Trabajo Final - Controlador de DVAs

Arquitectura de Software

Matias Ariel Ramirez

4 de diciembre de 2025

Índice

1. Introducción

Horas antes de una clase de búsqueda de personas atrapadas en avalancha, el profesor entierra aparatos denominados DVA (Detector de Víctimas de Avalancha) para simular una persona enterrada que los alumnos deben de buscar y encontrar.
En este punto entra el presente proyecto, “Control de DVAs”. Su función es brindar la capacidad de enterrar los DVA días antes de la clase, apagados, junto a un dispositivo llamado “Controlador de DVA”, que permitirá su encedido remoto via LoRa momentos antes de la clase. Con esto se pretende simular un escenario más parecido a la realidad o al menos más difícil, habiendo eliminado las huellas que quedan en la nieve al enterrar los DVAs.

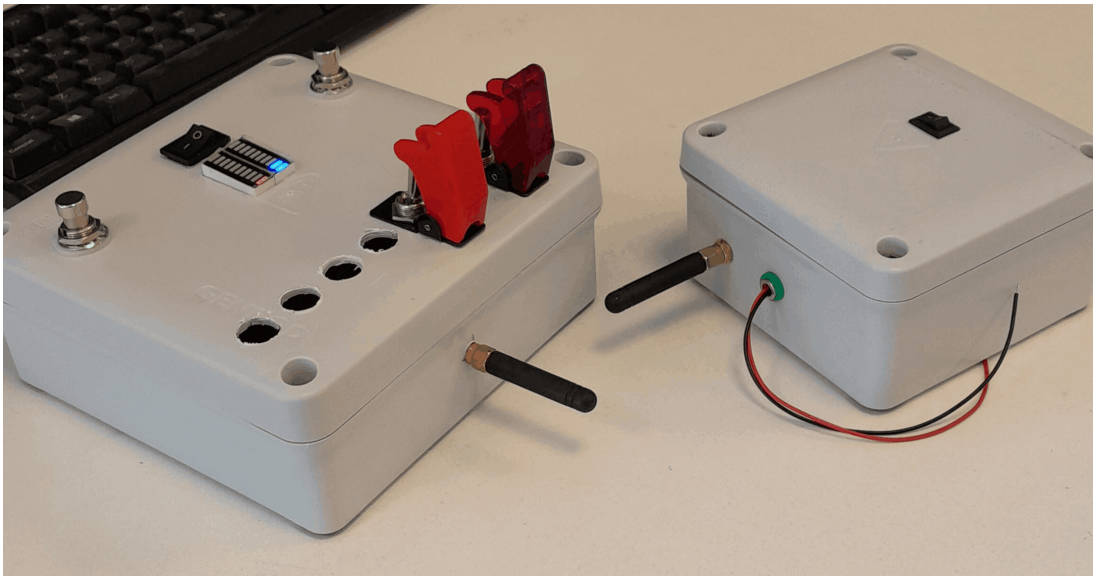
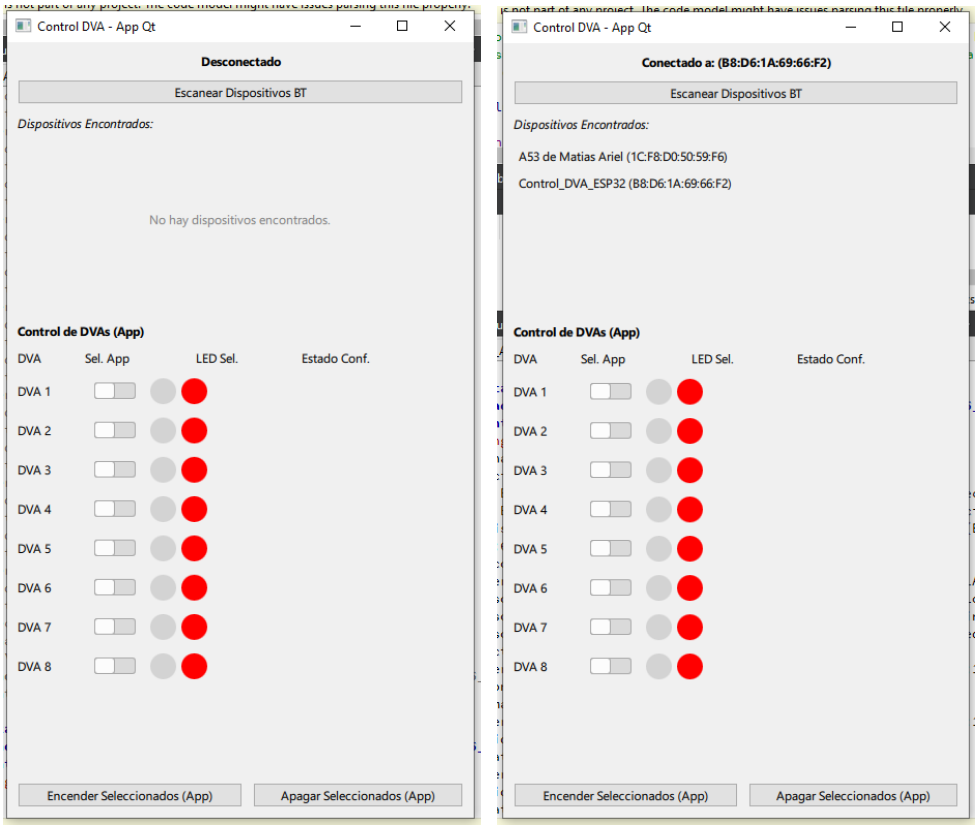


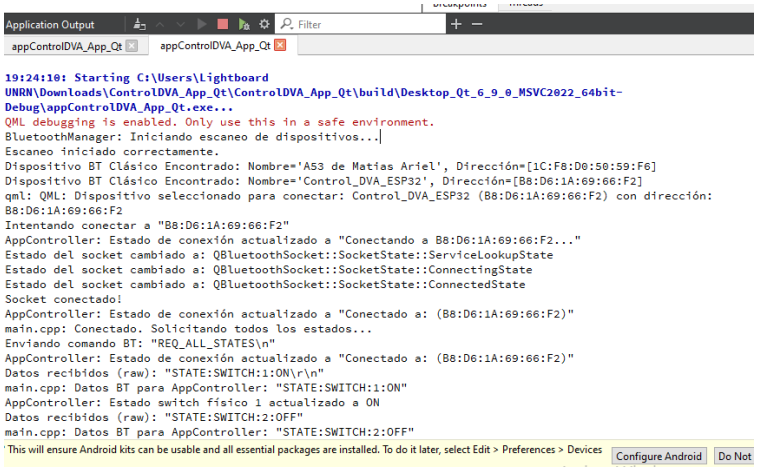
Figura 1: Control DVA y Controlador DVA.

“ControlDVApp” es una aplicación diseñada para controlar el Control de DVAs de forma remota vía Bluetooth, la cual debe de cumplir con los requisitos originales, además de otros que pueden ser de ayuda para realizar pruebas. Estos requisitos los veremos en la sección llamada Requerimientos.

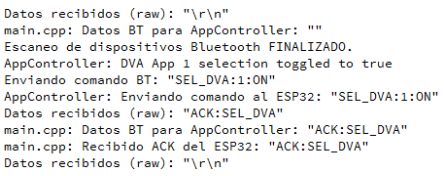


(a) Inicio de la app.

(b) Conección de bluetooth.



(c) Conección de bluetooth en Application Output.



