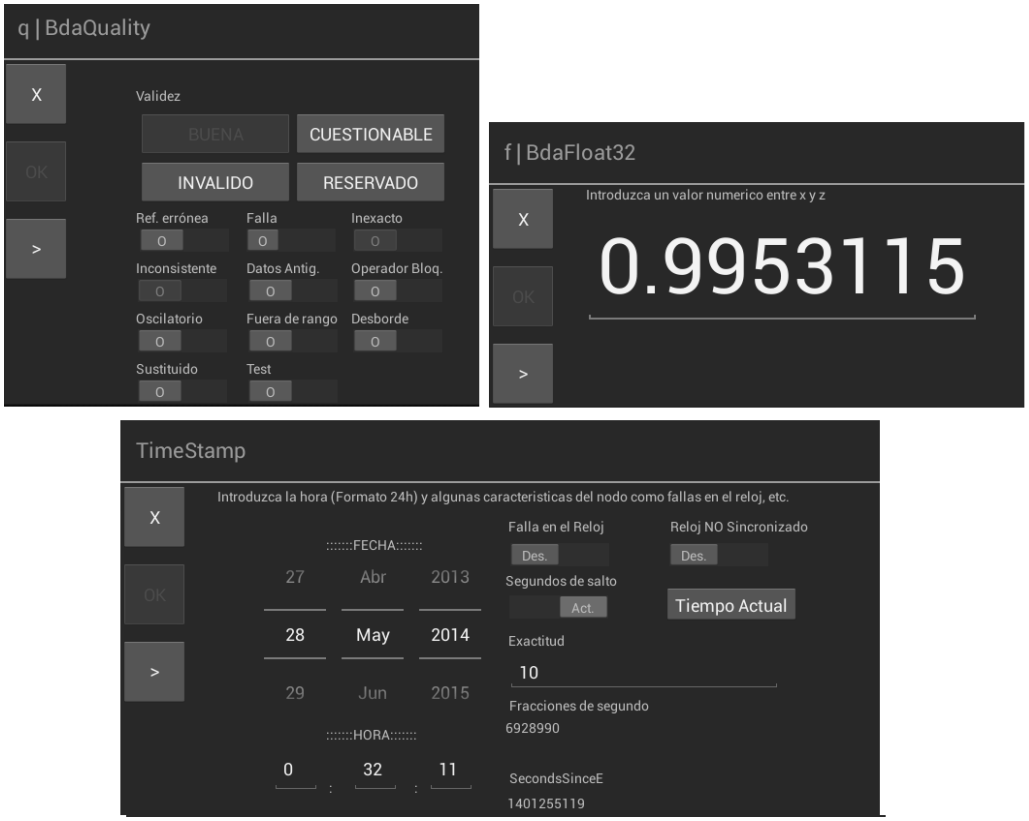


Figura 67. Ejemplos de algunos BDA en el asistente

4.4. ESCRITURA DE VARIABLES

Condiciones	El dispositivo móvil está conectado al IED Virtual y en pantalla

previas	se encuentra un nodo escribible de todos los tipos de BDA
Objetivo	1. Enviar una petición de escritura al IED cambiando el valor de la variable en el server
Procedimiento	1. Tocar un ítem que represente un BDA. Los BDA bajo pruebas son: BdaCheck, BdaQuality, BdaTimestamp, BdaBoolean, BdaInt8, BdaInt8U, BdaInt16, BdaInt16U, BdaInt32, BdaInt32U, BdaInt64, BdaFloat32, BdaFloat64, BdaVisibleString (con longitud 32, 64 y 255), BdaUnicodeString, BdaOctectString, BdaDoubleBitPos. 2. Editar el valor de la variable usando los elementos gráficos del asistente y hacer la petición. 3. Abrir nuevamente la variable en el asistente y corroborar el cambio
Resultado	El IED Virtual recibe la petición y acto seguido cambia el valor de la variable. Al abrir nuevamente la variable escrita cambia a como se había dejado en el dialogo.
Contratiempos	La implementación contiene errores en los métodos de escritura de las variables de texto (VisibleString y UnicodeString) llamados *setStringValue(String cadena)* y se debe a que este método no tiene en cuenta la extensión del String ocasionando la excepción *outofBounds*. La solución es hacerlo mediante setValue(byte[] valor) y transformando el String en una cadena de bytes con la longitud adecuada.
Desempeño	33.183 MB de RAM, 94% en el uso de la CPU para variable de solo lectura.
Comentarios	El tiempo de escritura de las variables de texto en la aplicación es un poco más demorado mientras se transforma de texto a bytes.
Punto de vista de IEC 61850	Satisfactorio uso del servicio ACSI *SetDataValues.*

4.5. MÓDULO DE DATASETS Y RCB

Condiciones previas	El dispositivo móvil está conectado al IED Virtual en cuyo archivo .ICD asociado se encuentran 2 dataset y sus dos RCB respectivos.
Objetivo	1. Leer los RCB y Dataset de un IED 2. Crear un Dataset en el servidor 3. Eliminar un Dataset borrable del servidor
Procedimiento	1. Se escoge el modo de Datasets y RCB 2. Navegar en la lista de RCB y de Datasets 3. Cambiar al modo navegador de nodos 4. Escoger un nodo FC y crear el dataset “DatasetPrueba” 5. Escoger otro nodo FC y agregarlo a “DatasetPrueba” 6. Eliminar el dataset.

