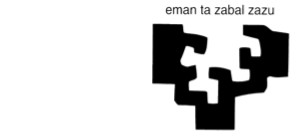


ZTF-FCT
Zientzia eta Teknologia Fakultatea
Facultad de Ciencia y Tecnología

Trabajo de Fin de Grado / Gradu Amaierako Lana

Doble Grado en Física e Ingeniería Electrónica
Fisika eta Ingeniaritza Elektronikoko Gradu bikoitza

Desarrollo de un sistema de monitorización, análisis y diagnóstico de máquinas y equipos industriales con acceso desde una aplicación móvil

---

Autor / Egilea:

Pablo Bedialauneta Rodríguez

Director / Zuzendaria:

Estibaliz Asua Uriarte

Leioa, julio de 2019 / Leioa, 2019ko uztaila

Índice

0. Introducción

La estrategia global, compromiso de dirección, cultura de adaptabilidad e innovación y las tecnologías facilitadoras, son los ingredientes de la transformación digital en las empresas. Desde la primera revolución industrial hasta la era digital en las empresas de fabricación, han pasado casi 300 años. En tiempos de James Watt los pocos datos que se recopilaban eran grabados en papel, y la fabricación se realizaba todavía por medios artesanales, lo que hacía que no fuese factible ni mucho menos probable o beneficiosa la integración con lo que ahora se conoce como las tecnologías de la información, también inexistentes en aquella época. Sin embargo, esta brecha fue decreciendo con la aparición de los ordenadores cuyo rápido desarrollo tecnológico condujo la fabricación hacia etapas tempranas de digitalización[1].

El amplio uso de las diferentes tecnologías desde la década de los 50 con los desarrollos en control numérico (Alan Parsons, MIT) hasta la de los 80 con la llegada de Internet y pasando por los 60 con los circuitos integrados (PCBs) dio como resultado tecnologías avanzadas como la fabricación integrada (CIM), el diseño asistido (CAD) y la fabricación por ordenador (CAM), los sistemas de ejecución de fabricación (MES) y sistemas de planificación de recursos empresariales (ERP).

Recientemente, el auge de nuevas tecnologías como "Internet de las cosas" (*Internet of Things*, IoT), la computación en la nube o en los dispositivos (Cloud y Edge Computing, respectivamente), el análisis de Big data y la inteligencia artificial, continúan revolucionando y evolucionando el paradigma de la fabricación. Sin embargo, las crisis económicas de los últimos años junto con la necesidad de brindar una respuesta rápida a nuevas oportunidades de negocio, han cambiado radicalmente el panorama del sector industrial. Las empresas en busca de mano de obra más económica, mejores condiciones contractuales, nuevos mercados y productos y mejores beneficios, están ahora comprometidas en recuperar la competitividad de antaño.

Actualmente, la modernización de los procesos industriales en el fenómeno llamado Industria 4.0 permite a los fabricantes la realización de proyectos complejos de manera eficiente.

En este trabajo se lleva a cabo la implementación completa de una herramienta de monitorizado en el Centro de Fabricación Avanzada Aeronáutica (CFAA), que es un centro especializado en la investigación e implementación de nuevas tecnologías de fabricación con enfoque en el sector aeronáutico. Se parte de una infraestructura incipiente compuesta por una red local y se termina con una aplicación móvil conectada a una base de datos que presenta en tiempo real la información devuelta por las máquinas y sensores. En

el trabajo aquí presentado se discuten todos los aspectos relacionados con el desarrollo mencionado, desde la programación hasta la decisiones tomadas a la hora de la implementación. En concreto, se ha incorporado un sistema de monitorización de una máquina-herramienta comunicado con una base de datos que almacena tanto los datos leídos como las credenciales de los usuarios de la aplicación móvil en la que se presenta la información adquirida del sistema de lectura.

Debido al carácter industrial del trabajo, hay varias entidades involucradas en el proyecto. Por ello, hay dos piezas de software que se usan en la infraestructura implementada que son confidenciales. La primera de ellas es el programa encargado de leer los datos de la máquina, y la segunda la encargada de subir estos datos a la propia base de datos. Sin embargo, se ha desarrollado un ejemplo simplificado de la primera para mostrar las capacidades y facilitar el entendimiento del proceso. Del segundo programa no se ha hecho un demostrador ya que el desarrollo es trivial y, además, se ha implementado una comunicación más compleja con la base de datos en la propia aplicación Android.

Los paso llevados a cabo en el trabajo son:

1. Selección de una primera máquina-herramienta para implementar la monitorización remota.
2. Selección del equipo para la lectura de datos.
3. Desarrollo de la arquitectura software y servicios para la lectura de datos.
4. Implementación de la arquitectura software y servicios para la lectura de datos mediante el instrumento.
5. Selección de la plataforma para el almacenamiento de datos.
6. Desarrollo de la arquitectura software para el almacenamiento de datos.
7. Implementación de la arquitectura software para el almacenamiento de datos.
8. Selección de la estrategia de muestra de datos.
9. Desarrollo del software para la muestra de datos.
10. Implementación del software para la muestra de datos.
11. Generalización del proceso para otras máquinas.

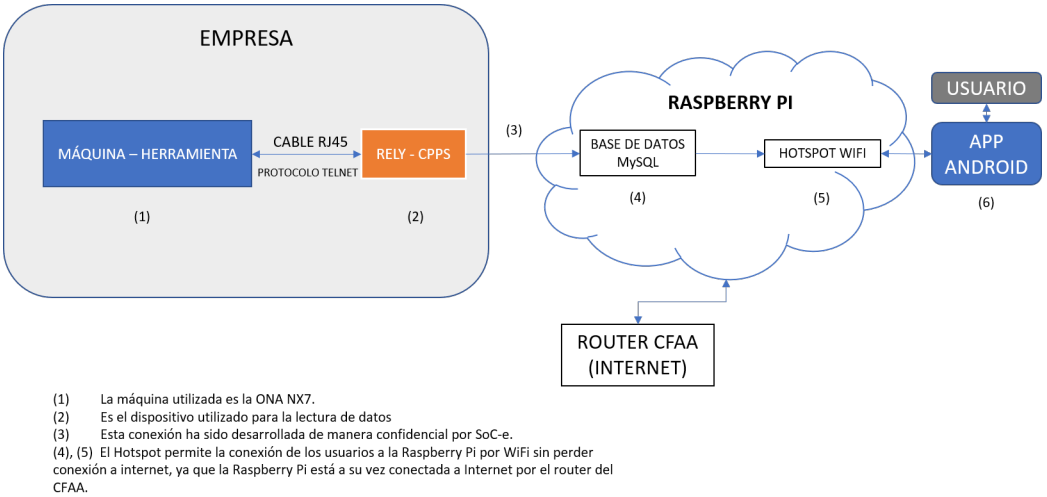


Figura 1: Esquema de la infraestructura implementada.

1. Selección de la máquina inicial para la toma de datos

La selección de la máquina en la que inicialmente recoger los datos se ha hecho de forma que, cuando el proceso de monitorizado esté terminado y completamente implementado, los datos extraídos puedan ser de gran utilidad tanto para el mantenimiento de la máquina como para mejorar los procesos que en ésta se den.

Dependiendo del proceso que se lleva a cabo en cada máquina, el margen de optimización de éste es mayor o menor. Por ejemplo, una máquina de metrología en la que se miden las dimensiones de diferentes piezas fabricadas no va a tener cabida para la optimización de las medidas si se conocen en todo momento los parámetros de operación, ya que la medición (y la precisión de ésta) es dependiente casi al completo del instrumento utilizado. Sin embargo, procesos como el mecanizado en el que se elimina material y se da forma a una pieza son por naturaleza dinámicos, por lo que un conocimiento de los parámetros de mecanizado en todo instante puede ser de gran valor a la hora de validar la integridad de una pieza en tiempo real, e incluso corregir desviaciones del proceso ideal de mecanizado utilizando los datos obtenidos.

Por ello, se ha de seleccionar una máquina del Centro de Fabricación Avanzada Aeronáutica (CFAA) que no sólo tenga cabida para la optimización del proceso, sino que esté involucrada en varios proyectos de investigación que

Figura 2: Máquina ONA NX7.

puedan aprovecharse de ese aumento de eficiencia.

La ONA NX7 es la máquina que se ha seleccionado. Esta máquina sirve para llevar a cabo procesos de fabricación y mecanizado mediante electroerosión. Además, presenta una gran ventaja ya que se dispone de la documentación detallada del conexionado de dispositivos a la máquina mediante Ethernet. El documento ha sido proporcionado bajo un acuerdo de confidencialidad, por lo que no se explicará en detalle el funcionamiento. Debido a la forma en la que se recoge la información tomada por los sensores integrados en la máquina, sólo se necesita un dispositivo programable con conector para cables RJ45. Además, la ONA NX7 es una máquina que se usa a tiempo parcial en el centro, por lo que la disponibilidad para paralizar las pruebas del sistema a implementar en este trabajo es mucho mayor que para muchas otras máquinas.

El proceso de electroerosión consiste en posicionar a una distancia mínima (normalmente del orden de la centésima de mm) en el seno de un dieléctrico la pieza a mecanizar y un electrodo cuya forma es el negativo de la geometría que se desea eliminar de dicha pieza. Es necesario que la pieza sea de un material conductor. Así, se aplica una diferencia de potencial pulsada suficiente para que se formen arcos eléctricos pulsados también entre el electrodo y la pieza. Esto arranca viruta de la pieza, y repitiéndolo suficientes veces, se consigue eliminar por completo la geometría deseada [2].

Figura 3: Dispositivo RELY-CPPS de SoC-e.

2. Dispositivo de lectura de datos.

2.1. Selección del dispositivo

La selección del dispositivo se ha realizado de acuerdo a las características e interfaces necesarias dictadas por la documentación de la ONA NX7. Las condiciones mínimas son que sea un dispositivo programable y que tenga al menos una conexión RJ45 y otra USB o RJ45, de forma que el dispositivo sea capaz de leer datos y canalizarlos al hardware en el que se despliega la base de datos (de ahí la necesidad de una conexión extra USB o RJ45 con respecto a las especificaciones impuestas por la ONA NX7).
Sin embargo, se ha decidido usar un dispositivo cuyas características excedan las mínimas aquí presentadas, de forma que pueda ser utilizado para otros propósitos o en otras máquinas sin necesidad de compra de periféricos.
El dispositivo seleccionado ha sido el RELY-CPPS de SoC-e/Relyum. Es un sistema Unix con interfaces para adquisición de datos directa desde sensores, conexión a dispositivos externos mediante USB y Ethernet, y capacidades de networking que permiten la ejecución de aplicaciones IoT mediante el uso de protocolos como el OPC-UA o MQTT, que actualmente son los estándares en la industria [3] [4].

2.2. Especificaciones del RELY-CPPS

Las principales especificaciones del dispositivo cedido y utilizado para la adquisición de datos en la ONA NX7 son las siguientes [5]:

Interfaces de comunicación:

- 4 conectores SFP para conexión de cobre Ethernet 10/100/1000Ba-

seTX o fibra 100BaseFX/1000BaseX.

- 1 puerto de cobre Ethernet 10/100/1000BaseTX.
- 1 puerto RS485 half y full duplex.
- 2 puertos USB tipo A.
- 1 puerto HDMI.

Capacidad de procesamiento:

- Procesador dual-core ARM9 embedido en la FPGA ZYNQ de Xilinx.

Módulo de sensores:

- 3 entradas digitales. 0-24 V.
- 1 entrada analógica. 0-20 mA.
- Interfaces de sensores de temperatura:
  - 1 entrada para RTD de 3 cables (PT100).
  - 1 entrada para termopar tipo K.
- 1 entrada para acelerómetro piezoeléctrico (IEPE).
- 1 conversor analógico-digital (ADC).