(d) Encender DVA1 Application Output (perdí la imagen de la app).

Figura 2

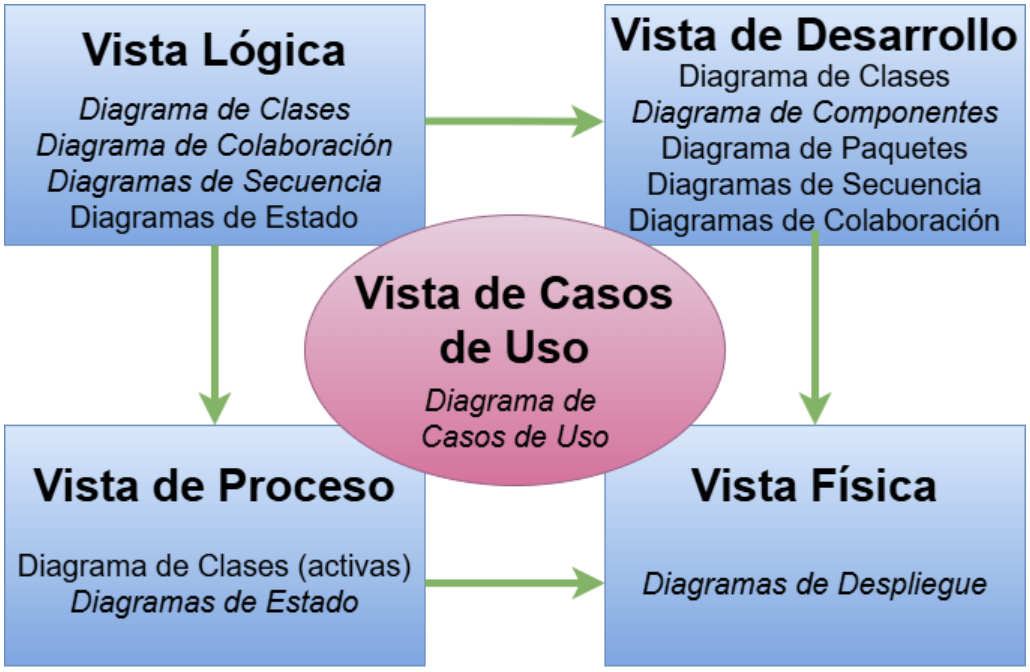
2. Requerimientos

De cumplir con:

- Seleccionar 1 o varios DVAs a la vez para realizar la siguiente acción.
- Encender o apagar DVAs a distancia.
- Visualizar cual o cuales DVAs están encendidos.
- Comunicarse con los Controladores de DVAs a una distancia máxima de 70 metros y con medio metro enterrados.

3. Resolución

Para la resolución arquitectónica del sistema, se ha optado por utilizar el modelo de “4+1 Vistas” de Philippe Kruchten. Este enfoque permite documentar las decisiones de diseño desde diferentes perspectivas interconectadas, asegurando que se cubran tanto los requerimientos funcionales como los no funcionales y facilitando la comunicación con los distintos interesados.



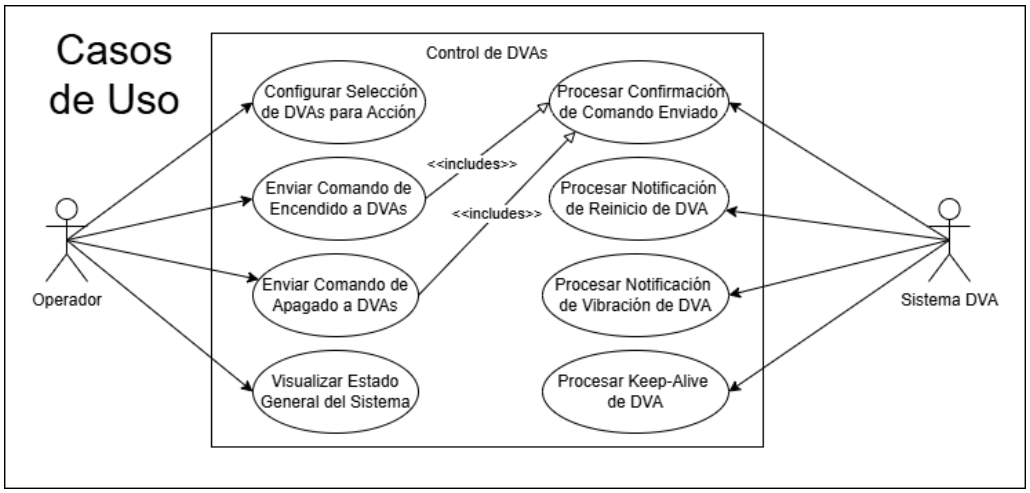
Si bien hay diagramas que comparten más de una vista, solo se describirán una sola vez, visto desde un enfoque en especifico. Dicho enfoque, en la figura, se resalta en itálica.
Estas vistas y sus respectivos Diagramas se detallaran en sus respectivos apartados.

3.1. Vista de Casos de Uso

La mejor forma de comenzar una arquitectura es saber quien interactúa y que funciones necesita. Esta es la vista central que describe la funcionalidad del sistema desde la perspectiva del usuario final, capturando los requisitos funcionales.

3.1.1. Diagramas de Casos de Uso

Para describir cómo interactúan los actores con el sistema y su funcionalidad, se utiliza el Diagrama de Casos de Uso.



Actores

Los actores que interactúan con el Sistema de Control DVA LoRa son:

- **Operador del Control (Primario):**
  - *Descripción:* Es el usuario humano que interactúa directamente con el dispositivo de Control para gestionar los DVAs remotos.
  - *Objetivos:* Encender/apagar DVAs, conocer el estado de selección y el estado confirmado de los DVAs, ser alertado sobre vibraciones.
- **Sistema DVA Remoto (Secundario):**
  - *Descripción:* Es la unidad DVA individual que se encuentra en la nieve. Se comunica con el Sistema de Control vía LoRa.
  - *Objetivos:* Responder a comandos, enviar notificaciones de estado (reinicio, vibración).

Casos de Uso del Sistema

El Sistema de Control DVA LoRa es el sistema bajo diseño.

Caso de Uso 1: Configurar Selección de DVAs para Acción

- **Actor Principal:** Operador del Control
- **Descripción Breve:** El operador utiliza las 8 palancas (switches) para seleccionar o deseleccionar los DVAs que serán afectados por comandos. El Sistema de Control refleja esta selección en la Barra 2 física (derecha).
- **Precondiciones:** El Sistema de Control está encendido.
- **Flujo Principal:**
  1. El Operador del Control ajusta la posición de una o más palancas.
  2. El Sistema de Control detecta el cambio en el estado de las palancas.
  3. El Sistema de Control actualiza el LED correspondiente en la Barra 2 física para cada palanca (LED encendido si palanca ON, LED apagado si palanca OFF).
- **Postcondiciones:** La selección de DVAs para la próxima acción está actualizada y es visible para el operador.