Resultado	Los dataset y RCB son cargados correctamente. Se crea el dataset satisfactoriamente, al que se le pueden agregar los nodos FC que el usuario necesite. Por último si no se requiere se elimina el dataset
Contratiempos	Cuando se crean los datasets, ellos no quedan asociados a ningún RCB, dando lugar a un acceso directo alojado en el servidor que elude a los nodos creados. Este contratiempo es de la misma de la norma al no poder crear un RCB. Para afiliar un dataset a un RCB se requiere hacerlo desde SCL.
Comentarios	Aunque la librería tiene el método setDatasetValue no se usa en el proyecto. Para escribir una variable siempre se usará el servicio datasetValue.
Punto de vista de IEC 61850	Se usan los objetos ACSI Dataset y ReportControlBlock Se usan los servicios ACSI createDataset, deteleDataset, getDatasetValue.

Los dos nodos involucrados en el proceso tienen la dirección URI “Ied1lDevice1/LLN0.Mod [ST]” y “Ied1lDevice1/MMXU1.TotW [ST].mag”

Figura 68. Creación de dataset y adición de nodos al mismo

4.6. RESISTENCIA A FALLAS EN LA CONEXIÓN O SERVIDOR

Condiciones previas	El dispositivo móvil está conectado al IED Virtual y en la lista de variables de seguimiento hay variables cuyo valor se actualiza periódicamente
Objetivo	Observar que ocurre cuando los procesos son cortados por eventos externos.
Procedimiento	1. Deshabilitar la conexión WI-FI en pleno proceso.

	2. Volver a condiciones previas 3. Desconectar el IED Virtual en pleno proceso.
Resultado	La aplicación se da cuenta que el servidor ha tenido un problema y cierra el socket y vuelve al modo Gestor de conexión.
Contratiempos	Todos los procesos que se corren desde hilos distintos al principal durante el evento de falla no pueden cerrar inmediatamente el socket. Se requiere que se notifique del proceso de segundo plano hacia la actividad principal para que luego esta sea quien desconecte la aplicación.
Comentarios	El programa percibe la desconexión con las excepciones IOException y ServiceError. Si el programa está conectado pero no usa ningún servicio la aplicación no se da cuenta sino hasta que intenta efectuar uno.
Punto de vista de IEC 61850	Si ocurre un error en el servidor o en la conexión, el cliente debe volver a solicitar la asociación puesto que la pasada no es reparable.

4.7. CONDICIONES DE MÁXIMO USO

Esta prueba muestra el aumento en el uso de la memoria RAM y uso del procesador usando los módulos más demandantes poniéndoles la carga máxima.

Condiciones previas	El dispositivo móvil está conectado al IED Virtual y en la lista de variables de seguimiento hay 10 variables, el panel de graficación está corriendo y Buffer ha llegado a los 100 datos y los enviar al servidor asociado.
Objetivo	Observar que ocurre con el uso de la memoria RAM en condiciones de máximo uso
Procedimiento	1. Seleccionar una variable del panel de seguimiento y esperar a que tome 100 datos 2. Observar cambios en el uso de la memoria RAM.
Resultado	Acer Iconia B1-A71: Esporádicamente el hilo de recepción continua es frenado por el sistema operativo quedando congelado el panel de seguimiento y el panel gráfico Samsung Galaxy Tab 2 7.0: La aplicación responde sin ralentizaciones o fallas.
Contratiempos	Acer Iconia B1-A71: Al minimizar la aplicación es rápidamente destruida la ejecución puesto que la memoria RAM es reducida y debe liberarla para las otras aplicaciones.
Desempeño	57.324 MB de RAM, 96% en el uso de la CPU
Comentarios	Los limites en la petición de variables ha sido seleccionada de forma estratégica para que no ocurra y exceso en el uso de la memoria ni en los recursos de red puesto que en condiciones máximas son 10 peticiones y 10 descargas de los valores de los

	datos. También se debe elegir un valor adecuado para el número de puntos máximo en el panel de graficación teniendo en cuanta que se llena un buffer de datos con una animación en curso. Se valida satisfactoriamente el panel de seguimiento. Se valida satisfactoriamente el panel de graficación. Finalmente se valida el script de recepción de Pares nombre-valor. Para validar el script PHP de recepción de datos se habilitó el servidor APACHE se enviaron los datos a la localización del servidor asociado. El resultado es la obtención de un archivo .txt
Punto de vista de IEC 61850	Para suplir la falta del servicio de reportes de IEC 61850 se ha recurrido al Polling para mantener actualizado al cliente ante los cambios del servidor.

Figura 69. Captura del Panel de graficación

4.8. SOBRECARGAR LOS UIELEMENTS

Si se sobrecarga de información un UIElement en Android, la aplicación se tornará pesada y lenta debido a que se ella debe pedir más memoria RAM al sistema operativo para no desperdiciar esta información.