El módulo de sensores no ha sido utilizado en la aplicación aquí presentada, sin embargo, será de gran utilidad cuando la máquina a monitorizar no tenga por sí misma un sistema de sensorizado, permitiendo así conectar los sensores externos deseados que se pongan en la máquina directamente al RELY-CPPS.

Así, este dispositivo dispone de todas las interfaces necesarias para poder realizar la conexión a la ONA NX7, y además, al llevar un sistema operativo Unix, se pueden implementar los programas (*scripts*) necesarios para la lectura y transferencia de datos a la base de datos.

2.3. Desarrollo de la arquitectura software y servicios para la lectura de datos mediante el dispositivo RELY-CPPS.

El desarrollo del software final necesario para la toma de datos mediante el dispositivo RELY-CPPS ha sido desarrollado por SoC-e [6]. Es un servicio

de Linux que permite la comunicación con la máquina ONA NX7 con una frecuencia especificada (frecuencia de lectura de datos). En la comunicación, se leen las variables especificadas también por el usuario. El modo de almacenamiento de los datos y la estructura interna de la memoria es información propietaria de SoC-e, por lo que no se discutirá en este documento.

Para demostrar el funcionamiento de la lectura y transmisión de datos, se ha desarrollado un script en Python que es capaz de realizar una petición de datos determinada a la máquina, y procesa la respuesta para mostrar el resultado al usuario.

El script se basa en el protocolo explicado en la documentación de la máquina [7], y se explica a continuación.

Para establecer la conexión entre la máquina y el dispositivo, hace falta conocer la dirección IP de la máquina. La dirección IP se puede configurar desde el propio mando de control de la ONA NX7. Se ha determinado la siguiente dirección IP: 192.168.4.65. También se ha de especificar el puerto en el que se hará la conexión entre el dispositivo y la máquina. Se ha mantenido el puerto por defecto: 8080.
Como se ha mencionado, la conexión entre el dispositivo de adquisición de datos se hace mediante un cable RJ45 al puerto de servicio de la ONA NX7. En la propia máquina, el operario ha de acceder al modo superusuario la primera vez que se enciende mediante la introducción de una contraseña establecida.

Como se puede observar en el ejemplo de mensaje **XML** mostrado más adelante, para obtener el valor de la coordenada X se ha de leer el valor entre las marcas **X**, que están entre las marcas **cavity_coords**, que están a su vez entre las marcas **exec_data**, que está finalmente entre las marcas **XML**. Recorriendo esta estructura anidada que sigue las directrices especificadas en [7] se puede obtener el valor de las variables deseadas.

Para mandar las peticiones se hace uso del protocolo de red Telnet [8][9]. Telnet es un protocolo que permite el acceso y control remoto de dispositivos conectados por una red global o local mediante el uso del estándar TCP/IP. En el caso de este trabajo, la conexión es entre el dispositivo de lectura de datos y la máquina ONA NX7. La máquina alberga un servidor Telnet cuya IP y puerto de conexión es el establecido manualmente con anterioridad. El cliente es, en este caso, el dispositivo de toma de datos, que ha de conectarse a dicha dirección IP en el correspondiente puerto. Así, el servidor está continuamente esperando peticiones XML en el puerto de red de servicio. Cuando recibe una petición XML, la procesa y devuelve la respuesta por el mismo canal de conexión.

La comunicación mediante Telnet se ha realizado usando la biblioteca *telnetlib* [10]. La clase implementada en *telnetlib* permite abrir una conexión remota con un servidor y mandar y esperar mensajes de forma sencilla mediante el constructor y los comandos `Telnet.write()` y `Telnet.read_all()` [10].
Para el módulo XML, se han predeterminado las direcciones en el árbol XML y las peticiones necesarias para obtener las variables de la posición X, Y y Z.

Una vez está configurada la conexión manualmente se puede proceder a la automatización de las peticiones. Las peticiones se hacen mediante Strings con formato XML. Todas las peticiones comienzan con `<XML>` y terminan con `</XML>`. Entre las marcas iniciales y finales, se escriben las categorías de información que se quiere leer. Cuando la ONA NX7 recibe la petición, responde con otra String con formato XML con la respuesta a la petición, o un error si no ha sido capaz de procesar dicha petición o la petición es errónea.
La respuesta de la máquina comienza y termina también con las marcas `<XML>` y `</XML>`, respectivamente. Además, los datos devueltos siguen una jerarquía de árbol, de forma que una petición y su correspondiente respuesta tendría la siguiente forma:

Petición:

```
<XML><exec_data/></XML>
```

Respuesta:

```
<XML>
  <exec_data>
    <cavity_coords>
      <X>15000</X>
      .
      .
      .
      .
      .
    </cavity_coords>
  </exec_data>
</XML>
```

El funcionamiento del programa es el mostrado en la Figura 4.

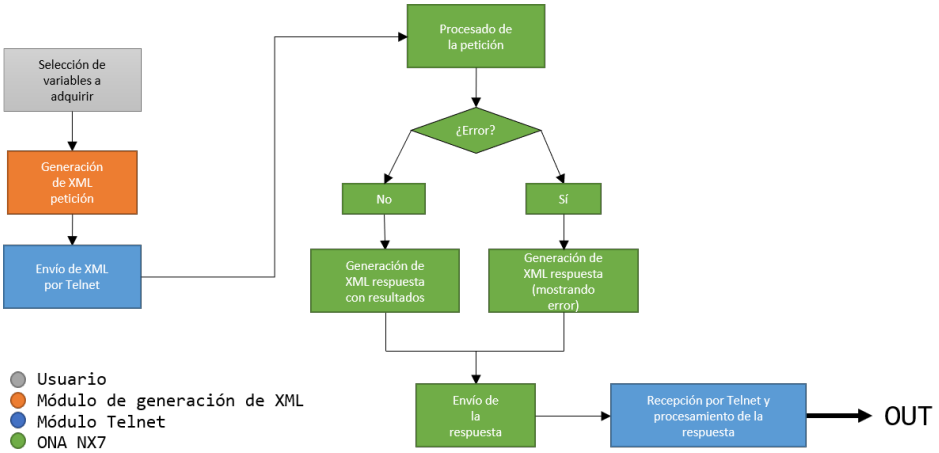


Figura 4: Diagrama de flujo del funcionamiento de la aplicación de adquisición de datos para la ONA NX7.

En la Figura 5 se muestra el código necesario para establecer la conexión telnet, mandar una petición y la lectura de la respuesta de la máquina ONA NX7. Para la demostración desarrollada, la salida OUT se trata de la consola de Python.

```
import telnetlib

onaConnection = telnetlib.Telnet("192.168.4.65")

onaConnection.write("<XML><exec_data/></XML>")
respuesta = onaConnection.read_all()
print(respuesta)
```

Figura 5: Ejemplo de una implementación sencilla de telnetlib en Python para la comunicación con la máquina ONA NX.

En los resultados mostrados hasta ahora y debido a que son demostraciones de funcionamiento, el dispositivo de lectura de datos no ha sido el RELY-CPPS sino que ha sido un ordenador convencional conectado a la ONA NX7. Los datos se han almacenado en el propio ordenador que, además, por simplicidad es también el dispositivo por el que se muestran las lecturas. El software desarrollado permite la lectura tanto continua como puntual de datos. En la Figura 6 se muestra un ejemplo de la salida de datos leídos durante un intervalo de tiempo mediante el uso del script desarrollado por SoC-e. Se han adjuntado imágenes de la posición del cabezal de la máquina en cada uno de dichos instantes, y una gráfica mostrando la trayectoria seguida hasta cada instante. En caso de que fuese necesario, se dispone de un vídeo que muestra la actualización por pantalla de los valores leídos, la actualización de la gráfica y el movimiento de la máquina en tiempo real.

Los valores presentados por la aplicación concuerdan con los mostrados en el propio mando de la ONA NX7, por lo que se da por correcto el funcionamiento del programa.

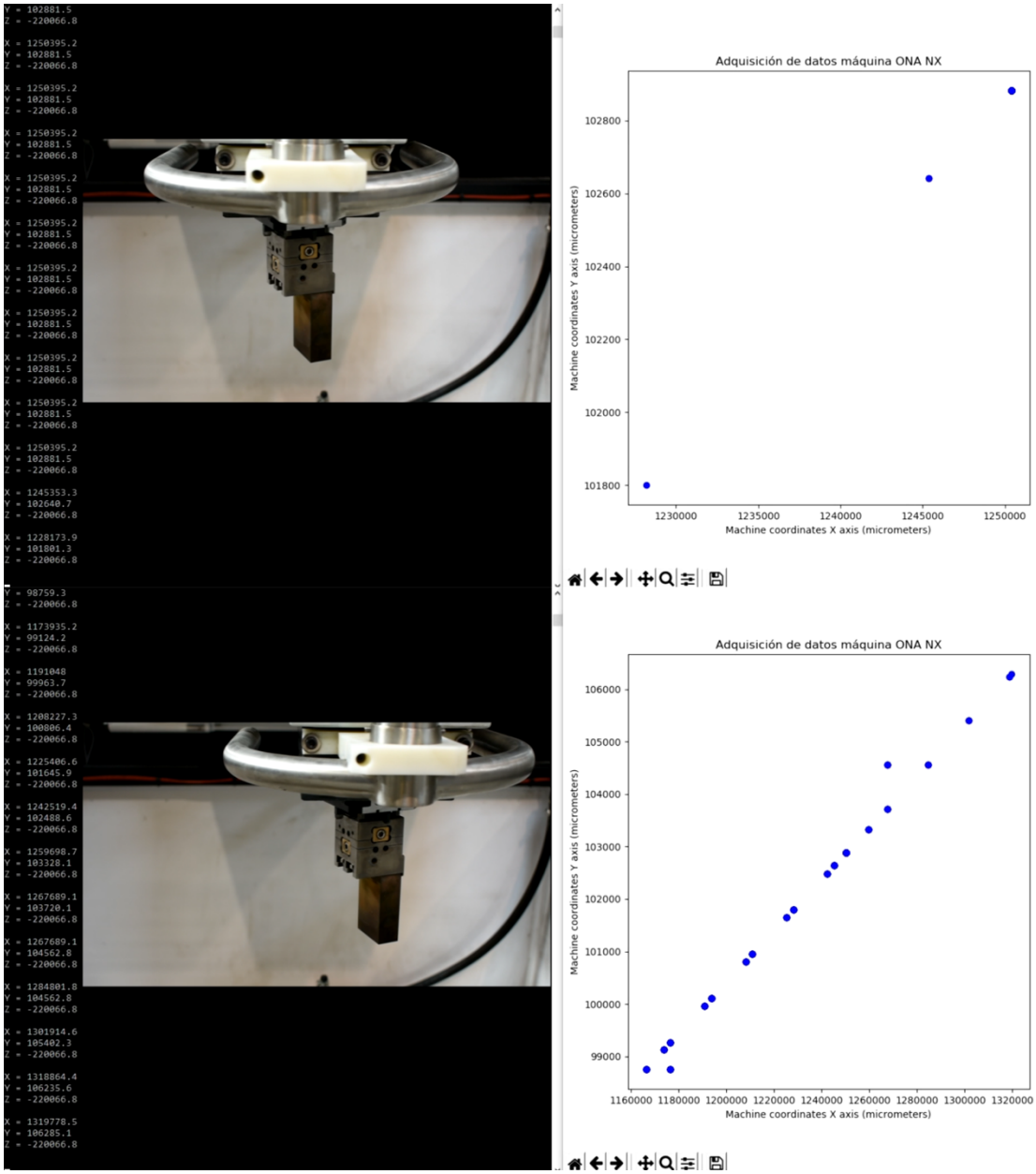


Figura 6: Izquierda: coordenadas XYZ en micrómetros leídas directamente del XML de respuesta y estado del cabezal. Derecha: Gráficas correspondientes a la trayectoria seguida hasta el instante correspondiente a la foto del cabezal.