Caso de Uso 2: Enviar Comando de Encendido a DVAs

- **Actor Principal:** Operador del Control
- **Descripción Breve:** El operador presiona el botón “Encender”. El Sistema de Control envía un comando LoRa “Encender DVA X” a cada Sistema DVA Remoto seleccionado y procesa la confirmación.
- **Precondiciones:**
  - El Sistema de Control está encendido.
  - Al menos un DVA está seleccionado mediante las palancas (resultado de Caso de Uso 1).
- **Flujo Principal:**
  1. El Operador del Control presiona el botón “Encender”.
  2. El Sistema de Control detecta la pulsación.
  3. Para cada DVA *X* seleccionado por las palancas:
     a) El Sistema de Control construye el mensaje “Encender DVA X”.
     b) El Sistema de Control envía el mensaje LoRa al Sistema DVA Remoto *X*.
     c) El Sistema de Control espera la confirmación (invoca Caso de Uso 5).
- **Postcondiciones:**
  - Comandos de encendido enviados a los DVAs seleccionados.
  - LEDs de confirmación (Barra 1 física - izquierda) actualizados según respuesta.
- **Relaciones:** «incluye» Caso de Uso 5.

Caso de Uso 3: Enviar Comando de Apagado a DVAs

- **Actor Principal:** Operador del Control
- **Descripción Breve:** El operador presiona el botón “Apagar”. El Sistema de Control envía un comando LoRa “Apagar DVA X” a cada Sistema DVA Remoto seleccionado y procesa la confirmación.
- **Precondiciones:**
  - El Sistema de Control está encendido.
  - Al menos un DVA está seleccionado mediante las palancas (resultado de Caso de Uso 1).
- **Flujo Principal:**
  1. El Operador del Control presiona el botón “Apagar”.
  2. El Sistema de Control detecta la pulsación.
  3. Para cada DVA *X* seleccionado por las palancas:
     a) El Sistema de Control construye el mensaje “Apagar DVA X”.
     b) El Sistema de Control envía el mensaje LoRa al Sistema DVA Remoto *X*.
     c) El Sistema de Control espera la confirmación (invoca Caso de Uso 5).

- **Postcondiciones:**
  - Comandos de apagado enviados a los DVAs seleccionados.
  - LEDs de confirmación (Barra 1 física - izquierda) actualizados según respuesta.
- **Relaciones:** «incluye» Caso de Uso 5.

Caso de Uso 4: Visualizar Estado General del Sistema

- **Actor Principal:** Operador del Control
- **Descripción Breve:** El operador observa las barras de LEDs para conocer el estado de selección de las palancas (Barra 2 física) y el último estado confirmado/reportado de cada DVA (Barra 1 física).
- **Precondiciones:** El Sistema de Control está encendido.
- **Flujo Principal:**
  1. El Operador del Control observa la Barra 2 física para ver los DVAs seleccionados por las palancas.
  2. El Operador del Control observa la Barra 1 física para ver:
     - LEDs encendidos: DVA confirmado como ENCENDIDO.
     - LEDs apagados: DVA confirmado como APAGADO / fallo de comando / reinicio.
     - LEDs parpadeando: DVA reportando vibración (Caso de Uso 7).
- **Postcondiciones:** El operador está informado visualmente del estado del sistema.

Caso de Uso 5: Procesar Confirmación de Comando Enviado

- **Actor que interactúa (inicia respuesta):** Sistema DVA Remoto
- **Descripción Breve:** El Sistema de Control recibe un mensaje de confirmación (”Encender DVA X OK” o “Apagar DVA X OK”) de un Sistema DVA Remoto como respuesta a un comando enviado (desde Caso de Uso 2 o Caso de Uso 3).
- **Precondiciones:** El Sistema de Control ha enviado un comando y está en espera activa de una confirmación.
- **Flujo Principal:**
  1. El Sistema DVA Remoto procesa el comando y envía el mensaje de confirmación.
  2. El Sistema de Control recibe el mensaje LoRa.
  3. El Sistema de Control verifica que la confirmación es la esperada para el comando y DVA X.
  4. Si la confirmación es para “Encender DVA X OK”: El Sistema de Control enciende el LED de confirmación $X + 7$ (Barra 1 física) y detiene el parpadeo si estaba activo.
  5. Si la confirmación es para “Apagar DVA X OK”: El Sistema de Control apaga el LED de confirmación $X + 7$ (Barra 1 física) y detiene el parpadeo si estaba activo.
- **Postcondiciones:** El LED de confirmación del DVA X se actualiza. Se cancela la espera de confirmación para ese comando.

Caso de Uso 6: Procesar Notificación de Reinicio de DVA

- **Actor que interactúa (inicia notificación):** Sistema DVA Remoto
- **Descripción Breve:** El Sistema de Control recibe un mensaje espontáneo “Apagar DVA X OK” de un Sistema DVA Remoto, indicando que dicho DVA se ha reiniciado y está apagado.
- **Precondiciones:** El Sistema de Control está en modo de escucha LoRa.
- **Flujo Principal:**
  1. El Sistema DVA Remoto se reinicia y envía el mensaje “Apagar DVA X OK”.
  2. El Sistema de Control recibe el mensaje LoRa (fuera de una espera activa de confirmación).
  3. El Sistema de Control identifica el DVA X.
  4. El Sistema de Control apaga el LED de confirmación $X + 7$ (Barra 1 física).
  5. El Sistema de Control detiene el parpadeo del LED si estaba activo.
- **Postcondiciones:** El estado del DVA X se actualiza a APAGADO en el Sistema de Control.

Caso de Uso 7: Procesar Notificación de Vibración de DVA

- **Actor que interactúa (inicia notificación):** Sistema DVA Remoto
- **Descripción Breve:** El Sistema de Control recibe un mensaje espontáneo “Vibrando DVA X”. Hace parpadear el LED de confirmación correspondiente por un tiempo y luego lo deja encendido.
- **Precondiciones:** El Sistema de Control está en modo de escucha LoRa.
- **Flujo Principal:**
  1. El Sistema DVA Remoto detecta vibración y envía “Vibrando DVA X”.
  2. El Sistema de Control recibe el mensaje LoRa.
  3. El Sistema de Control identifica el DVA X.
  4. El Sistema de Control inicia el parpadeo del LED de confirmación $X + 7$ (Barra 1 física).
  5. Tras un periodo definido (ej. 2 segundos), el Sistema de Control detiene el parpadeo y deja el LED $X + 7$ encendido fijo.
- **Postcondiciones:** El Operador del Control es alertado de la vibración. El LED del DVA X queda encendido.

Relaciones entre Casos de Uso

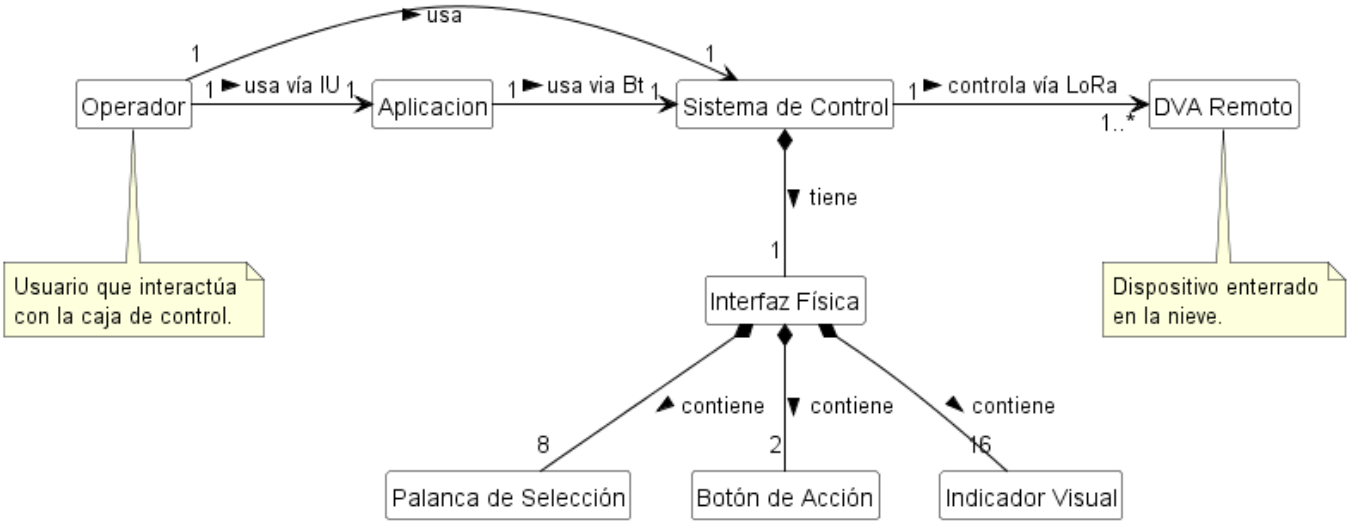
- **Caso de Uso 2 (Enviar Comando de Encendido a DVAs)** *«incluye»* **Caso de Uso 5 (Procesar Confirmación de Comando Enviado)**
- **Caso de Uso 3 (Enviar Comando de Apagado a DVAs)** *«incluye»* **Caso de Uso 5 (Procesar Confirmación de Comando Enviado)**