Lo anterior se evidencia con un fichero .ICD de gran envergadura en el mapa jerárquico consta de muchos LN y vario LD. Inicialmente se tenía un módulo que mapeaba todo el IED y guardaba sus URI en un TextView lo cual era catastrófico para la aplicación la aplicación se ponía excesivamente lenta haciéndola inoperable. A raíz de esto se optó por los ListView como alternativa en el mapeo parcial del IED.

Condiciones previas	El servidor virtual corre un archivo .ICD extenso de un dispositivo real suministrado por EPSA

Objetivo	Observar que ocurre el rendimiento del programa al recibir un modelo más extenso que el modelo de las pruebas anteriores.
Procedimiento	1. Conectarse al IED 2. Observar respuesta del navegador de nodos, RCB y Datasets.
Resultado	Se llega a tener una lista de 53 ítems correspondientes a los datasets configurados en el IED. En ambos dispositivos funciona correctamente la aplicación sin perder en algún momento maniobrabilidad.
Contratiempos	Hay problemas para interpretar el parámetro RptID (ID del reporte) debido a que esta parte no está aún terminada en el proyecto abierto.
Desempeño	50.117 MB de RAM, 91% en el uso de la CPU. 6% minimizado
Comentarios	Al ser un modelo más extenso el tiempo de conexión es de alrededor 15 segundos en ambos dispositivos. En el archivo suministrado se crean varios LD cada uno con sus correspondientes datasets. No existe forma de saber cuál dataset o RCB corresponde a cual LD ya que los métodos getUrcbs() y getDatasets no distinguen su procedencia ni las clases Dataset y Rcb tienen métodos para conocerlas.
Punto de vista de IEC 61850	Todos los datasets corresponden a alarmas y medidas. Sus Fc son ST y MX respectivamente.

4.9. CONEXIÓN, LECTURA Y ESCRITURA EN IED REAL

Condiciones previas	IED T60 de General Electric escuchando peticiones de clientes. Se usa únicamente la Tablet Samsung Galaxy Tab 2 7.0
Objetivo	1. Lograr conexión con el IED Real. 2. Leer variables del IED Real. 3. Escribir variables de IED Real.
Procedimiento	1. Crear un perfil virtual de IED con IP y puerto correctos 2. Configurar opciones de la aplicación por defecto 3. Lanzar el módulo de conexión
Resultado	La conexión y lectura es satisfactoria. La escritura no.
Contratiempos	La escritura no es satisfactoria por los enums de la norma y los modos de escritura de las variables.
Desempeño	78.1357 MB de RAM, 92% máximo en el uso de la CPU, 6% de uso promedio minimizado
Comentarios	Durante la prueba no se contaba aun con el módulo operate y select.
Punto de vista de IEC 61850	La seguridad de tipo sbo-with-normal-security estaba presente en los nodos controlables del IED y el servicio ACSI setDataValue no es apto para escribir. Se comprueba el uso de *GetDataDirectory*, *GetDataDefinition, GetAllDataValues* y *getDataValue.*

5. CONCLUSIONES

A lo largo del proyecto de grado se ha explorado y utilizado la norma IEC 61850 de automatización de subestaciones eléctricas a través del uso de la implementación OPEN IEC 61850 desarrollada por ISE Fraunhofer usándola para el desarrollo de la aplicación WEB de telecontrol sobre un dispositivo móvil Android. En el presente capitulo se exponen la conclusiones del desarrollo anterior en relación a los objetivos planteados durante el anteproyecto, así como también su trabajo futuro en el marco de la movilidad de los sistemas de telecontrol para subestaciones eléctricas mencionando lo necesario para que la aplicación desarrollada sea una herramienta completa en la industria eléctrica basados en el conocimiento acumulado sobre la norma y el alcance de su implementación.

El primero de los objetivos del proyecto de grado consistía en familiarizarse con la implementación OPEN IEC 61850 para lo cual se documentó en los capítulos 3, 4 y 5 que describen la anatomía de la implementación no sólo detallando las funciones de cada clase sino que también clasificándolas según el modelo de la norma haciendo un paralelo entre la implementación y la norma. Es importante resaltar en la referencias del proyecto no se encuentra la norma IEC 61850 puesto que lo que se usó como referencia fue la información presente en bases de datos de ingeniería de la web.

Lo anterior supone una mayor dificultad en la familiarización de la implementación puesto que los objetos y servicios que esta presenta son nombrados según la terminología que compone la jerga de IEC 61850.

La documentación de cada miembro de la implementación se realizó mediante la depuración y ensayo en cada una siendo de gran ayuda el cliente y servidor de muestra de la misma. Gracias a esto se pudo establecer las siguientes limitaciones:

- ✓ Los servicios de reporte en el lado servidor están bajo desarrollo aún y se prefirió excluir las clases que representa de la aplicación móvil teniendo en cuenta los demás objetivos. El desarrollador advierte que dichos servicios no son soportados por los servidores pero si por los clientes sin embargo esto necesitaría un IED real en una operación real para hacer uso de los reportes y comprobarlo.
- ✓ La clase de asociación carece de un listener que permita reconocer si un servidor está en red o no. Se puede emplear técnicas de poll para saberlo, sin embargo no es viable puesto que satura la red si hay varios clientes haciendo polling en al IED.
- ✓ En el diseño de archivos de configuración con SCL es común usar varios LD. La implementación impide saber el origen de los datasets configurados en los distintos LD.