El desarrollo aquí presentado ha servido para afianzar los conocimientos de la adquisición de datos en la máquina ONA NX7 y para demostrar la correcta implementación de dicha herramienta. No obstante, de este punto en adelante se utilizarán los servicios de Linux desarrollados por SoC-e, ya que presentan un grado de customización mayor y de fácil manejo por estar directamente implementados en el sistema operativo y permiten filtrar la lectura de variables, especificando las que se desean leer y las que se desean ignorar, así como la frecuencia con la que se leen las seleccionadas.

La implementación en el ordenador personal no es óptima, ya que en un caso de uso real supondría que hay un sólo punto de fallo con el que caería todo el sistema. Además, el ordenador personal ocupa un espacio grande de forma innecesaria (ya que se podría implementar en dispositivos con menor capacidad de procesado y, por lo tanto, menor huella). Es por ello por lo que el siguiente paso de este trabajo es tratar de modularizar y optimizar el sistema de lectura de datos.

El primer paso para la modularización del proceso es el uso del RELY-CPPS en vez del ordenador personal. Para ello, se conecta como se ha mencionado previamente a la ONA NX7 y se configuran los servicios de Linux para la toma de datos.

El siguiente paso es diferenciar el dispositivo de lectura de datos con el de almacenamiento de datos.

3. Almacenamiento de datos.

La selección de la base de datos para el almacenamiento de datos ha sido directa teniendo en cuenta el caso de uso que se desea a largo plazo. Ha de ser una plataforma que permita el acceso remoto de forma que se puedan guardar nuevos datos constantemente y además diferentes usuarios puedan acceder a dichos datos en tiempo real.

Por ello, la selección obvia es el uso de una base de datos relacional alojada en un servidor local [11]. Una base de datos relacional es una base de datos basada en el modelo relacional de la información. Dicho modelo consiste en organizar la información en tablas de filas y columnas, donde cada fila representa una entrada de datos, y las columnas de esa fila son los atributos que tienen esas entradas. Una de las columnas es lo que se denomina *clave única* que sirve para referenciar los datos de una tabla desde otra [12]. Por ejemplo, una tabla de una base de datos relacional con la información de los ciudadanos de un país podría utilizar como clave única el DNI, y los atributos serían nombre, apellidos, fecha de nacimiento, etc.

Con esta infraestructura se consiguen los requisitos propuestos. Además, permite no sólo el almacenamiento de datos provenientes de las máquinas monitorizadas, sino que también es viable la disposición de datos relativos a credenciales de los trabajadores (con las que posteriormente acceder a la plataforma de visualización de los datos) o del estado de los proyectos asociados a cada máquina y sus piezas.

Finalmente, la gestión de una base de datos relacional depende en gran parte de la herramienta utilizada para gestionarla. Un sistema de gestión de base de datos relacionales es un programa o lenguaje que permiten la creación y administración de tablas y bases de datos relacionales [11]. En el presente trabajo, se ha optado por el uso de MySQL.

MySQL es un sistema de gestión de bases de datos relacionales que permite (mediante MySQL server) el acceso remoto a dicha base de datos. Además, la relativa simplicidad del lenguaje para ejecutar comandos permite que la automatización de comunicaciones con y desde la base de datos sea trivial (hecho que será importante a la hora de desarrollar la herramienta de interacción entre el usuario y la base de datos) [13].

Además, la disponibilidad de MySQL Server para una gran variedad de sistemas operativos [14] proporciona una gran escalabilidad a la herramienta, permitiendo migrar la base de datos de un servidor a otro diferente independientemente del sistema operativo de cualquiera de ellos.

3.1. Desarrollo de la arquitectura software para el almacenamiento de datos.

El desarrollo aquí presentado consiste en la creación de una base de datos MySQL accesible desde la red local del Centro de Fabricación Avanzada Aeronáutica (CFAA) albergada en una Raspberry Pi 3 B+. Todo el trabajo desarrollado y presentado en este proyecto de fin de grado se ha hecho con la base de datos implementada en la Raspberry Pi. Sin embargo, se ha sabido desde un primer instante que la base de datos creada en la Raspberry Pi sería, en un futuro, migrada a un servidor diferente, por lo que se ha mantenido en mente en todo momento la generalización y la facilidad para la migración de la base de datos. El servidor al que se migrará será un Ubuntu Server en una red pilotada hacia 5G haciendo uso principalmente de dos tecnologías, Software-Defined Networking (SDN) y Network Function Virtualization (NFV). La red será desplegada por el grupo de investigación Smart Networks for Industry (SN4I) en el centro [15]. Mediante NFV se pueden crear diferentes instanciaciones de máquinas virtuales en el mismo servidor. Cada una de estas instanciaciones está completamente aislada del

resto tanto a nivel de conectividad como a nivel de memoria, maximizando así la seguridad de los datos almacenados (ya que si una instancia es comprometida, no afecta a las demás). SDN permite hacer túneles de red tanto entre máquinas virtuales como entre dispositivos reales, y también entre máquinas virtuales y dispositivos.

Así, inicialmente se generan todas las instancias necesarias para implementar todos los servicios que cumplen las necesidades del centro y después se crean las conexiones requeridas entre las instancias. Por ejemplo, se podrían almacenar las credenciales de los usuarios en una base de datos albergada en una máquina virtual que esté completamente aislada de la base de datos (en otra máquina virtual del mismo servidor) que contiene la información adquirida de las máquinas del centro, asegurando así que si se comprometen los datos no se comprometen las credenciales, y viceversa [16].
El proceso de despliegue de cada una de estas máquinas virtuales es largo e independiente al alumno, por lo que no se ha podido utilizar más de una instancia en este trabajo. Sin embargo, y como ya se ha comentado múltiples veces, migrar parte de la base de datos implementada a otras instancias (una vez éstas han sido creadas) es un proceso rápido y sencillo.

La base de datos no ha sido implementada desde un principio en la red de SN4I debido a la incertidumbre en la fecha en la que finalmente estaría instalada y accesible desde el centro. Por ello, se ha optado por iniciar el desarrollo en la Raspberry Pi y más tarde migrar la base de datos al servidor final.

3.2. Preparación de la Raspberry Pi y de su conexión a la red local.

En primer lugar, se ha instalado el sistema operativo Raspbian en la Raspberry Pi, permitiendo así la conexión remota mediante SSH [17]. SSH (Secure Shell) es un protocolo de control remoto de dispositivos mediante un canal encriptado [18]. Mediante SSH se puede controlar la Raspberry Pi desde la estación de trabajo personal, facilitando así su uso e instalación (ya que se puede prescindir de un monitor y dispositivos de entrada específicos para interactuar con la Raspberry Pi).

Para la configuración de la dirección IP de la Raspberry Pi, hace falta primero conocer los dos tipos de dirección IPv4 que se pueden establecer (para el uso que se le dará en esta aplicación, no será necesaria ninguna configuración de IPv6):

Por un lado, se puede configurar la dirección IP como **dinámica**. Que un dispositivo en una red tenga una dirección IP dinámica implica que cada vez

que se conecta a la red el router le asigna una dirección IP arbitraria dentro de la subred mediante el protocolo DHCP [19]. Esto hace que la conexión a determinadas redes sea automática y que el usuario no tenga que configurar nada. Sin embargo, para la aplicación de este trabajo no es adecuada, ya que cada vez que se quiera realizar una comunicación con la Raspberry Pi habría que utilizar una dirección IP diferente, imposibilitando así la automatización del proceso.

Por otro lado, se puede establecer una dirección IP **estática**. Consiste en determinar para un dispositivo una dirección IP constante en el tiempo, por lo que cada vez que se conecte a ese mismo router mantendrá la dirección. La desventaja que presenta este sistema es que la dirección seleccionada queda ocupada y no puede ser utilizada por otros dispositivos de la red ni adjudicada a dispositivos mediante DHCP.

Se ha asignado una dirección IP estática a la Raspberry Pi de forma que se pueda hacer siempre SSH desde un ordenador conectado a la red local y que conozca las credenciales establecidas en la Raspberry Pi. La dirección IP asignada es 10.109.71.90.

En la Figura 7 se puede observar la Raspberry Pi conectada en el Centro de Fabricación Avanzada Aeronáutica (CFAA).

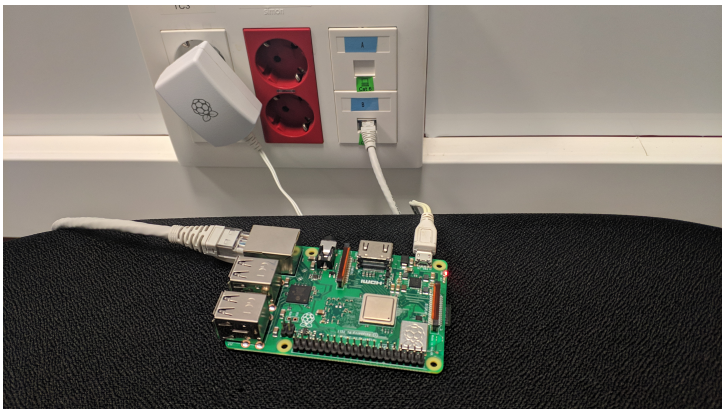


Figura 7: Raspberry Pi 3 B+ configurada y conectada en el CFAA.

3.3. Despliegue de la base de datos MySQL.

Se ha desplegado una base de datos MySQL en la Raspberry Pi. El nombre de dicha base de datos es *"CFAA"* y se ha establecido una contraseña del usuario *root* de modo que sólo los que la conozcan puedan acceder. Se han habilitado las conexiones remotas en modo lectura a la base de datos desde cualquier dirección IP, . Esta libertad de conexión está en realidad

restringida, ya que la base de datos sólo es visible desde la red local del centro.

El siguiente paso es la creación de las tablas que contendrán la información necesaria. Se han creado dos tablas dentro de la base de datos *'CFAA'* llamadas *'empleados'* y *'ona_nx'*.

La tabla *'empleados'* contiene las credenciales de los trabajadores del centro de forma que puedan acceder a la herramienta de presentación de datos. El root (a partir de ahora referido como administrador) de la base de datos tiene la potestad de introducir y eliminar entradas de la tabla. Los atributos de la tabla son los siguientes:

1. ID del empleado: este atributo es la **clave única** de la tabla. Consiste en un número entero identificador de cada empleado. Es del tipo `int(11)`.
2. Nombre del empleado: este atributo indica el nombre y apellidos del empleado asociado al ID correspondiente. Es del tipo `varchar(30)`.
3. Fecha de entrada: este atributo indica la fecha de incorporación del empleado asociado al ID correspondiente. Es del tipo `date`.
4. Fecha de salida: este atributo indica la fecha de terminación del empleado asociado al ID correspondiente. Es del tipo `date`.
5. Usuario: este atributo indica el email con el que el empleado asociado al ID correspondiente accederá a la herramienta de presentación de datos. Es del tipo `varchar(20)`.
6. Contraseña: este atributo indica la contraseña con la que el empleado asociado al ID y usuario correspondiente accederá a la herramienta de presentación de datos. Es del tipo `varchar(20)`.

```
mysql> desc empleados;
+------------------+-------------+------+-----+---------+-------+
| Field            | Type        | Null | Key | Default | Extra |
+------------------+-------------+------+-----+---------+-------+
| empleado_id      | int(11)     | NO   | PRI | NULL    |       |
| nombre           | varchar(30) | YES  |     | NULL    |       |
| fecha_de_entrada | date        | YES  |     | NULL    |       |
| fecha_de_salida  | date        | YES  |     | NULL    |       |
| usuario          | varchar(20) | YES  |     | NULL    |       |
| contraseña       | varchar(20) | YES  |     | NULL    |       |
+------------------+-------------+------+-----+---------+-------+
6 rows in set (0.01 sec)

mysql>
```

Figura 8: Descripción de la tabla *'empleados'*.