3.2. Vista Lógica

Describe la estructura y funcionalidad del sistema en términos de clases, interfaces y sus relaciones. Responde a la pregunta: ¿Qué debe hacer el sistema funcionalmente?
Se centra en los conceptos funcionales del dominio del problema, identificando las abstracciones clave con las que interactúa el usuario, independientemente de su implementación en software.
Esta vista se divide en los Diagramas y Patrones de Diseño. Para mantener una estructura de mostrar todos los diagramas primero, en esta subsección se mostrara eso, los Patrones de Diseño están descritos en la sección 4.

3.2.1. Diagramas de Clases

El Diagrama de Clases conceptual presentado a continuación ilustra las abstracciones clave del dominio del problema, desvinculadas de los detalles de implementación de software. Su objetivo es establecer un vocabulario común y definir las relaciones estructurales entre las entidades físicas y lógicas del sistema.



Los elementos principales identificados son:

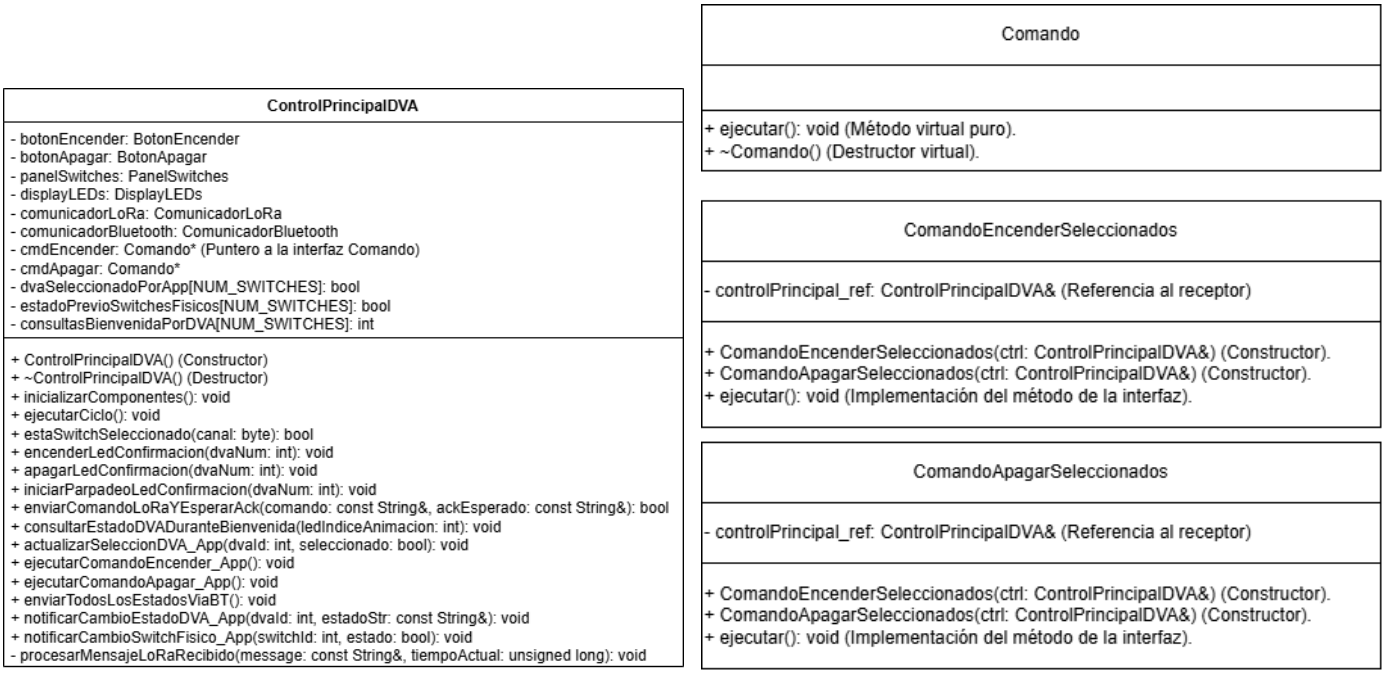
- **Operador**: Representa al usuario encargado de la gestión de los dispositivos. Es el actor principal que inicia las interacciones con el sistema.
- **Aplicación**: Puede funcionar como intermediario entre el Operador y el Sistema de Control en caso de no querer controlar el Sistema de Control mediante su Interfaz Física.
- **Sistema de Control**: Actúa como la entidad central coordinadora. Representa la "caja" física del controlador que orquesta la lógica de negocio y mantiene el estado general de la operación.
- **DVA Remoto**: Modela las unidades físicas desplegadas en el terreno (los dispositivos enterrados). El sistema mantiene una relación de cardinalidad uno a muchos con estos dispositivos, gestionándolos a través de comunicación inalámbrica (LoRa).
- **Interfaz Física**: Agrupa los elementos tangibles de interacción (entradas y salidas). Se ha modelado mediante una relación de composición fuerte, ya que el sistema de control contiene físicamente a las palancas de selección (para elegir los DVAs activos), los botones de acción (para ejecutar comandos) y los indicadores visuales (LEDs) que proporcionan retroalimentación inmediata al operador.

3.3. Vista de Desarrollo

Esta vista se enfoca en la organización del sistema desde la perspectiva de un programador. Muestra cómo el software está dividido y organizado en módulos y componentes en el entorno de desarrollo. Responde a la pregunta: ¿Cómo se organiza el código fuente del sistema?. Para modelar esta vista, UML utiliza principalmente el Diagrama de Componentes y el Diagrama de Paquetes.

3.3.1. Diagramas de Clases

Mediante este diagrama podemos ver como se relacionan las clases, sus métodos y atributos. A continuación se mostraran los atributos y operaciones de cada una de las clases y luego las relaciones entre ellas.



(a) Clase ControlPrincipalDVA.