- ✓ Los tipos de archivo de texto presentan algunos problemas de longitud de palabra. Los métodos de escritura de variable como setStringValue(String valor) no funcionan correctamente teniendo que armar byte a byte la trama del mensaje.

No obstante no es obstáculo para desarrollar los demás objetivos. La implementación garantiza la conexión a un IED y obtención del modelo.
En cuanto a movilidad, se logra embeber satisfactoriamente la librería en un dispositivo móvil logrando usarla en él gracias a que fue implementada en el mismo lenguaje de programación utilizado para el desarrollo de aplicaciones en Android, Java. Algunos métodos de sus clases usan interfaces o librerías no soportadas por Android pero corresponde a implementaciones de prueba de la librería como el servidor y cliente de muestra que hacen uso de la librería Logger (no soportada por Android) y se encuentra en el código fuente de la implementación. Se identifican estos métodos para obviarlos y que no exista conflicto con el sistema operativo.
En cuanto al uso de protocolos inalámbricos se usa WI-FI (IEEE 802.11). En OPEN IEC 61850 existe la clase clientAssociation para lograr una conexión entre un IED y un cliente. Esta clase recibe como parámetros la dirección IP y puerto para establecer una conexión TCP lo que sugiere que el cliente no necesariamente debe tener la misma pila OSI de un IED para llevar a cabo una conexión. Los únicos protocolos necesarios son del nivel 3 hacia arriba, esto es, TCP/IP en las capas de transporte y red respectivamente y como protocolo de aplicación MMS/ACSI. El protocolo de enlace de datos y la capa física es la única diferencia entre las dos pilas y es lo que permite hablar de movilidad en un cliente para interactuar con un IED.
En cuanto a la interacción lograda con el IED la aplicación móvil fue capaz de conectarse, leer, escribir e identificar todos los miembros del modelo en un IED. Inclusive miembros que no eran objetivo en el proyecto de grado como los Datasets y RCB. Se hace una exploración casi completa de la librería cuando solo se planeaba al más el uso de dos servicios ACSI.
Adicionalmente el conocimiento de los servicios ACSI y su presencia en la implementación favoreció el número de posibles interacciones con el IED además de las ya mencionadas se lograron implementar a la aplicación los servicios ACSI de creación, eliminación, lectura y escritura de datasets y el servicio de escritura SBO (Select y Operate). El desempeño en todas fue la apropiada y se notó una gran diferencia en el equipo real y el simulado en cuanto al tiempo de conexión que varía según la envergadura del modelo en el archivo .ICD.
Finalmente para la representación virtual de los datos recibidos se logra con el asistente de lectura y escritura que contiene elementos gráficos de Android que permiten visualizar todos los atributos de los tipos de datos. Inicialmente se

dispone de muchos tipos de dato pero se profundiza en los atributos básicos sin descuidar los tipos para los servicios. El único es excluido es un tipo de dato que ofrece el tiempo en el que se produce un reporte llamado *BdaEntryTime* de la que no se documenta al excluirse la parte de los reportes en el cliente. Para suplir este servicio se creó el Hilo receptor. Se trata de usar una técnica de polling para el IED con el fin de hacer hacer la lectura periódica de variables abiertas en el asistente y que se pueda preguntar automáticamente cada cierto tiempo. Se logra mediante el hilo de recepción continua que además permite hacer seguimiento de hasta diez variables del IED al mismo tiempo. La principal dificultad y en general del diseño de aplicaciones es el uso inteligente de la memoria del dispositivo. Se requiere que las variables globales usadas en el programa no excedan un umbral razonable en la información contenida, sino limitarse al uso necesario. Por ejemplo para la navegación en nodos se podría guardar nivel por nivel las listas de nodos que componen el IED para que no sea necesario el algoritmo de retroceso en el nivel jerárquico pero es preferible, para mejorar el desempeño, añadir un método que genere la lista de nodos del nivel anterior. Esta premisa es la que limita el número de nodos a seguir simultáneamente o el número máximo de puntos que se pueda guardar en el buffer de salida hacia el servidor. Los límites fueron elegidos según el desempeño del dispositivo con el que se desarrolla la aplicación y la prueba de máximo uso de las capacidades de la aplicación procurando lograr que no se deteriore el desempeño de aplicación evitando las ralentizaciones o saltos de cuadros por segundo. Basados en las pruebas con la tablet Hacer se puede concluir que se requiere mínimo 1GB de RAM para que funcione correctamente la aplicación con estos límites.

Para reforzar el objetivo se implementa un plano cartesiano animado que permite observar el movimiento de las variables de tipo numérico conforme van siendo recibidas en la aplicación.

Adicional a lo que se tenía planeado en la tesis está el módulo de apertura de archivos .ICD que permite observar offline la constitución de un IED configurado cargando con la misma interfaz de la aplicación la disposición de los nodos.