Esta disposición de la información permite un seguimiento básico de los empleados, tanto actuales como pasados. Además permite al administrador conocer los credenciales de forma que si hay algún problema con las sesiones de los usuarios se puedan solucionar remotamente sin necesidad de que el usuario esté presente.

Es evidente que el sistema implementado no es altamente seguro. Los datos no están encriptados por lo que cualquier brecha de seguridad los haría accesibles. Esta carencia no deberá existir cuando el sistema se implemente de manera definitiva, por lo que es una de las líneas futuras a desarrollar.

La tabla *'ona_nx'* contiene los valores de las variables leídas en la toma de datos. Están ordenados según la fecha y hora en la que se tomaron, no según la fecha en la que se añadieron a la base de datos. De esta forma, si la base de datos está temporalmente en mantenimiento y no permite la inserción de nuevas entradas en la tabla *'ona_nx'*, se pueden almacenar los datos localmente en el RELY-CPPS y volcarlos en la base de datos una vez vuelva a estar activa. Esto asegura que ningún dato se pierda, permitiendo así la creación de un historial de la máquina completo. Los atributos de la tabla son los siguientes:

1. Timestamp: Es un indicador de la fecha y hora en la que los datos fueron tomados. Es la **clave única** de esta tabla. Es proporcionado por el RELY-CPPS cuando se transfieren los datos a la Raspberry Pi. Es del tipo `timestamp`.
2. ID de la variable: Es el nombre de la variable leída en el timestamp correspondiente. También es proporcionado por el RELY-CPPS. Es del tipo `varchar(10)`.
3. Valor de la variable: Es el valor leído de la variable asociada al timestamp correspondiente. Es proporcionado por el RELY-CPPS y, por generalización, es del tipo `varchar(10)`. En función de lo que se desee realizar con los datos leídos (presentar o procesar) habrá que convertir esta cadena de texto en un valor del tipo necesario (int, float...).

```
mysql> desc ona_nx;
+-------------+-------------+------+-----+---------+-------+
| Field       | Type        | Null | Key | Default | Extra |
+-------------+-------------+------+-----+---------+-------+
| Timestamp   | timestamp   | NO   | PRI | NULL    |       |
| Variable_id | varchar(10) | YES  |     | NULL    |       |
| Value       | varchar(10) | YES  |     | NULL    |       |
+-------------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql>
```

Figura 9: Descripción de la tabla *'ona_nx'*.

De esta forma, la herramienta de presentación de datos podrá acceder a los datos ordenados cronológicamente, permitiendo generar histogramas sin necesidad de un procesamiento computacionalmente costoso y agilizando así la presentación. Además, la estructura de datos facilita el acceso a la última entrada añadida (la más reciente cronológicamente) en caso de que se quieran mostrar los parámetros de operación en tiempo real.

3.4. Implementación de la arquitectura software para el almacenamiento de datos

Para implementar la base de datos previamente descrita, se ha instalado MySQL en la Raspberry Pi y se han creado tanto la base de datos como las dos tablas manualmente. Se han poblado dichas tablas con información de prueba de cara a comprobar el correcto funcionamiento de la herramienta de presentación de los datos en el futuro. Con esto se tendría una base de datos accesible desde cualquier ordenador conectado a la misma red local que la Raspberry Pi. Sin embargo, el centro dispone de dos redes independientes. Una de ellas es la red local inalámbrica de Eduroam, accesible de la misma forma que se accede a la red inalámbrica de la UPV-EHU [20]. La otra es una red cableada independiente de Eduroam. A esta red se conectan los ordenadores y máquinas del centro. La Raspberry Pi corre un sistema operativo Raspbian, y la herramienta para la configuración de la red Eduroam en dicho sistema operativo no funciona correctamente, por lo que para tener conectividad a internet, la Raspberry Pi debe estar conectada por cable a la red local. A priori esto no presenta un impedimento, ya que la Raspberry Pi sería accesible desde los ordenadores del centro.
Sin embargo, la futura herramienta para la presentación de datos debe de ser accesible desde los smartphones de los técnicos y operarios, que sin duda estarán conectados a la red inalámbrica del centro.

La solución a este problema es clara, el smartphone ha de conectarse a un dispositivo intermedio que a su vez sea capaz de conectarse a la Raspberry Pi por la red cableada. La conexión al dispositivo intermedio ha de ser inalámbrica. Una posibilidad sería habilitar un ordenador que actúe como punto de acceso (de aquí en adelante referido como *Hotspot*) continuamente. Este *Hotspot* no es más que un punto de acceso WiFi. La conexión a esta red WiFi virtual hace que el dispositivo móvil pueda conectarse a la Raspberry Pi a través de la red cableada. Sin embargo, esto complicaría la programación de la herramienta, ya que el ordenador debería de procesar las peticiones de la herramienta, comunicárselas a la Raspberry Pi, que la Raspberry Pi procese la petición y la devuelva al ordenador intermediario y finalmente que este procese la respuesta y se la devuelva al dispositivo correspondiente. Es decir, el ordenador haría las funciones de multiplexor

de solicitudes a la Raspberry Pi proporcionando, y además, ser el túnel por el que conectarse.
Sin embargo, existe un solución similar pero más sencilla. En vez de utilizar un ordenador intermedio, se puede crear un *Hotspot* directamente en la Raspberry Pi. Esto simplifica enormemente el desarrollo necesario para la conectividad a la base de datos, ya que la conexión es directa entre el smartphone y la Raspbery Pi, por lo que las peticiones van directas al servidor MySQL. Además, se elimina la necesidad de tener un seguimiento de las peticiones y respuestas a más de un cliente que necesitaría tener el sistema que implementa un ordenador intermedio para que los usuarios no reciban respuestas que corresponden a otros usuarios. Esto es debido a que, como se verá más adelante, la conexión a la base de datos se hace (aunque indirectamente) mediante un *Socket*[1], el cuál representa una única conexión entre el dispositivo móvil y el servidor (en este caso, la Raspberry Pi).

3.5. Creación del *Hotspot* en la Raspberry Pi.

El *Hotspot* se ha implementado mediante el software RaspAP WebGUI [22]. Esta herramienta permite configurar de forma sencilla un punto de acceso en la Raspberry Pi corriendo Raspbian. La configuración se hace mediante la interfaz web de RaspAP. Los parámetros relevantes para la presente implementación son la dirección IP y la contraseña del punto de acceso. Anteriormente se ha mencionado la dirección IP estática asignada a la Raspberry Pi en su configuración inicial. Esta dirección IP estática es la dirección IP de la Raspberry Pi para la comunicación con el router del CFAA, que hace que la Raspberry Pi tenga conexión a internet. La dirección IP que se mencionará a continuación es diferente a la anterior, ya que al actuar la Raspberry Pi como un *Hotspot*, ésta se comporta esencialmente como otro router (que a su vez estaría conectado al router del CFAA) y requiere de una dirección IP adicional. Así, la Raspberry Pi tiene dos direcciones IP asociadas: la de comunicación entre la Raspberry Pi y el router del CFAA, y la de comunicación entre la Raspberry Pi y los dispositivos conectados al *Hotspot*. Para el *Hotspot* se ha mantenido la dirección IP establecida de forma predeterminada, 10.3.141.1. Sin embargo, la inserción de contraseña se ha anulado de forma que cualquiera que esté en el centro dentro del alcance del punto de acceso pueda conectarse. Aunque esto pueda parecer un fallo de seguridad, se ha de tener siempre en mente que la capa de seguridad en la que se evita que agentes externos al centro se conecten a la base de datos se tiene en cuenta en la misma aplicación explicada más adelante, donde

[1] Un *Socket* en Java representa un punto de conexión entre dos programas corriendo en una red. Los objetos *Socket* se usan para controlar la comunicación entre el servidor y cliente, y permiten comenzar y terminar operaciones de transmisión de datos [21].

usuarios deberán introducir sus credenciales. Así, anulando la contraseña se simplifica el proceso de distribución y acceso a la herramienta de monitorizado mejorando así la experiencia de usuario y manteniendo un grado de seguridad aceptable con las credenciales de los trabajadores.

Una vez realizada la configuración y reiniciada la Rasberry Pi, el punto de acceso es visible desde los smartphones dentro del rango de cobertura de la Raspberry Pi con el SSID "*raspi-webgui*", como se puede observar en la Figura 10.

Una vez se ha implementado el punto de acceso y la base de datos en la Raspberry Pi falta desarrollar la aplicación que presenta los datos.

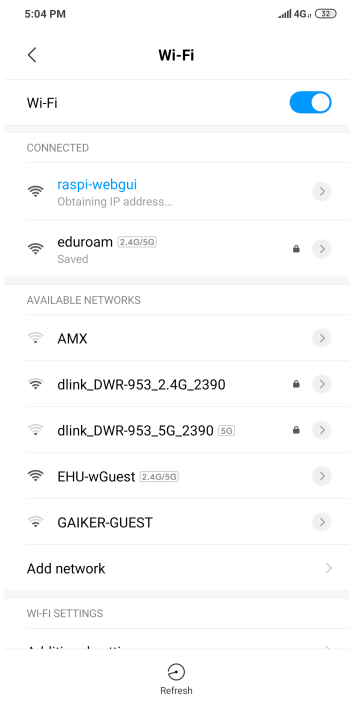


Figura 10: *Hotspot* de la Raspberry Pi visible desde el smartphone.

4. Visualización de datos.

La presentación de los datos de forma útil y clara es tan importante como la propia toma y estructuración de éstos. Si el usuario no es capaz de acceder de forma rápida y sencilla a estos datos, la disponibilidad de los mismos pierde valor . Por ello, se ha de elaborar una solución de acuerdo a estas exigencias.

El desarrollo a realizar será el de una aplicación móvil debido a la movilidad y escalabilidad que ofrece. Además, presenta un desafío en comparación a la herramienta implementada en el ordenador, ya que la programación para el ordenador es un proceso conocido y trabajado en el grado, mientras que el desarrollo de aplicaciones móviles es una técnica nueva.

La aplicación se desarrollará para la plataforma Android por dos razones. En primer lugar, al ser una plataforma abierta los costes de desarrollo son nulos. Por otro lado, el uso del sistema operativo Android es el más extendido entre los trabajadores del centro, de forma que la herramienta desarrollada estará disponible para el mayor número de operarios posibles.

4.1. Desarrollo del software para la presentación de datos

Para el desarrollo de la aplicación se ha utilizado el IDE Android Studio [23]. Las aplicaciones Android se programan en un híbrido entre Java y XML. La arquitectura de una aplicación Android, al igual que cualquier programa que interactúa con un usuario, se puede dividir en dos partes, el *front end* y el *back end*. El *front end* es todo aquello que interactúa con el usuario, como la interfaz o el teclado. El *back end* es la lógica detrás del programa, los módulos, clases, etc. que se ejecutan para actualizar el *front end* en función de lo que el usuario haga.
En una aplicación Android, el *front end* se configura mediante XML y el *back end* se programa mediante Java [24]. Por ello, es esencial mantener un código completamente modularizado, de forma que los accesos a la información generada por el *front end* mediante el *back end* y viceversa sean eficientes y claros.
Como se puede intuir, el *front end* y el *back end* están fuertemente relacionados, por lo que en vez de explicarlos por separado se explicarán en conjunto para cada actividad.