(b) Clase Comando, ComandoEncenderSeleccionados y ComandoApagarSeleccioandos

Figura 3: Atributos y operaciones de los Diagramas de Clases

`ControlPrincipalDVA`

- **Responsabilidad Principal:** Orquesta todas las operaciones del sistema. Actúa como clase central que gestiona las entradas del usuario (botones físicos, switches, comandos Bluetooth), interactúa con los periféricos (LEDs, LoRa) y mantiene el estado general.
- **Relaciones con otras clases:**
  - **Agregación fuerte (Composición) con:** `PanelSwitches` (1), `DisplayLEDs` (1), `ComunicadorLoRa` (1), `ComunicadorBluetooth` (1), `BotonEncender` (1), `BotonApagar` (1). Estas partes son esenciales para el funcionamiento de `ControlPrincipalDVA` y su ciclo de vida está ligado al de `ControlPrincipalDVA`. Multiplicidad: 1 a 1 en cada caso.
  - **Asociación con:** `Comando` (específicamente instancias de `ComandoEncenderSeleccionados` y `ComandoApagarSeleccionados`, referenciadas como `cmdEncender` y `cmdApagar`). `ControlPrincipalDVA` “tiene un” o usa estos comandos. Multiplicidad: 1 a 1 para cada comando.

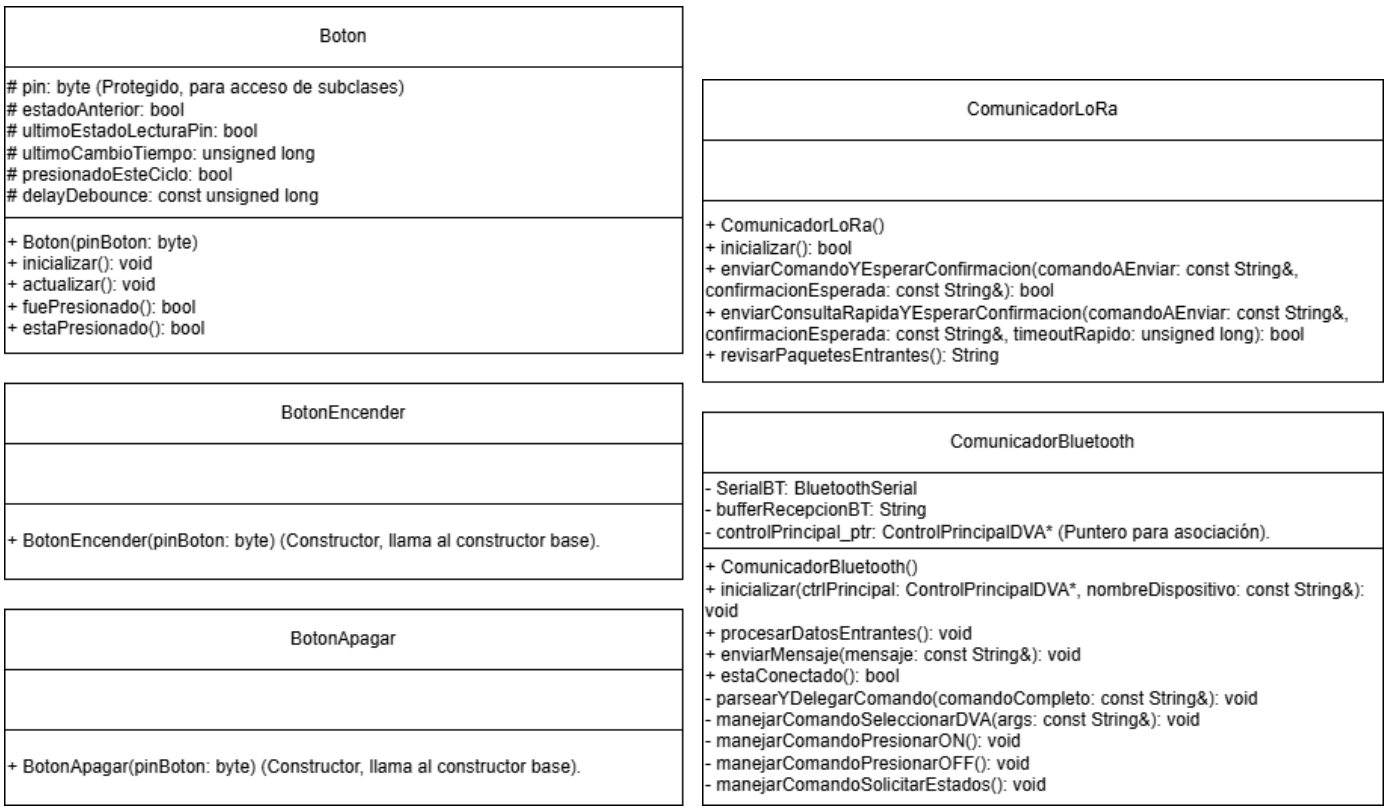
- **Rol del Receptor:** Actúa como el *receptor* para los objetos `Comando`.

`Comando` (Interfaz)

- **Responsabilidad Principal:** Define la interfaz común para todas las acciones concretas (comandos), típicamente con un método `ejecutar()`.
- **Relaciones con otras clases:**
  - **Generalización por:** `ComandoEncenderSeleccionados` y `ComandoApagarSeleccionados` "son un tipo de" `Comando`.
  - En UML, se representaría con el estereotipo «`interface`» o como una clase abstracta con métodos abstractos (itálica).

`ComandoEncenderSeleccionados` / `ComandoApagarSeleccionados`

- **Responsabilidad Principal:** Encapsulan la acción específica de encender o apagar los DVAs seleccionados. Conocen al receptor (`ControlPrincipalDVA`) de la acción.
- **Relaciones con otras clases:**
  - **Generalización a:** Implementan la interfaz `Comando`.
  - **Asociación (o Dependencia) con:** `ControlPrincipalDVA`. Tienen una referencia (`controlPrincipal_ref`) al `ControlPrincipalDVA`, que actúa como el receptor de la acción. Multiplicidad: Cada comando concreto tiene una referencia a un `ControlPrincipalDVA`.



(a) Clase Boton, BotonEncender y BotonApagar. (b) Clase ComunicadorLoRa y ComunicadorBluetooth

Figura 4: Atributos y operaciones de los Diagramas de Clases

`Boton` / `BotonEncender` / `BotonApagar`

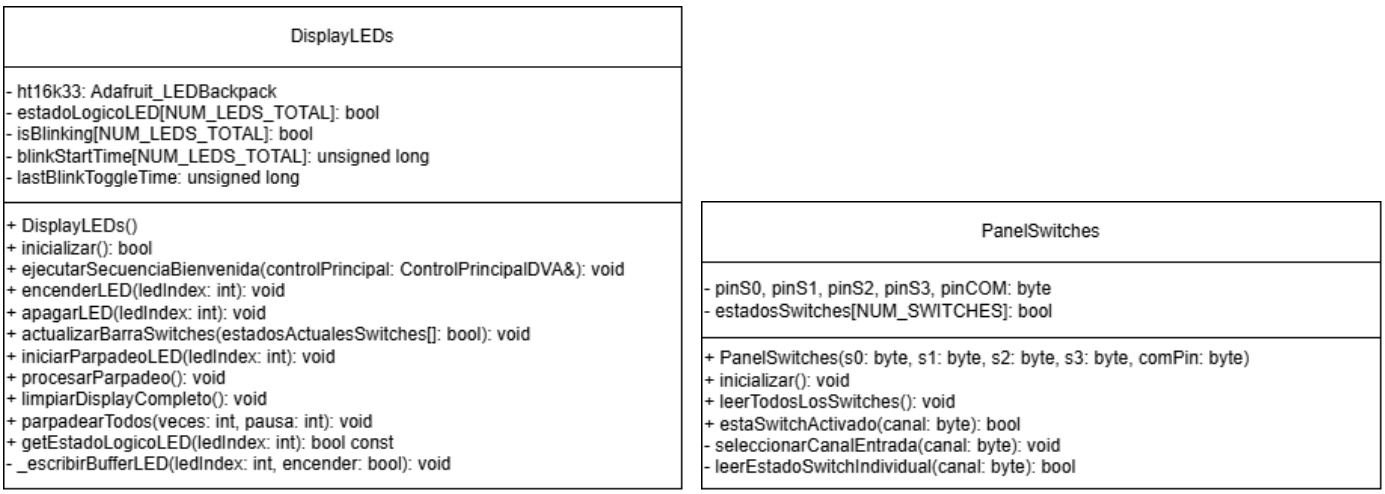
- **Responsabilidad Principal:** `Boton` maneja la lógica de detección de pulsaciones de un botón físico, incluyendo anti-rebote. `BotonEncender` y `BotonApagar` son especializaciones semánticas.
- **Relaciones con otras clases:**
  - **Generalización:** `BotonEncender` y `BotonApagar` heredan de `Boton` ("es un tipo de" Boton).
  - **Agregación (Composición) por:** `ControlPrincipalDVA` "tiene un" `BotonEncender` y un `BotonApagar`.