5.1 TRABAJOS FUTUROS

El proyecto abierto OPEN IEC 61850 está aún en fase de desarrollo pero se pueden hacer aportes para complementar el simulador actual de IED para que sea capaz prestar el servicio de reportes. De este modo se puede implementar el uso de servicio de reportes en el cliente para funcionar con IED reales prescindiendo de la técnica de polling actualmente implementada.

Gracias a los reportes y las funciones de notificación al usuario de Android se pueden usar los reportes recibidos emitidos por el IED de los RCB a los que se haya suscrito el usuario para mantenerse al tanto de lo que acontece en la planta.

Esta operación se puede llevar a cabo sin necesidad de estar en la LAN industrial sino que se puede estar suscrito a través de internet mediante un NAT en la red de comunicación de la subestación que trate a un equipo externo como uno de la intranet.

Adicionalmente se pueden extrapolar las funciones de un cliente recurriendo a las bases de datos. Un acercamiento inicial fue la publicación de logs en el servidor con valores de variables en una toma de datos en el IED simulado pero es posible reportar a un servidor WEB los valores tomados, el IED del que fueron tomados, el sector de su ubicación, etc.

REFERENCIAS

Illinois security lab (2007). IEC 61850 - Communication Networks and Systems in Substations, An Overview of Computer Science.

Rivas, E. (2009). Diseño de una plataforma de comunicaciones bajo la norma IEC-61850.

Systemcorp PTY Ltda. (2010). IEC 61850 Protocol API User Manual, protocol Integration Stack

Adamiak, M., Baigent D., Mackiewicz R. (2009). IEC 61850 Communication Networks and Systems In Substations, an Overview for Users.

Kirrman, H. (2012). Introduction to the IEC 61850 electrical utility communication standard.

SISCO Documentation (1995). Overview and Introduction to MMS

Sidhu T., Kanabar M., Parikh, P., (2009). Implementation Issues with IEC 61850 Based Substation Automation Systems

Ozansoy C. R., Zayegh A., Kalam A. (2008). Time Synchronisation in an IEC 61850 Based Substation Automation System

Android Developers (2014). Página WEB para desarrolladores de la plataforma Android.

Fraunhofer ISE, Stefan Feuerhahn (2014). OPEN IEC 61850, proyecto de OpenMUC (Monitor and Control)

Rivadeneira, I. (2005). Análisis de protocolos de comunicación para la Automatización de Subestaciones de Transmisión Eléctrica.

BPL Global, Ltd. (2011). IEC 61850 Guide. Serveron TM8TM and TM3TM On-line Transformer Monitors.

Cubero, K. (2012). Automatización de las subestaciones de la Compañía Nacional de Fuerza y Luz S.A., aplicando la Norma IEC-61850.

ANEXOS

1. SCL (SUBSTATION CONFIGURATION LANGUAGE)

Es un lenguaje de descripción de los componentes de una subestación eléctrica basado en XML. La utilidad más importante es la creación de scripts de configuración para los IED aunque con este se puede describir toda la subestación.
La información que se puede incluir con los scripts SCL, las respectivas etiquetas y parámetros se encuentran en la parte 7-X de la norma.

1.1 PARTES DEL SCL

Las partes del SCL divide la tarea del diseñador de la funcionalidad de la subestación en varias partes. Cada parte tiene su etiqueta asociada en el script SCL final.

(A) Encabezado

El encabezado se usa para identificar la versión del archivo, autor, software usado para su creación y otros detalles básicos sobre el archivo. Un ejemplo puede verse a continuación.
<Header id="header1" version="4.0.2" revision="1.02" toolID="Kalkitech SCL Manager" nameStructure="IEDName" />

(B) Subestación

La figura 69 muestra el organigrama del script, cada rectángulo representa una etiqueta y por ser de tipo XML cada etiqueta tiene unos parámetros asociados y otras etiquetas internas.

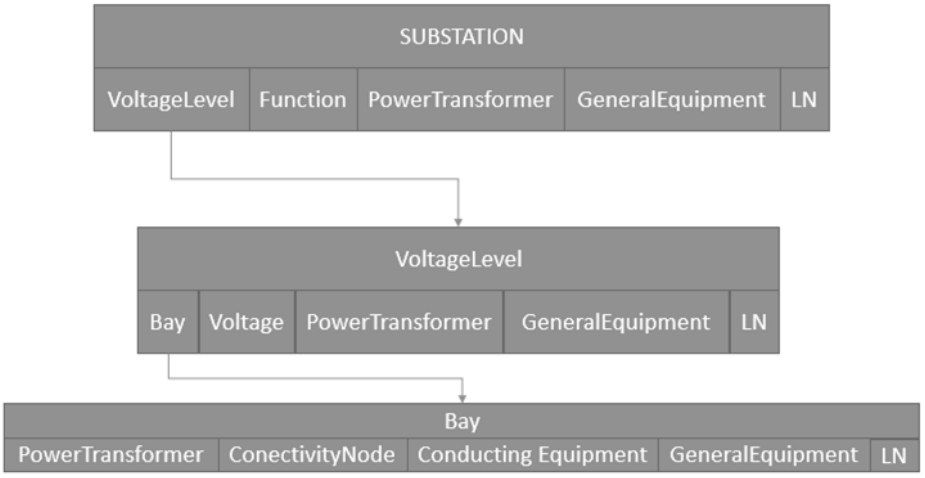


Figura 69. Organigrama de la etiqueta Substation

La parte de Subestación describe la interconexión de elementos al interior de la subestación basados en el diagrama unifilar de esta. Se pueden especificar