Las actividades en Android son las diferentes etapas de una aplicación. Por ejemplo, una aplicación que permite leer el correo electrónico podría tener las siguientes actividades:

- Actividad de log-in.
- Actividad de selección de correo a leer.
- Actividad de lectura de correo.
- Actividad de escritura de correo.

Cada actividad tiene una interfaz diferente (por lo que interactúa con el usuario de diferente forma, de ahí el nombre de Actividad). Una actividad puede presentar más de una posibilidad de uso. Por ejemplo, la actividad de lectura de correo permite leer el correo seleccionado, pero también dará la opción de reenviar o responder a dicho correo. Las transiciones entre actividades se hacen mediante el *back end*. Si la aplicación se encuentra en la actividad de selección del correo a leer y el usuario selecciona un mensaje, el *front end* debe comunicar al *back end* que se ha de cambiar de actividad a la de lectura de correo, y que se ha de leer específicamente el correo seleccionado.

Esta comunicación se hace mediante Listeners. Los listeners son clases de Java cuyas instancias son asociables a elementos de la interfaz (que también son instancias de las clases que generan la interfaz). Esta asociación funciona de la siguiente manera: se genera un elemento de la interfaz con el que el usuario puede interactuar, como por ejemplo un botón. Después, se asocia un objeto Listener a este botón. Una vez hecho esto, cuando ocurra un evento en el botón (por ejemplo, que sea pulsado) el Listener ejecutará el código que el programador ha determinado en la función correspondiente a dicho Listener.
En concreto, cuando se implementa un Listener siempre se ha de sobreescribir (con un *@Override*) la función que se ejecuta al interactuar con el elemento de la interfaz que implementa dicho Listener. Dependiendo del elemento al que se asociará un Listener, éste será de un tipo u otro. Por ejemplo, para un botón *Toggle* con dos estados, activado y desactivado, se ha de sobrescribir la función *onCheckedChanged*, que es la que se ejecuta cuando se cambia de un estado a otro. Para botones seleccionables, se ha de sobrescribir la función onClick, etc.

Las actividades se representan mediante clases Java y, por lo tanto, permiten ser usadas como tal. Por ello, se pueden declarar campos, constructores y sus instancias pueden ser pasadas como argumentos, etc. Siempre se ha de sobrescribir el método *onCreate* con el código que se ejecuta al crear una instancia de esta clase. En concreto, en el caso de que un cambio de actividad suponga un cambio de interfaz, se ha de especificar el fichero XML que contiene la configuración de dicha interfaz. Se carga la interfaz y se presenta al usuario por pantalla. El resto del código consiste en poner a punto todos los elementos de la interfaz (por ejemplo, asignándoles Listeners) de forma que sea interactuable o presente la información que deba de presentar.

Para pasar de una actividad a otra se utiliza la función `startActivity()` que toma como argumento una instancia de la clase *Intent*. La clase *Intent* y sus instancias describen de forma abstracta una operación a realizar. Para utilizarlos para el cambio de una actividad a otra, se ha de generar un objeto

Intent de la clase a la que se desea pasar. El constructor de la clase *Intent* toma dos argumentos de entrada. En primer lugar, el contexto actual de la aplicación (la clase en la que está descrita la actividad actual) y la clase en la que se describe la actividad a la que se quiere transicionar. Este objeto es el que se utilizará con `startActivity()` y representa el cambio específico entre ambas clases (actividades).

En la Figura 11 se muestra un esquema del proceso explicado.

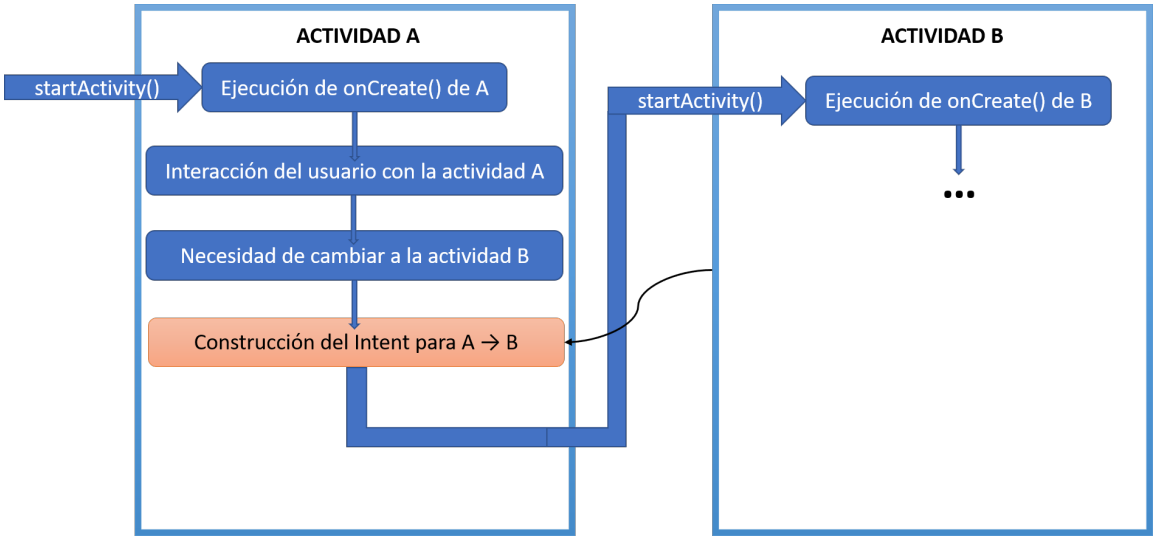


Figura 11: Esquema del proceso de transición entre actividades.

Finalmente, hace falta mencionar un aspecto muy importante de la programación para Android. Cuando se tiene una aplicación que ha de ejecutar funciones continuamente, como por ejemplo presentar los datos más recientes adquiridos de la base de datos en pantalla, mientras a la vez espera input del usuario y hace otras tantas operaciones necesarias para mantener la actividad en funcionamiento, es lógico pensar en utilizar hilos para paralelizar dichas operaciones. De hecho, el uso de hilos es la única manera de implementar la aplicación aquí presentada. El uso de hilos impone la obligación de tener un código correctamente modularizado y claro, de forma que no se generen infinitos hilos o *daemons*[2] innecesarios, y que en caso de crear un daemon y después desecharlo, el sistema no de errores.
Por otro lado, el sistema operativo Android introduce una restricción en los hilos que pueden hacer modificaciones en la interfaz: tan sólo el hilo que ha creado la interfaz de usuario (el llamado *UI Thread*) puede interactuar con la interfaz. Esta será una limitación a tener en cuenta a la hora de presentar los datos.

[2]Un *daemon* es un hilo que nunca termina, se ejecuta infinitamente.

A continuación, se explicará la estructura de la aplicación desarrollada.

4.2. Aplicación CFAA Fully Connected.

La aplicación desarrollada se llama *CFAA Fully Connected*. Funciona de la siguiente manera: una vez iniciada la aplicación, se piden al usuario las credenciales (email y contraseña) así como una dirección IP. Esta dirección IP es la dirección IP de la base de datos (o de la Raspberry Pi que interactúa con la base de datos) y, dejando este campo en 0.0.0.0, se conecta a la dirección IP por defecto de la Raspberry Pi: 10.3.141.1.

Una vez iniciada la sesión, se muestra el listado de máquinas del taller del CFAA. Seleccionando el nombre de cada máquina, se accede a la pantalla en la que se muestran los datos de dicha máquina. Las máquinas con las que se puede interactuar en esta pantalla son las que están monitorizadas, las demás son marcadores de posición que fácilmente se podrían convertir en interactuables en el futuro.

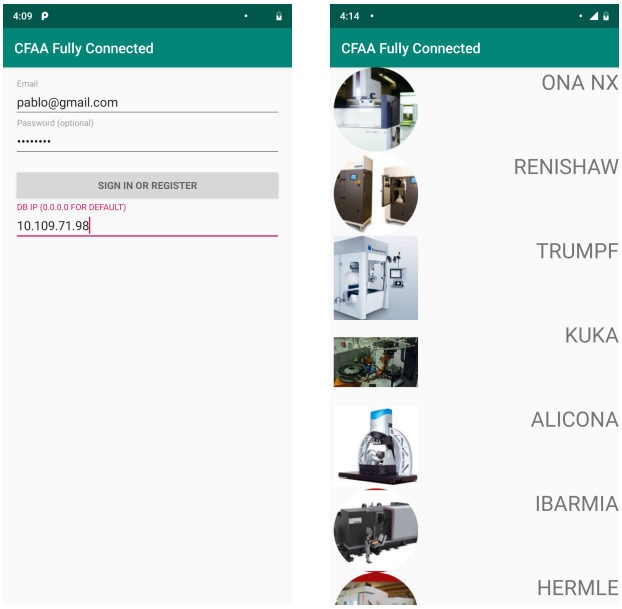
Finalmente, se muestra la pantalla de la máquina seleccionada. En esta pantalla se puede activar o desactivar la muestra de datos mediante un botón interruptor. Cuando la muestra de datos está activada, éstos se muestran en un panel reservado para la muestra de texto. En la Figura 12 se puede observar la apariencia de las diferentes pantallas de la aplicación.

Para el desarrollo mencionado, se han usado 6 clases (3 actividades y 3 clases convencionales) con diferentes métodos que se explicarán a continuación.

Las clases y actividades permiten el inicio de sesión, la selección de máquina y la muestra de los datos asociados a esa máquina en tiempo real desde la propia aplicación. Las 3 actividades son `LoginActivity`, que es la clase que coordina el inicio de sesión, `DataActivity`, que es la clase que coordina la selección de máquina y `OnaNXActivity` que es la clase que coordina la muestra de datos de la ONA NX7.
Por otro lado, las clases auxiliares son `SQLConnection`, que es una abstracción que permite, de forma sencilla, interactuar con una base de datos MySQL, `ConnectionData`, que sirve para almacenar la información de conexión a la base de datos y `ToggleLiveUpdatesListener` que es una clase que coordina la obtención de datos cuando el usuario lo solicita.

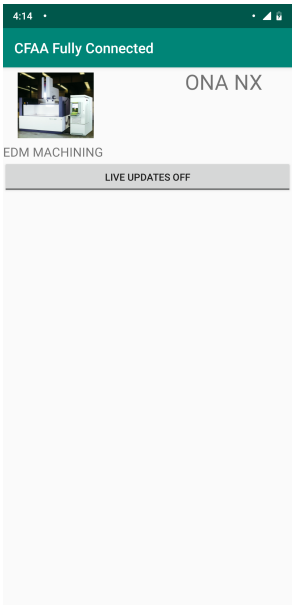
4.2.1. Clase SQLConnection

Para interactuar con la base de datos se ha creado una clase llamada SQLConnection. Esta clase se sirve de la API **Java Database Connectivity**

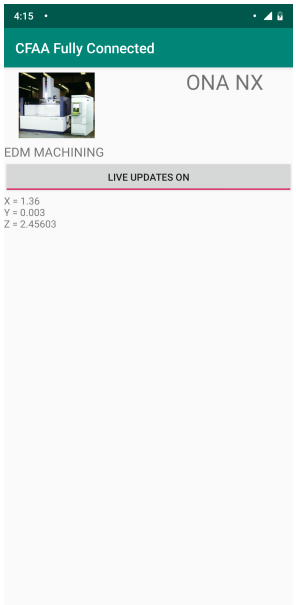


(a)

(b)



(c)



(d)

Figura 12: Actividades de la aplicación CFAA Fully Connected. (a) Actividad de log in. (b) Actividad de selección de máquina. (c) Actividad de muestra de datos OFF. (d) Actividad de muestra de datos ON.