`ComunicadorLoRa`

- **Responsabilidad Principal:** Encapsula la lógica para enviar comandos y recibir respuestas a través del módulo LoRa.
- **Relaciones con otras clases:**
  - **Agregación (Composición) por:** `ControlPrincipalDVA` "contiene un" `ComunicadorLoRa`.

`ComunicadorBluetooth`

- **Responsabilidad Principal:** Maneja la comunicación bidireccional (comandos y estados) con la aplicación Qt a través de Bluetooth Serial (SPP).
- **Relaciones con otras clases:**
  - **Agregación (Composición) por:** `ControlPrincipalDVA` "contiene un" `ComunicadorBluetoo`
  - **Asociación con:** `ControlPrincipalDVA`. Tiene un puntero (`controlPrincipal_ptr`) a `ControlPrincipalDVA` para poder invocar sus métodos (ej. `actualizarSeleccionDVA_Ap` `ejecutarComandoEncender_App`) cuando recibe comandos de la app.



(a) Clase DisplayLEDs. (b) Clase PanelSwitches.

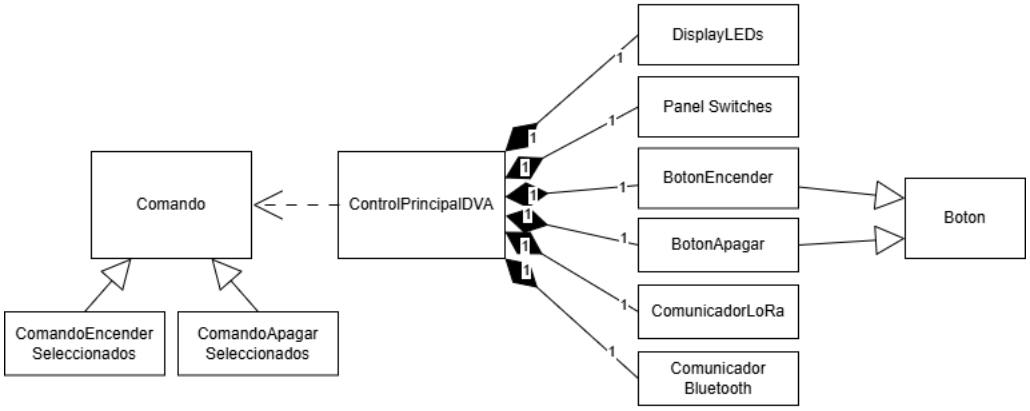
Figura 5: Atributos y operaciones de los Diagramas de Clases

`DisplayLEDs`

- **Responsabilidad Principal:** Controla las dos barras de LEDs (selección y confirmación) a través del controlador HT16K33. Maneja el encendido, apagado y parpadeo de LEDs individuales y secuencias.
- **Relaciones con otras clases:**
  - **Agregación (Composición) por:** `ControlPrincipalDVA` "contiene un" `DisplayLEDs`.
  - **Asociación (o Dependencia) con:** `ControlPrincipalDVA` en la función `ejecutarSecuencia` para llamar a `consultarEstadoDVADuranteBienvenida`. Es una dependencia porque `DisplayLEDs` "usa un" `ControlPrincipalDVA` en ese contexto.

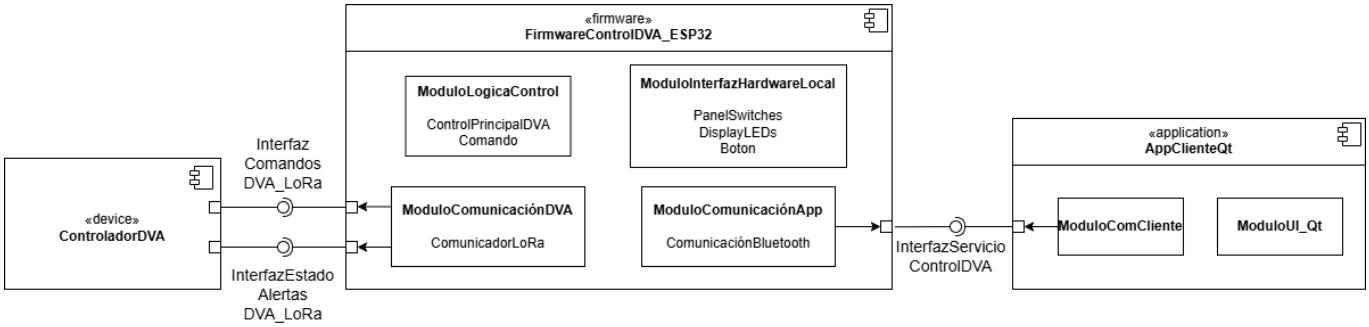
`PanelSwitches`

- **Responsabilidad Principal:** Abstrae el manejo del multiplexor para leer el estado de los 8 switches (palancas) de selección de DVA.
- **Relaciones con otras clases:**
  - **Agregación (Composición) por:** `ControlPrincipalDVA` "contiene un" `PanelSwitches`.



3.3.2. Diagrama de Componentes

El diagrama de componentes ilustra la arquitectura modular del sistema de Control DVA, mostrando los principales componentes de software y hardware, sus interfaces y las dependencias entre ellos. Presenta una vista arquitectónica de alto nivel del sistema de Control DVA, mostrando sus principales bloques de construcción modulares y las relaciones entre ellos. Este tipo de diagrama ayuda a entender la organización estructural del sistema y cómo las diferentes partes del software y hardware colaboran.



Los principales componentes identificados son:

- **`AppClienteQt:`** Es el componente de software que se ejecuta en el dispositivo del usuario (móvil o PC). Su responsabilidad es proveer la interfaz gráfica para el control y monitoreo remoto de los DVAs. Internamente, se subdivide en un módulo de interfaz de usuario (UI) y un módulo de comunicación cliente que maneja la conexión con el ESP32.
- **`FirmwareControlDVA_ESP32:`** Representa el software embebido que se ejecuta en la placa ESP32. Es el corazón del sistema, responsable de la lógica de control de los DVAs, la gestión de la interfaz física local (botones, LEDs, switches), y la mediación de la comunicación tanto con los DVAs remotos (vía LoRa) como con la `AppClienteQt` (vía Bluetooth o WiFi). Se puede conceptualizar con subcomponentes internos para la lógica de control, la comunicación con la app, la comunicación con los DVAs, y la interfaz con el hardware local.
- **`DVAFisico:`** Es un componente externo que representa cada uno de los dispositivos de víctima de avalancha desplegados en campo. El firmware del ESP32 interactúa con múltiples instancias de este tipo de componente.