niveles de voltaje de líneas en particular, así como la inclusión de transformadores de potencia, equipo general, etc.
Para la edición de estos archivos se usan editores gráficos que arrojan el script SCL de forma automática. El *Kalkitech SCL Manager* o *Visual SCL* son ejemplo de ello.
En el diagrama unifilar que puede observarse en la figura 70, los elementos que se encuentran en él tiene una jerarquía simple: Se usan bloque de nivel de voltaje como bloques contenedores, dentro de ellos se agregan bloques de bahías que son líneas de funciones como Q3, que tiene un transformador de corriente I1 al cual se le mide el voltaje en el primario mediante U1 y su alimentación puede ser interrumpida mediante un breaker QB1 o mediante un breaker de control dentro de la bahía QA1.

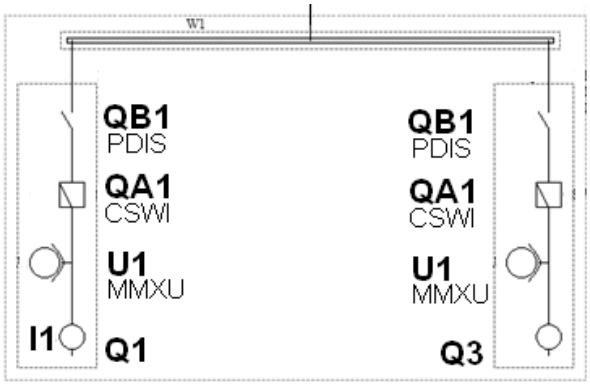


Figura 70. Diagrama unifilar del ejemplo

Las condiciones de la alimentación son ajenas a la función de la bahía, por esto se encuentra fuera del bloque. A cada elemento se le puede poner un nodo lógico que debe ser acorde a la función en el diagrama. Por ejemplo al medidor de voltaje U1 se le puede asociar un nodo lógico MMXU (Nomenclatura de un nodo lógico para la medición).
Dentro del archivo, la subestación se puede reconocer porque es definida mediante la etiqueta *Substation.*
Dentro de la etiqueta *Bay* se define cada elemento de la bahía. Dentro de la etiqueta *ConductingEquipment*se definen los transductores o actuadores, cada uno con su correspondiente nodo lógico. La conexión de los elementos se define dentro de la etiqueta *ConnectivityNode* así como también la conexión entre las bahías. Con la etiqueta *PowerTransformer* se puede incluir un transformador de potencia a este nivel para integrar la bahía aunque también se pueden llamar transformadores de potencia definidos en etiquetas de niveles más arriba como en *Substation* o *VolageLevel*.
Existen otras etiquetas internas en algunas de las mostradas en el organigrama útiles para describir detalladamente un transformador de potencia o una función de la subestación.

(C) Comunicación

La figura 71 es el organigrama de la etiqueta *Communication* para la descripción de la parte de comunicación del archivo SCL

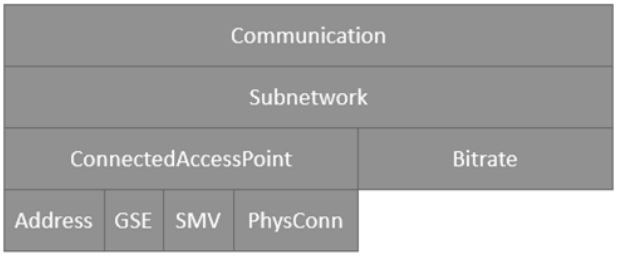


Figura 71. Organigrama de la etiqueta Communication

En esta sección sirve para definir puntos de acceso a los IED desde cualquier equipo de red como un computador para interactuar con él o configurarlo. Se puede seccionar la red de la subestación mediante la etiqueta *SubNetwork* a la que se le configura el Bitrate y se añaden algunos puntos de acceso con la etiqueta *ConnectedAP*. Con la etiqueta *Address* se le configura la dirección IP, la máscara de red, la puerta de enlace por defecto y otros parámetros de una red basada en IP para el acceso al servidor MMS de un IED.

Tipos de conexiones como GSE, SMV y PhysConn como se vio con anterioridad está basada en Ethernet y no en IP, es decir que a este tipo de etiquetas se configuran parámetros como la dirección MAC, la prioridad de VLAN y APPID (ID de aplicación).

(D) IED

En esta sección se describen las funciones que efectúa cada IED que compone la parte de subestación del archivo SCL.

Un IED puede representar en el archivo SCL desde un simple breaker accionado a distancia hasta toda una bahía con un transformador de instrumentación, transductores para monitorear su estado, protecciones de sobretensión, sobrecorriente y sobretemperatura según el criterio del diseñador y lo que el IED permita.