(JDBC). JDBC proporciona los métodos necesarios para establecer una conexión con una base de datos remota e interactuar con ésta [25], así como procesar los resultados devueltos en una *query*[3]. JDBC está contenido en el paquete *java.sql* por lo que no hace falta ninguna instalación adicional a la básica de Java.

Para interactuar con una base de datos mediante JDBC hace falta crear una instancia de la clase Connection. Esta instancia representa la conexión con la base de datos. Para instanciar el objeto Connection hace falta conocer la dirección IP de la base de datos, el nombre de la base de datos y el nombre de usuario y contraseña con el que iniciar sesión en la base de datos. Una vez se tiene esa información, se debe cargar el driver de JDBC. Este driver se encarga de utilizar el protocolo correspondiente a la conexión que se quiere entablar con la base de datos específica[4].

Con el driver cargado, tan sólo hace falta llamar a la función `DriverManager.getConnection()` pasando los argumentos URL, USUARIO y CONTRASEÑA, todos como Strings. Esta función devuelve un objeto Connection que es el que representa la conexión con la base de datos especificada en URL. En el caso aquí presentado, la base de datos a la que se conectará la aplicación es MySQL, por lo que la URL ha de tener la forma siguiente:

```
URL = "jdbc:mysql://DIRECCION_IP/NOMBRE_DB"
```

La clase SQLConnection desarrollada para la aplicación Android consiste en abstraer, aún más, la interacción con la base de datos específica del CFAA. Para ello presenta 3 métodos:

- `SQLConnection()`: Este método es el constructor, y toma como argumento de entrada tan sólo una String con la dirección IP de la base de datos. Devuelve un objeto SQLConnection con el que interactuar con la base de datos mediante los otros dos métodos. El método instancia el campo Connection del objeto SQLConnection como de la forma que se ha explicado anteriormente. Tanto la contraseña, el usuario y la dirección al driver de JDBC están introducidas a mano como campos

[3]Una *query* es una petición a la base de datos, ya sea para añadir entradas a las tablas o para hacer lecturas de éstas.

[4]En concreto, JDBC no utilizará los mismos módulos ni protocolos para conectarse a una base de datos Oracle o MySQL, por ejemplo. Por ello, la interfaz con JDBC está varios niveles de abstracción por encima de lo que en realidad sucede a nivel protocolos de conexión, permitiéndo el uso de la clase JDBC como una herramienta para interactuar con bases de datos en general, sin importar la arquitectura de éstas.

de la clase. La dirección IP se utiliza para construir la URL de la base de datos.

- `executeQuery()`: Este método permite ejecutar una *query* en la base de datos a la que el objeto SQLConnection está conectado. El único argumento de entrada es una String que contiene el comando MySQL a ejecutar en la base de datos (el *query*). El funcionamiento es el siguiente: inicialmente se crea un objeto de la clase Statement asociado a la conexión Connection del objeto SQLConnetion. Un Statement no es más que una representación de un comando a ejecutar en la base de datos asociada a un objeto Connection. A continuación, se crea un objeto ResultSet que contiene los resultados de la ejecución del *query*. Si el *query* recogía datos, ResultSet contendrá dichos datos. Si el *query* introducía datos, ResultSet no contendrá información. La creación del Statement se hace mediante el método `this.connection.createStatement()`[5]. La ejecución del comando MySQL se hace al crear el objeto ResultSet mediante el comando `statement.executeQuery()` que devuelve dicho ResultSet y ejecuta el *query*. El *query* se pasa como argumento a esta función mediante un comando MySQL en una String.

- `closeConnection()`: Este método no toma argumentos de entrada y tampoco devuelve ningún objeto (es un método *void*). Simplemente ejecuta el método `this.connection.close()` para cerrar la conexión con la base de datos. `this.connection.close()` tampoco toma argumentos de entrada y también es un método *void*.

Así, conociendo tan sólo la dirección IP en la que está alojada la base de datos MySQL se puede ejecutar todo tipo de *query* mediante la clase SQLConnection. Esto será de gran ayuda tanto para aligerar el resto del código de la aplicación como para la búsqueda de errores en el código (*debugging*).

Un detalle importante a destacar es el siguiente. En Android Studio se puede correr la aplicación programada en una máquina virtual de Android, que simula el sistema operativo e incluso la interfaz e interacción con éste. Se podría pensar, de forma intuitiva, que si se desea establecer una conexión a una base de datos alojada en el servidor local, es decir, en el propio ordenador en el que se está programando la aplicación, la dirección IP a introducir sería `localhost`. Sin embargo, esto resulta en errores a la hora de conectarse al servidor desde la máquina virtual de Android. La dirección IP que representa una conexión al servidor local es 10.2.2.2.

[5] `this` representa el objeto SQLConnection. `this.connection` representa el objeto Connection que es atributo de nuestro objeto SQLConnection. El método `createStatement()` es un método dinámico de los objetos Connection.

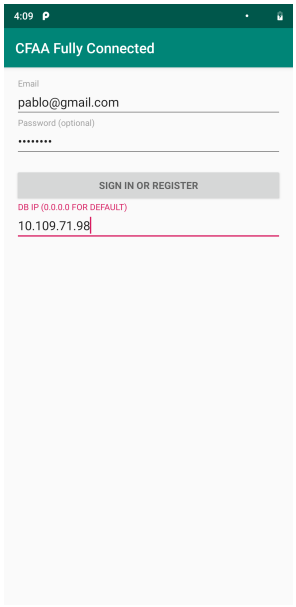
La clase SQLConnection se utilizará para toda interacción con la base de datos del CFAA. En la Figura 13 se puede observar el código de esta clase.

```java
package com.example.android.cfaafullyconnected;
import java.sql.*;

public class SQLConnection {

    //  Database URL: siempre es jdbc:mysql://DIR_IP:PORT/DB_NAME
    //  Para la dirección IP, usar 10.2.2 para localhost.
    static String URL;
    static final String USER = "root";
    static final String PASS = "CFAA";
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";

    private Connection conn;

    // CONSTRUCTOR

    public SQLConnection(String IP) throws ClassNotFoundException, SQLException{
        // Carga del driver de JDBC
        Class.forName(JDBC_DRIVER);
        // Construcción de la URL
        URL = "jdbc:mysql://"+IP+"/cfaa";
        // Construcción del objeto Connection
        this.conn = DriverManager.getConnection(URL, USER, PASS);
    }

    // EJECUTAR Y DEVOLVER RESULTADOS DE QUERY
    public ResultSet executeQuery(String query) throws SQLException{
        // Creación del statement
        Statement statement = this.conn.createStatement();
        // Ejecución del Query
        ResultSet rs = statement.executeQuery(query);
        return rs;
    }

    public void closeConnection() throws SQLException{
        this.conn.close();
    }

}
```

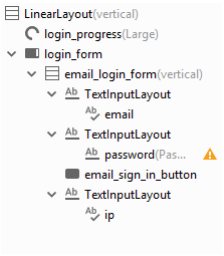
Figura 13: Código de la clase SQLConnection.

4.2.2. Clase LoginActivity

La clase LoginActivity es la clase que representa la primera Actividad de la aplicación. Esta clase contiene tanto la interacción con la interfaz como con la clase SQLConnection para ejecutar la validación de credenciales y el inicio de sesión en la aplicación. Además, es desde esta actividad desde donde se lanza la siguiente actividad de selección de máquina en caso de que el login sea válido.
Parte del código de esta clase como de la interfaz que la apoya se ha tomado de la plantilla preprogramada disponible en Android Studio. En concreto, se ha modificado la plantilla preprogramada para que el login se haga contra la base de datos y las credenciales ahí almacenadas, y para que aparezca un nuevo campo de entrada de texto en el que se pueda introducir la dirección IP de la base de datos a la que conectarse.



(a)



(b)

Figura 14: Interfaz y esquema de los componentes en el fichero XML de la actividad LoginActivity. (a) Interfaz de la actividad LoginActivity. (b) Árbol de componentes de la interfaz de LoginActivity

El elemento principal de la interfaz es un LinearLayout vertical. Un layout es un elemento que dispone a sus subelementos en un orden y colocación especiales. El LinearLayout dispone a cada uno de los elementos seguido del anterior. En el caso vertical, cada elemento está debajo del anterior, mientras que en el horizontal, cada elemento estará a la derecha del anterior.

Dentro del LinearLayout hay un elemento ProgressBar, que lo único que hace es dibujar un círculo rotando en pantalla para indicar que se está procesando algo. Cuando el ProgresBar se muestra, los demás elementos se dejan de mostrar, por lo que no afecta a la disposición normal de los elementos dentro del LinearLayout.

A continuación hay un ScrollView nombrado *login_form*. ScrollView simplemente indica que el usuario puede desplazarse verticalmente entre todos los elementos que están contenidos dentro de ScrollView. Normalmente ScrollView se utiliza en combinación con un LinearLayout vertical directamente dentro de él, de forma que cuando el usuario hace scroll, todos los elementos, ordenados, suben o bajan.
Por ello, después del ScrollView hay otro LinearLayout vertical, llamado *email_login_form* donde ya están contenidos los campos de introducción de texto y los botones.
Los tres elementos de introducción de texto están contenidos en tres TextInputLayout diferentes. TextInputLayout simplemente es un Layout que permite formatear con textos flotantes los campos de introducción de texto. Por simplicidad, se ha dejado el formato de los textos flotantes predeterminados por la plantilla de LoginActivity[6].
El TextInputLayout de introducción de email, al igual que el de introducción de dirección IP, contienen un elemento AutoCompleteTextView. Este elemento es un campo en el que se muestra el texto introducido por el usuario, que es accesible por el programa (en este caso, por la clase LoginActivity) y además abre un desplegable con textos introducidos anteriormente, que permiten autocompletar la introducción de texto. En el caso de la contraseña, se utiliza un campo Password, que es un TextView que actúa igual que el AutoCompleteTextView, sin embargo, no permite autocompletar el texto y además oculta los caracteres escritos.
Finalmente, hay un elemento Button que se llama *email_signin_button*.

Todos los elementos presentados son accesibles desde la clase LoginActivity, de forma que en cualquier método se puede acceder a la información contenida en ellos. El código de la plantilla utilizada ha sido programado por Google, por lo que no se explicará en detalle. Simplemente se mencionarán los aspectos indispensables para la comprensión de los cambios realizados y líneas añadidas.

Como cualquier actividad, LoginActivity tiene un método `onCreate()` que se ejecuta al crear la actividad. Al ser esta la actividad principal, la actividad se crea al abrir la aplicación, por lo que el método `onCreate()` se ejecuta también al iniciar la aplicación. Este método genera la interfaz a mostrar, y asigna las instancias del campo de introducción de email, contraseña y dirección IP. Además, asocia un Listener al botón de inicio de sesión, de forma que cuando éste sea seleccionado, se ejecute su método `onClick()`

[6]El formato es un texto de color rojo que aparece cuando el texto introducido no cumple con algún requisito programado o cuando se está interactuando con el campo de introducción de texto. En la Figura 14 a) se puede observar dicho texto formateado en el campo de la dirección IP. Las condiciones de la plantilla para dar error son la falta del caracter '@' en el email introducido y una contraseña con menos de 4 caracteres.

que simplemente llama al método `attemptLogin()`. Éste comprueba que no haya errores en la introducción del email y contraseña. En caso de que los haya, muestra el error en cada campo y permite la reinserción de los credenciales. Si los credenciales eran correctos, ejecuta, en un hilo paralelo, el proceso de autetificación, que está contenido en el método `execute()` de los objetos de la clase UserLoginTask, que está definida también dentro de esta clase.