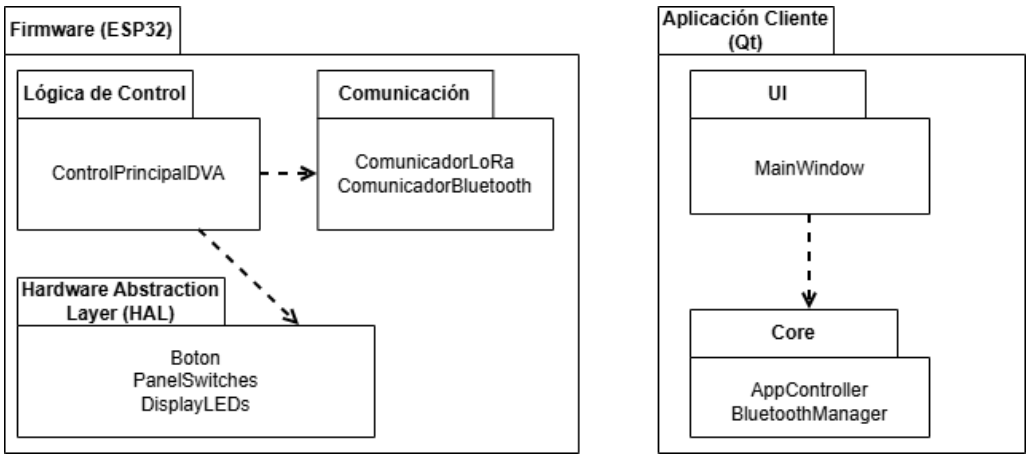
Las interacciones entre estos componentes se definen a través de interfaces:

- La `AppClienteQt` requiere la **`InterfazServicioControlDVA`**, la cual es proporcionada por el `FirmwareControlDVA_ESP32`. Esta interfaz define el contrato de comunicación (el protocolo de mensajes) que permite a la app enviar comandos (ej. seleccionar DVA, encender, apagar) y recibir actualizaciones de estado.
- El `FirmwareControlDVA_ESP32` interactúa con los `DVAFisico` a través de dos interfaces principales sobre LoRa: requiere una **`InterfazEstadoAlertasDVA_LoRa`** (para recibir datos del DVA) y utiliza (o el DVA requiere) una **`InterfazComandosDVA_LoRa`** (para enviar comandos al DVA).

3.3.3. Diagrama de Paquetes

Complementando el diagrama de componentes, se presenta un diagrama de clases organizado por paquetes para ilustrar la estructura modular del código fuente. Este diagrama refleja cómo las clases definidas en la vista lógica se agrupan en unidades de desarrollo coherentes (namespaces o carpetas), facilitando la gestión de dependencias y la compilación separada.

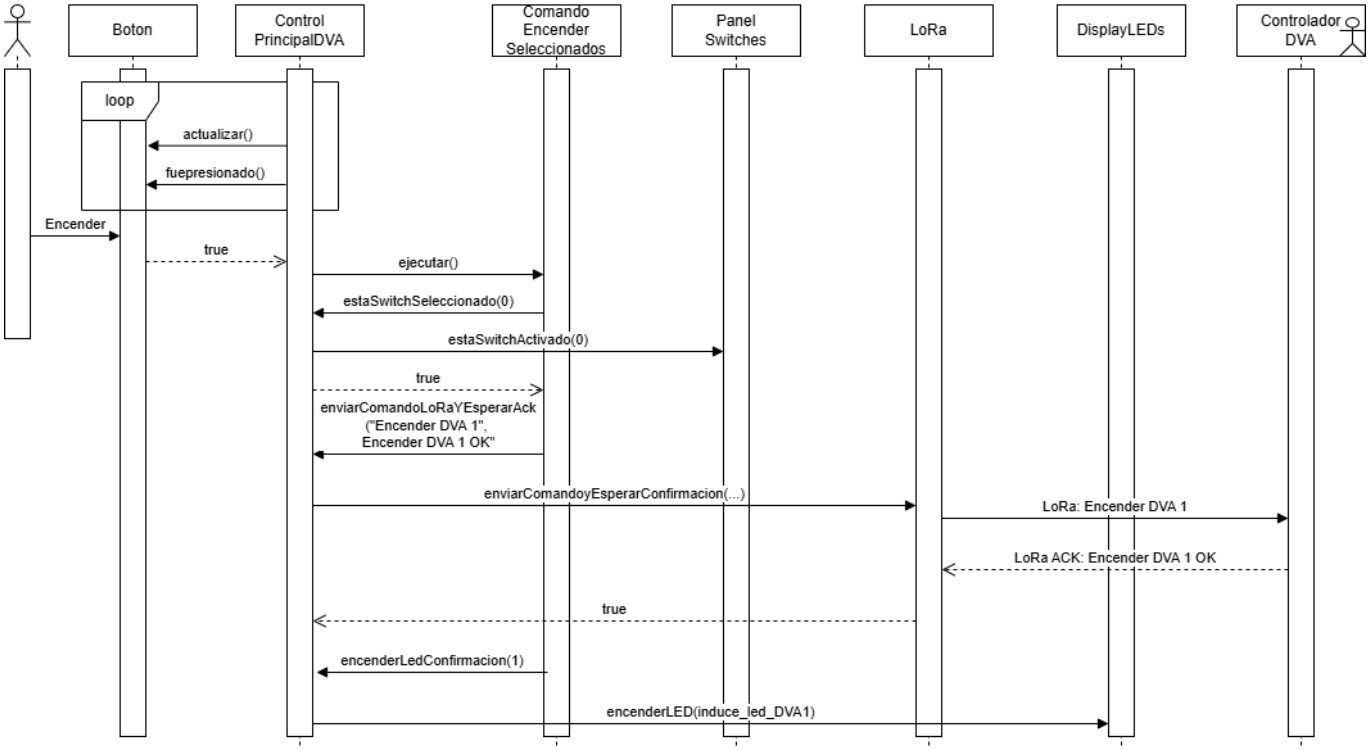
Se observa una clara separación entre las clases que manejan la abstracción de hardware (paquete HAL), las que gestionan la lógica de control (paquete Lógica) y los módulos de comunicación, respetando el principio de separación de responsabilidades también a nivel de estructura de archivos.



- **Firmware (ESP32)**: El paquete *Lógica de Control* necesita leer los botones, encender LEDs y comunicarse por LoRa y Bluetooth.
- **Aplicación (Qt)**: El paquete *UI* necesita llamar a las funciones del controlador.

3.3.4. Diagrama de Secuencia

El diagrama de secuencia muestra el flujo de colaboración entre objetos en el orden temporal. El siguiente diagrama muestra la secuencia de interacciones cuando un usuario presiona el botón físico “Encender” en el dispositivo ESP32 para activar los DVAs seleccionados mediante las palancas físicas.



La secuencia se inicia con una acción externa del **Usuario** al presionar el objeto `BotonEncender`. Dentro del bucle principal de ejecución del `ControlPrincipalDVA` (representado conceptualmente por la auto-llamada a `ejecutarCiclo()`), se detecta esta pulsión mediante llamadas a los métodos `actualizar()` y `fuePresionado()` del objeto `BotonEncender`.