El organigrama de la etiqueta IED es el que se muestra en la figura 72.

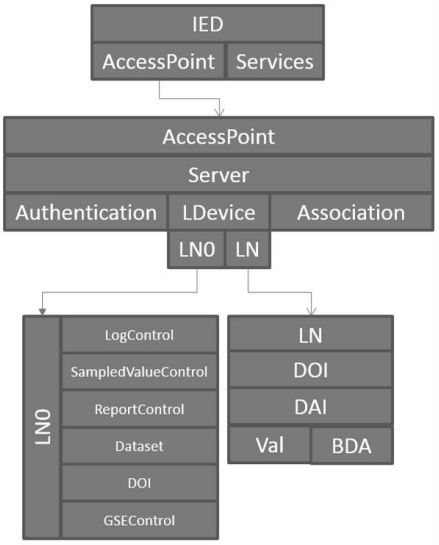


Figura 72. Organigrama de la etiqueta IED

Con la etiqueta *Services* se permite la selección de servicios a ofrecer por parte del IED. Estos servicios son predefinidos por la norma mediante la mención de la etiqueta correspondiente. Algunos de los servicios no tienen ningún parámetro, basta con mencionarlos para decidir usarlos, estos son:
DynAssociation, GetDirectory, GetDataObjectDefinition, DataObjectDirectory, GetDataSetValue, SetDataSetValue, DataSetDirectory, ReadWrite, FileHandling, GSEDir, TimerActivatedControl, GetCBValues, ReportSettings, LogSettings, GSESettings, SMVSettings, ConfLNs, SettingsGroups. *ConfDataSet, DynDataSet. ConfReportControl, ConfLogControl, GOOSE, GSE.*
Algunos de estos servicios se refieren a servicios ACSI y a permisos de acceso a la configuración de un cliente al servidor bajo configuración.
La etiqueta *AccessPoint* contiene la etiqueta *Server* clave para contactar a un IED en una red basada en IP. Esta a su vez contiene la etiqueta *LDevice* (Logical Device) de suma importancia en el proyecto puesto que es la representación virtual del IED desde el punto de vista funcional.
Cualquier función que puede ejecutar un IED se representa con la etiqueta de nodo lógico *LN* que está contenida en la etiqueta LDevice. Algunas funciones van desde medición, protección, descripción, etc.
LDevice tiene también una etiqueta reservada para la creación del nodo lógico principal LN0 en el que se pueden crear datasets y bloques de control de reporte.

(E) Nodos Lógicos indispensables

Todo IED debe tener al menos dos nodos lógicos configurados.

✓ El primero se refiere al dispositivo físico (PD) el cual debe ser llamado LPHD. Este nodo contiene la descripción del equipo, algunas fechas de revisión y demás información referente al fabricante.

✓ El segundo nodo, el principal, se refiere al dispositivo lógico (LD) el cual debe ser llamado LLN0. Contiene los datasets y bloques de control de reportes. Es el nodo de apertura, el primero que debe ser mencionado.

(F) DataTypeTemplates

Los *DataTypeTemplates* son formatos de estructuras de datos presentes en el script SCL. Son etiquetados con una cadena de caracteres como "Template1" o "MMXU1" gracias al parámetro *id*. El uso de este le permite a un *DataTemplate* ser invocado por otras etiquetas que componen el script. La figura 73 muestra las etiquetas internas de la etiqueta *DataTemplates*.

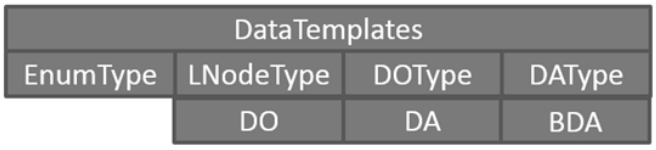


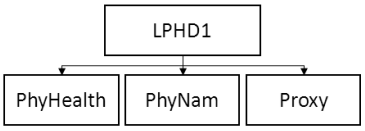
Figura 73. Organigrama de la etiqueta *DataTypeTemplates*

Se pueden describir los LN, DO y DA para definir la estructura del IED.

La etiqueta LNodeType puede ser llamada por la etiqueta LN de la parte IED. Es la única que puede ser invocada desde etiquetas externas a *DataTemplate*. La definición de los DO y DA requieren el uso las etiquetas DOType y DAType respectivamente. Para asociar definiciones de LN, DA y DO se hace uso del parámetro type en donde se enuncia el id de la definición.