La clase UserLoginTask extiende a AsyncTask y, mediante la sobreescritura del método `doInBackground()`, permite ejecutar una tarea de autentificación. En el constructor de UserLoginTask se pasan dos argumentos como Strings, el email y la contraseña, y se asignan a los campos de esta clase. Es por ello que cuando se crea un objeto UserLoginTask en el método `attemptLogin()` se leen los caracteres introducidos en los campos de texto para el email y la contraseña, y con ellos se genera el objeto de UserLoginTask.

Cuando se llama a `execute()` se ejecuta `doInBackground()`.
En `doInBackground()` se lee la dirección IP del campo de introducción de texto. Si esta dirección IP es 0.0.0.0, se utiliza la dirección IP por defecto (10.3.141.1). En caso de que no sea 0.0.0.0, se utiliza la introducida por el usuario. También se crea un objeto Boolean llamado *success* que inicialmente es *False*. Este Boolean es un indicador de si la autentificación ha sido exitosa o no. A continuación, se genera un objeto de la clase SQLConnection utilizando esta dirección IP, y también se genera un String que contiene la *query* a realizar. Esta *query* utiliza la sintaxis de MySQL y su construcción tiene la siguiente forma:

```
String query = "SELECT contraseña FROM empleados WHERE
               usuario=" + mEmail + ";";
```

En concreto, se accede a la tabla empleados de la base de datos CFAA y se recoge la contraseña que coincida con el email introducido por el usuario de la aplicación. Se ejecuta este *query* con `executeQuery()` y se procesa el ResultSet obtenido para extraer la contraseña. Si la contraseña coincide con la introducida por el usuario, se cambia el valor del Boolean *success* a *True*. Si no, se mantiene en *False*. Finalmente, se devuelve la variable *success*.

Después de ejecutarse `doInBackground()` siempre se ejecuta `onPostExecute()` si ha sido sobreescrita. Esta función toma como parámetro de entrada un Booleano que es el que devuelve `doInBackground()`. Por ello, una vez que se ejecuta `onPostExecute()` se sabe si el login ha sido exitoso o no. Si ha sido exitoso, se lanza la siguiente actividad. Esto se hace mediante la creación del Intent de la actividad DataActivity, que será explicada más adelante.

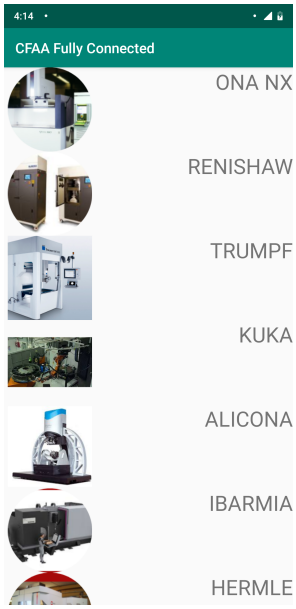
Una vez creado el Intent de dicha actividad, se llama a `startActivity()` y la aplicación cambiará de actividad. En caso de que el login no haya sido exitoso, se lanza un error en el campo de la contraseña.

4.2.3. Clase ConnectionData

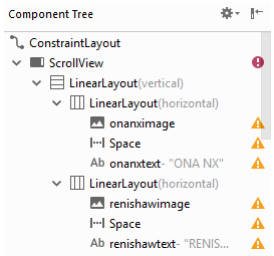
La clase ConnectionData es una clase auxiliar simple. Simplemente sirve para almacenar los datos de la conexión que se establece en la actividad LoginActivity. Consta de dos campos estáticos. Uno de ellos es una String y representa la dirección IP de la base de datos. El otro es un objeto SQLConnection y representa la conexión a dicha base de datos. Consta de cuatro métodos, dos *setters* y dos *getters* que permiten asignar valores a los dos campos estáticos y leer dichos valores. La razón por la que se ha implementado esta clase es para simplificar el acceso a la conexión a la base de datos. Cuando la actividad LoginActivity intenta hacer login y obtiene la dirección IP y una conexión a esa base de datos, llama a los *setters* de la clase ConnectionData de forma que cualquier otra clase de esta aplicación pueden tener acceso a la conexión con la base de datos. Esto es debido a que la aplicación se congela y se cierra automáticamente cuando se intenta conectar dos veces a la misma base de datos MySQL. Utilizando esta clase como almacenador de la conexión inicial, se evita el tener que generar una nueva conexión cada vez que se desea interactuar con la base de datos, eliminando así el error mencionado.

4.2.4. Clase DataActivity

La actividad DataActivity es la que se inicia después de un inicio de sesión exitoso. Presenta la lista de máquinas que se pueden seleccionar para acceder a su presentación de datos. A medida que se vayan monitorizando más máquinas, aumentará el número de máquinas a las que acceder. En la versión actual de la aplicación, aparecen todas las máquinas del CFAA pero tan sólo la ONA NX es accesible, por lo que el resto está tan sólo como prueba de concepto.



(a)



(b)

Figura 15: Interfaz y esquema de los componentes en el fichero XML de la actividad DataActivity. (a) Interfaz de la actividad DataActivity. (b) Árbol de componentes de la interfaz de DataActivity

La interfaz tiene como layout principal un ConstraintLayout. Este layout permite colocar los elementos dentro de él en relación uno con el otro, es decir, definiendo condiciones que se han de cumplir. Dentro del ConstraintLayout se encuentra un ScrollView que a su vez contiene un LinearLayout vertical. Esto permite hacer scroll en los contenidos del LinearLayout, que serán las imágenes y texto de cada máquina (propiamente organizadas). Este LinearLayout vertical contiene, a su vez, tantos LinearLayout horizontales como máquinas. De esta forma, se pueden tener regiones horizontales apiladas una encima de otra (las regiones horizontales corresponden al LinearLayout horizontal y las verticales al vertical) y se puede desplazar la pantalla mostrando las regiones que no caben.

Dentro de cada LinearLayout horizontal, hay tres elementos: una imagen que se corresponde con la máquina del CFAA, el texto con el nombre de la máquina (en forma de TextView) y un espacio entre la imagen y el texto. Se ha elegido el TextView para poner los nombres de máquinas por dos razones, por un lado, es un elemento simple y que no permite, por defecto, la edición del texto que muestra por el usuario. Por otro lado, a los TextView se les puede asociar un Listener del tipo OnClickListener, que ejecuta el código contenido en la función `onClick()` sobreescrita por el usuario, que se ejecuta cuando se selecciona el texto.

El código de la clase DataActivity es simple. El método `onCreate()` simplemente inicializa la interfaz y asigna a cada TextView de cada máquina el OnClickListener previamente mencionado. Se asigna el mismo Listener a todos los TextViews. Para sobreescribir el método `onClick()` del OnClickListener, se ha hecho mediante una clase local. Una clase local no es más que una clase que ha sido definida dentro de un método convencional de Java, a diferencia de las clases normales que se definen independientes del resto de métodos.

El método `onClick()` toma como entrada automáticamente el objeto View en el que se ha seleccionado. En el método, se castea[7] el objeto View a TextView y se obtiene el texto que se muestra en el TextView. A continuación, se compara el texto obtenido con los correspondientes a los TextViews de diferentes máquinas, y se lanza la actividad correspondiente a la máquina seleccionada.

En el caso aquí presentado, y como ya se ha mencionado, tan sólo se ha implementado la monitorización de la ONA NX, por lo que sólo es accesible la actividad asociada a ella. Cuando todas las máquinas estuviesen monitorizadas, se debería implementar un `switch` que compare la string obtenida con todas las strings posibles. El código de la definición de la clase OnClickListener se muestra en la Figura 16.

[7]Castear es cambiar el tipo de un objeto. En este caso, `onClick()` toma el tipo View, que es más general que TextView. Sin embargo, se conoce que únicamente objetos TextView pueden hacer saltar este método ya que sólo se ha asignado el Listener a objetos TextView, por lo que el casteo es seguro. En otros casos, puede saltar un error al tratar de convertir entre tipos incompatibles.

```
View.OnClickListener clickLS = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        TextView clickedTextView = (TextView) v;

        // OBTENER EL STRING DEL TEXTO CLICKADO

        String machineName = clickedTextView.getText().toString();
        if (machineName.equals("ONA NX")){
            Intent intent = new Intent(getApplicationContext(), OnaNXActivity.class);
            startActivity(intent);
        }
        else if (machineName.equals("OTHER MACHINES")){
            // TODO: SWITCH CASE QUE LANCE EL RESTO DE ACTIVIDADES
            //       PARA EL RESTO DE MÁQUINAS
        }
    }
};
```

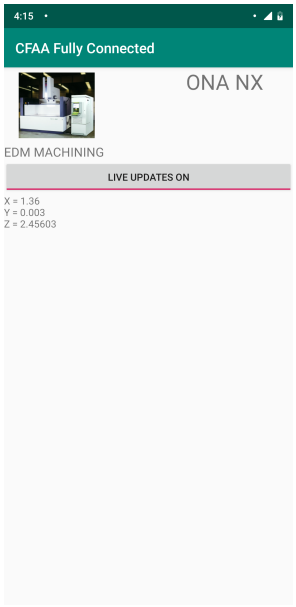
Figura 16: Código de la definición de la clase local OnClickListener.

4.2.5. Clase OnaNXActivity

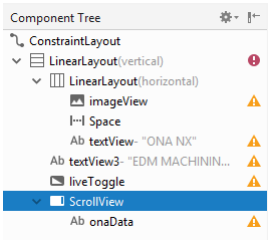
La actividad OnaNXActivity es la que se inicia al seleccionar el texto ONA NX de la actividad de selección de máquina. Presenta información básica de la máquina (tecnología que implementa, nombre y foto) así como un botón con el que activar la muestra de datos. Si se selecciona el botón, en la zona inferior se mostrarán las últimas entradas en la base de datos correspondientes a la máquina ONA NX.

La interfaz consta de un ConstraintLayout en el nivel más alto, que contiene un LinearLayout vertical. Dentro del LinearLayout vertical hay otro horizontal, que contiene, al igual que en la actividad DataActivity, la imagen, el texto y el espaciado correspondiente a la ONA NX. Debajo del LinearLayout horizontal hay otro TextView que muestra la tecnología que implementa la ONA NX, en este caso, el mecanizado por descarga eléctrica (EDM). Debajo del TextView está el botón ToggleButton llamado liveToggle que será el que cambie entre el estado de muestra y no muestra de datos. Debajo, hay un ScrollView que contiene un TextView cuyo texto se actualizará con las últimas lecturas disponibles en la base de datos cuando el ToggleButton esté activado.

Para la implementación de esta clase, se ha supuesto que se toman 3 datos de la ONA NX, la posición X, Y, Z del cabezal. En la práctica se pueden tomar muchas más medidas de forma automática, sin embargo, por simpleza a la hora de desarrollar el código se ha optado por implementar primero los 3 datos mencionados y más adelante, una vez todo el software sea estable y esté probado, implementar el resto.



(a)



(b)

Figura 17: Interfaz y esquema de los componentes en el fichero XML de la actividad OnaNXActivity. (a) Interfaz de la actividad OnaNXActivity. (b) Árbol de componentes de la interfaz de OnaNXActivity.

El funcionamiento de la clase es el siguiente. El método `onCreate()` inicializa la interfaz y asigna al botón ToggleButton un Listener de la clase ToggleLiveUpdatesListener. Esta clase es una clase que se ha desarrollado que implementa las funciones necesarias para actualizar el texto en tiempo real con las medidas obtenidas de la base de datos cuando se activa el botón.