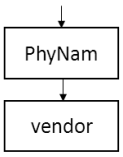
Por ejemplo en la creación de un nodo LPHD para la descripción del sistema físico se puede definir de la siguiente forma:

```
<IED name=IEDejemplo>
...
<LN lnClass="LPHD" lnType="LPHD1" inst="1" prefix="" />
</IED>
<DataTypeTemplates>
...
<LNodeType id="LPHD1" lnClass="LPHD">
      <DO name="PhyNam" type="DPL_1_PhyNam" />
      <DO name="PhyHealth" type="INS_1_Beh" />
      <DO name="Proxy" type="SPS_1_Proxy" />
</LNodeType>
...
</DataTypeTemplates>
```



lnType permite hacer el llamado al LNodeType. Esta referencia permite heredar todo lo descrito en el DataTypeTemplate. Así el nodo LPHD1 tendrá los DO PhyNam, PhyHealth y Proxy. Como se observa cada DO también tiene su dataTypeTemplate asociado, que por deducción deben ser <DOType/>. La definición, por ejemplo de *DPL_1_PhyNam* puede ser como está a continuación.

```
<DOType id="DPL_1_PhyNam" cdc="DPL">
      <DA name="vendor" bType="VisString255" fc="DC" />
  </DOType>
```



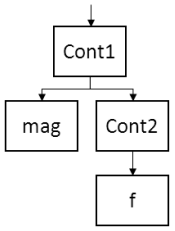
De esta forma LPHD1 queda asociado con DPL_1_PhyNam. Con la etiqueta CDC se puede especificar el CDC al que pertenece la definición (Véase a continuación). Con el parámetro bType se define el tipo primitivo de dato para el DA que presenta la norma IEC 61850. En este caso se define como un VisibleString de modo que este DA es un BDA. Finalmente con la etiqueta fc se especifica el FunctionalConstraint del DA que lo clasifica por su funcionalidad.

(G) DA Contenedor

La etiqueta *DAType* es un contenedor de BDAs que son creados mediante la etiqueta *BDA*. Esta puede presentar un tipo de dato básico (parámetro bType) alusivo a los tipos básicos de datos presentes en el estándar. Sin embargo existe un tipo básico que convierte este DA en un contenedor de otros DA, se trata del bType '*struct'*.

Por ejemplo se define un DA Cont1:

```
<DA name="Cont1" bType="Struct" type="Vector_1"
       fc="DC" />
...
<DAType id="Vector_1">
      <BDA name="mag" bType="INT16U" />
      <BDA name="Cont2" type="AnalogueValue_1" bType="Struct" />
</DAType>
...
<DAType id="AnalogueValue_1">
      <BDA name="f" bType="INT32U" />
</DAType>
```

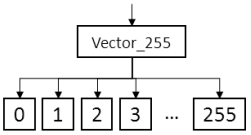


(H) Vectores BDA

Con la etiqueta BDA se puede dar lugar a una estructura vectorial de datos en vez de tener un solo dato.

Por ejemplo se crea un DAType

```
<DAType id="Vector_255">
      <BDA name="f" bType="INT16U" count="255"/>
</DAType>
```



De este modo el nodo Vector_255 será el contenedor de un arreglo de DA nombrados con un numero de cero al valor de *count* con el tipo básico especificado.

(I) Common Data Classes (CDC)

A la hora de crear un archivo SCL, la norma ofrece plantillas de LN, DO y DA llamados CDC (Clases comunes de datos). Las plantillas han sido establecidas para facilitar la tarea de diseño de un IED ya que estas contienen los componentes relevantes y que no se deberían pasar por alto para llevar a cabo una tarea como la medición de una variable física.

El lenguaje para la configuración de un IED es el instrumento para implementar lo que está en los CDC aunque el usuario es libre de crear sus propios LN, DO y DA. Las CDC contienen algunos elementos que no deben faltar (mandatorios) y otros que no (opcionales). Los DataTypeTemplates son la herramienta idónea para la implementación de los CDC.

(J) Ejemplo de nodo lógico perteneciente a las CDC

El nodo lógico XCBR tiene una plantilla establecida al estar incluida en el CDC. La plantilla contiene en la columna Attribute Name el nombre del DO. Cada DO tiene su Attt. Type y si es Mandatorio u Opcional. En el ejemplo se ha seleccionado el DO Loc cuyo Attr. Type es SPS (Single Point Status). Esta nomenclatura de tres letras mayúsculas hace referencia a especificaciones de funciones del nodo lógico al que pertenecen.

En la norma la clase XCBR de nodo lógico se muestra como aparece en la figura 74.

XCBR CLASS		
Attribute Name	Attr. Type	M/O
LNName		M
DATA		
Common Logical Node Information		
Loc	SPS	M
EEHealth	INS	O
Controls		
Pos	DPC	M

SPS CLASS		
Attribute Name	Attr. Type	M/O
DataName		M
DATA ATTRIBUTE		
Status (ST)		
stVal	Boolean	M
q	Quality	M
t	Timestamp	M
Description (DC)		
d	VisibleString	O

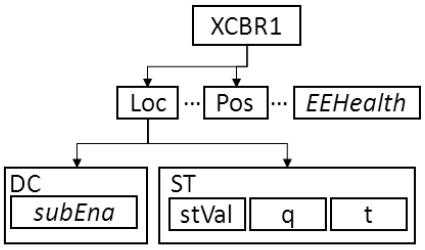


Figura 74. Ejemplo del CDC de XCBR

El organigrama de la derecha muestra cómo se debe crear el nodo lógico. En amarillo los nodos opcionales y en azul los mandatorios en la rama del DO ‘Loc’.

LISTA DE TABLAS

LISTA DE FIGURAS