Hay un detalle en la clase OnaNXActivity que es crucial. Tiene un campo estático del tipo Handler[8], que guarda la referencia al hilo en el que se está ejecutando el método `onCreate()` de esta clase. Como el método `onCreate()` es el que inicializa la interfaz, el hilo al que se hace referencia

[8] A efectos prácticos, en este trabajo un Handler es equivalente a un Thread, un objeto que representa un hilo de ejecución.

será el que es encargado de gestionar los cambios en la interfaz.
Este detalle es importante ya que cuando el Listener indique que se han de mostrar los datos por pantalla, la continua actualización de los datos corriendo en el mismo hilo que la interfaz impediría la interacción con la interfaz mientras se espera, por ejemplo, a que la base de datos responda. Como esto es inviable, se ha decidido que la obtención de los datos se haga en un hilo paralelo, de forma que no se interrumpe la capacidad del usuario de interactuar con la aplicación mientras se muestran los datos.
Se ha implementado una función pública, estática y con retorno *void* llamada `setViewText()` que toma como argumentos de entrada un TextView y una String. Este método ejecuta un Runnable en el hilo guardado como campo, de forma que puede hacer cambios en la interfaz. En concreto, el cambio es actualizar el texto del TextView que se le pasa como argumento con el texto contenido en la String.

```java
public class OnaNXActivity extends AppCompatActivity {

    package static Handler UIHandler = new Handler(Looper.getMainLooper());

    public OnaNXActivity(){

    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_ona__nx__activity);
        ToggleButton liveToggle = (ToggleButton) findViewById(R.id.liveToggle);
        TextView liveView = (TextView) findViewById(R.id.onaData);
        try {

            ToggleLiveUpdatesListener listener = new ToggleLiveUpdatesListener(liveView);
            liveToggle.setOnCheckedChangeListener(listener);

        } catch (SQLException e) {
            ExceptionHandler.sqlException(e);
        } catch (ClassNotFoundException e) {
            ExceptionHandler.classNotFoundException(e);
        }

    }

    public static void setViewText(final TextView view, final String text){

        UIHandler.post(() -> { view.setText(text); });

    }


}
```

Figura 18: Implementación de la clase OnaNXActivity.

Así, el propósito de esta clase no es más que inicialice la interfaz, asigne Listeners, y sea capaz de acceder al hilo de la interfaz para actualizar el texto. La interacción con la base de datos y construcción del texto se hace en la clase ToggleLiveUpdatesListener.

4.2.6. Clase ToggleLiveUpdatesListener

Esta clase está específicamente construida para interactuar con el resto de clases de la aplicación y con la base de datos. Por ello, no se enfoca desde el punto de vista de la generalización sino que se trata de implementar de la forma más eficaz posible para el caso práctico concreto aquí presentado.

Con ToggleLiveUpdatesListener se extiende AsyncTask para poder ejecutarse en hilos paralelos y se implementa CompoundButton.OnCheckedChangeListener para que la clase pueda actuar como Listener y sobreescribir los métodos necesarios.

La clase tiene dos campos privados y dinámicos, uno es un objeto SQLConnection y otro un objeto TextView. Como se puede intuir, el objeto SQLConnection será el que se utilice para representar y operar la conexión con la base de datos, y el TextView será el TextView a actualizar con los datos obtenidos. Los métodos implementados son los siguientes:

- `ToggleLiveUpdatesListener()`: Es el constructor de la clase y toma un sólo argumento de entrada, un objeto TextView. Sin embargo, dentro del constructor se hace la asignación de los dos campos anteriormente mencionados. Por un lado se asigna el TextView de entrada al campo TextView, y por otro, se asigna una conexión al campo SQLConnection. Esta última asignación se hace mediante la clase previamente explicada, ConnectionData, haciendo uso del *getter* `ConnectionData.getConnection()`. Así, se utiliza la misma conexión a la base de datos que la utilizada por la actividad LoginActivity para iniciar sesión. Se ha tomado esta estrategia porque los objetos ToggleLiveUpdatesListener tan sólo serán instanciados por las actividades de cada máquina, que a su vez sólo son accesibles si se ha llevado a cabo un inicio de sesión exitoso, lo que implica que ya exista una conexión establecida con la base de datos.
- `onCheckedChange()`: Esta clase ha sido sobreescrita de CompoundButton.OnCheckedChangeListener para que, simplemente, ejecute el método `doInBackground()` cuando el botón esté activado, y no haga nada si se desactiva.
- `doInBackground()`: Este método es el que se ejecuta en un hilo paralelo cuando `onCheckedChange()` lo dicta. Es un ciclo infinito (hasta que el estado del botón cambia, entonces el hilo se destruye y se deja de ejectuar el ciclo) que cada 0.5 segundos manda tres *querys* a la base de datos, cada una pidiendo el último valor conocido de las coordenadas X, Y, Z del cabezal de la máquina. Finalmente, construye una String con los valores obtenidos y llama al método

`OnaNXActivity.setViewText()` para actualizar el texto por pantalla.

Figura 19: Implementación del método `doInBackground()` de la clase ToggleLiveUpdatesListener.

De esta forma, queda completamente desarrollada la aplicación dentro del enfoque y planteamiento aquí presentados.

4.3. Implementación del software para la muestra de datos.

Con la toma de datos configurada, la base de datos en funcionamiento y la aplicación desarrollada e instalada en el teléfono móvil, el desarrollo de la infraestructura para la toma de datos de la ONA NX7 en el CFAA está completada. Debido al carácter académico del trabajo aquí presentado y a la falta de implementación de una capa de seguridad más robusta, el trabajo realizado sirve como base sobre la que construir futuros desarrollos que aporten seguridad y expandan la funcionalidad del sistema de adquisición y presentación de datos.

5. Conclusiones

Con el Trabajo de Fin de Grado realizado, se han puesto en práctica conceptos de hardware, software, redes y comunicaciones para implementar satisfactoriamente una infraestructura que presenta la posibilidad de seguir en tiempo real el estado y las operaciones de la máquina ONA NX7. Además, el desarrollo se ha hecho en todo momento con su expansibilidad en mente, por lo que ampliar la infraestructura no implicaría más que desarrollar los módulos correspondientes a las nuevas máquinas y, basándose en el software ya implementado, desarrollar (trivialmente) la presentación de los datos en la aplicación Android.

La infraestructura se ha implementado en el Centro de Fabricación Avanzada Aeronáutica (CFAA)[1] mediante el dispositivo RELY-CPPS proporcionado por SoC-e, una base de datos MySQL migrable a un servidor alojada en una Raspberry Pi accesible como punto de acceso WiFi y una aplicación Android que permite la presentación de la información de la base de datos al usuario en tiempo real, además de proporcionar una capa de seguridad mediante el uso de credenciales para el acceso a ella.

Con la información acumulada a lo largo del tiempo en la base de datos se puede, además de mostrarla al usuario, alimentar algoritmos de prevención de riesgos y de mantenimiento de máquinas de forma que se puedan evaluar posibles puntos e instantes de fallo en las máquinas antes incluso de que éstos se den, aumentando así la eficiencia y eficacia del centro y la seguridad de sus operarios.

6. Líneas futuras de desarrollo

Evidentemente, la principal línea de desarrollo es implementar una capa de encriptación adicional al sistema, de manera que sin una clave de decriptación, la información de la base de datos sea ilegible en caso de que una tabla sea filtrada. Para ello, una de las formas más seguras es el uso del algoritmo de encriptación y decriptación Advanced Encryption Standard (AES) [26].

Por otro lado, la implementación actual tan sólo permite la lectura de datos de una máquina. Ampliar el número de máquinas es, por lo tanto, imprescindible si se desean digitalizar y automatizar todos los datos obtenidos en el centro.

Finalmente, una de las principales posibilidades que permite la acumulación masiva de datos es el procesado de éstos para labores preventivas y de mantenimiento. La implementación de estos algoritmos en un centro de cálculo

del CFAA y la muestra de los resultados y posibles eventos en tiempo real a los técnicos implicaría un avance considerable en seguridad y eficiencia.

Referencias

[1] L. Sastoque, P. Bedialauneta, S. Sendino, and N. Toledo-Gandarias, "Digitalización al servicio de la fabricación. un nuevo reto para el centro de fabricación avanzada aeronáutica cfaa." *Interempresas*, 2019.

[2] S. K. F. and A. Gill, *Exploring Advanced Manufacturing Technologies*, I. Industrial Press, Ed. Industrial Press, Inc., 2003.

[3] "What is opc?" opcfoundation.org.

[4] R. Coppen, "Mqtt specification," https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html, Feb. 2019.

[5] "Rely-cpps brochure," Especificación de producto. [Online]. Available: https://www.relyum.com/web/wp-content/uploads/2017/07/RELY-CPPS-170530.pdf

[6] A. Lozoya, *Documentación proyecto CFAA [Confidencial]*, SoC-e, Oct. 2018.

[7] *ONA TECHNICAL DOCUMENTATION, Remote Machine Control Protocol for the NX/QX/AX/AF/AV machines.*, ONA, Jun. 2018.

[8] A. Wheen, *Dot-dash to Dot.com: How Modern Telecommunications Evolved from the Telegraph to the Internet.*, Springer, Ed. Springer, 2011.

[9] E. Siever, S. Spainhour, S. Figgins, and J. P. Hekman, *Linux in a Nutshell*, third edition ed. O'Reilly Media, Inc., Aug. 2000, ch. 3, pp. 359–366.

[10] P. S. Foundation, "Telnetlib - telnet client," https://docs.python.org/2/library/telnetlib.html.

[11] T. Halpin and T. Morgan, *Information modelling and Relational Databases*, Elsevier, Ed. Morgan Kauffman, 2008.

[12] E. F. Codd, "A relational model of data for large shared data banks." *Communications of the ACM*, 1970.

[13] S. M. ahaghoghi and H. E. Williams, *Learning MySQL*, A. Oram, Ed. O'Reilly Media, Inc., 2007.

[14] "Mysql server downloads," https://dev.mysql.com/downloads/mysql/.

[15] E. Jacob, J. Astorga, J. J. Unzilla, M. Huarte, D. García, and L. N. L. de Lacalle., "Towards a 5g compliant and flexible connected manufacturing facility," *DYNA Ingeniería e Industria*, 2018.

[16] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob, "Toward an sdn-enabled nfv architecture," *IEEE Communications Magazine*, vol. 53, no. 4, Apr. 2015.

[17] "Raspberry pi documentation: Remote access," https://www.raspberrypi.org/documentation/remote-access/ssh/.

[18] "Official ssh protocol website," https://www.ssh.com/ssh/.

[19] R. Droms, *Dynamic Host Configuration Protocol*, https://tools.ietf.org/pdf/rfc2131.pdf, Network Working Group Std.

[20] "Wifi eduroam ehu," https://www.ehu.eus/es/web/ikt-tic/wifi-eduroam.

[21] *All About Sockets*, The Java Tutorials, https://docs.oracle.com/javase/tutorial/networking/sockets/.

[22] "Raspap webgui," https://github.com/billz/raspap-webgui/blob/master/README.md.

[23] "Android studio," https://developer.android.com/studio.

[24] Z. mednieks, L. Dornin, G. B. Meike, and M. Nakamura, *Programming Android: Java Programming for the New Generation of Mobile Devices*, O'Reilly, Ed. O'Reilly, 2012.

[25] "Jdbc lesson," https://docs.oracle.com/javase/tutorial/jdbc/overview/index.html.

[26] *MySQL Encryption Manual*. [Online]. Available: https://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html