Una vez confirmada la pulsación, `ControlPrincipalDVA` invoca el método `ejecutar()` de su objeto `ComandoEncenderSeleccionados`. Este último es una instancia del patrón Command, responsable de encapsular y orquestar la acción de encendido.

El objeto `ComandoEncenderSeleccionados` primero determina cuáles DVAs están seleccionados. Para ello, invoca el método `estaSwitchSeleccionado(indice)` en su referencia al `ControlPrincipalDVA`. Este, a su vez, consulta al objeto `PanelSwitches` mediante su método `estaSwitchActivado(indice)` para obtener el estado de la palanca física correspondiente.

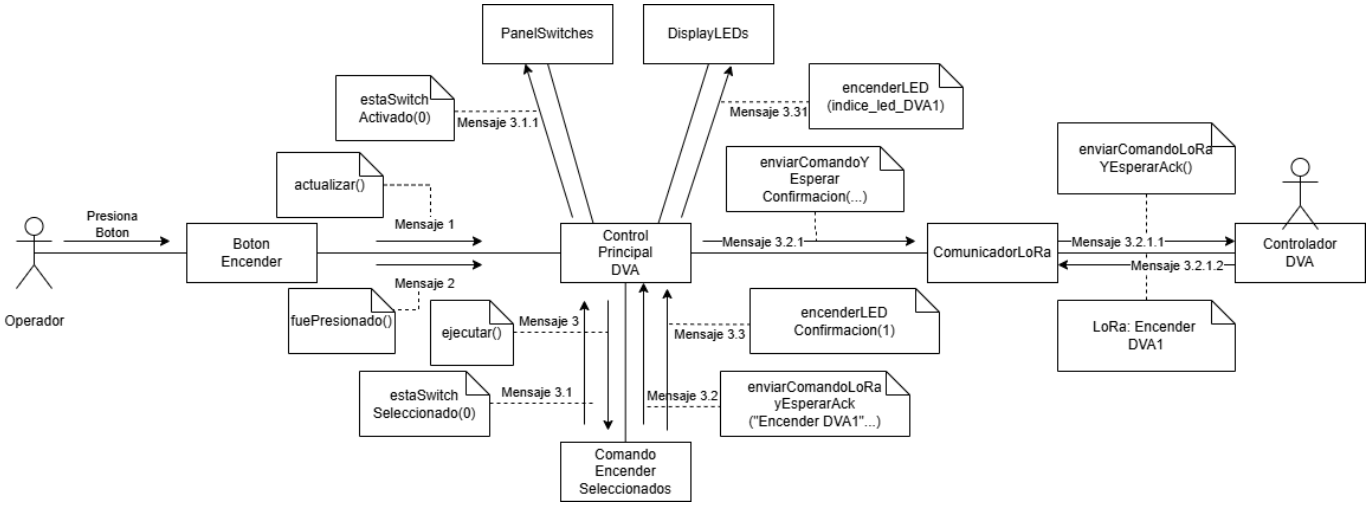
Por cada DVA que se encuentre seleccionado (en el diagrama se ejemplifica con el DVA 1), el `ComandoEncenderSeleccionados` instruye al `ControlPrincipalDVA` (actuando como *Receptor* del comando) para que envíe el comando de encendido vía LoRa. Esto se realiza a través del método `enviarComandoLoRaYEsperarAck(...)` del `ControlPrincipalDVA`, el cual delega la comunicación real al objeto `ComunicadorLoRa`.

`ComunicadorLoRa` transmite el mensaje “Encender DVA 1” al `DVAFisico` remoto y gestiona la recepción de la confirmación (ACK) por parte de este. Suponiendo una transmisión y confirmación exitosas, `ComunicadorLoRa` retorna un indicador de éxito al `ControlPrincipalDVA`.

Finalmente, el `ComandoEncenderSeleccionados`, nuevamente a través de su referencia al `ControlPrincipalDVA`, invoca `encenderLedConfirmacion(1)`. El `ControlPrincipalDVA` entonces instruye al objeto `DisplayLEDs` para que encienda el LED de confirmación correspondiente, proporcionando así retroalimentación visual al usuario de que el DVA 1 ha sido encendido (o al menos, se ha recibido la confirmación de ello).

3.3.5. Diagrama de Colaboración

El Diagrama de Colaboración se utiliza para modelar cómo los objetos colaboran entre sí para realizar una tarea o caso de uso específico. A diferencia del Diagrama de Secuencia, que trata sobre el orden temporal de los mensajes, el Diagrama de Colaboración se centra en la organización estructural de los objetos que interactúan y los mensajes que se intercambian a lo largo de los enlaces (relaciones) que los conectan.
A continuación se mostrara un diagrama de colaboración para el caso especifico de encender el DVA 1 en caso exitoso.

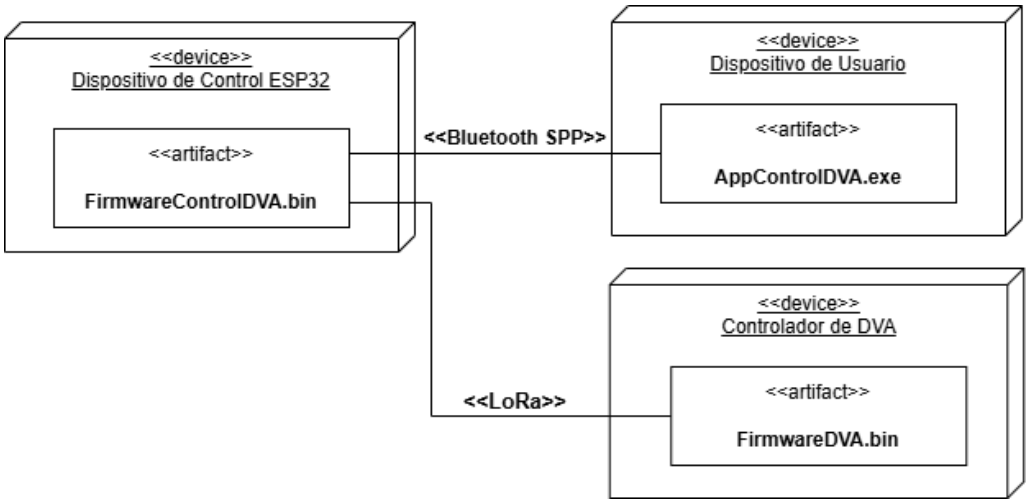


3.4. Vista Física

Esta vista se ocupa de la topología del sistema y de cómo los componentes de software se mapean en la infraestructura de hardware sobre la que se ejecutan. Responde a la pregunta: ¿Dónde se despliega y ejecuta el software, y cómo se comunican las diferentes piezas de hardware?.
Para modelar esta perspectiva, UML utiliza principalmente el Diagrama de Despliegue.

3.4.1. Diagrama de Despliegue

El diagrama de despliegue ofrece una perspectiva clara de la arquitectura física del sistema, detallando dónde reside cada pieza de software y cómo se comunican las diferentes partes del hardware para lograr la funcionalidad completa del sistema de control DVA.



Se identifican tres tipos principales de nodos en el sistema:

1. **`Dispositivo de Control ESP32:`** Es el nodo central del sistema, implementado sobre una placa ESP32-WROOM-32. Alberga el artefacto de software principal, `FirmwareControlDVA.bin`, que contiene toda la lógica embebida para la gestión de los DVAs, la interfaz de usuario local (botones, LEDs, switches) y la comunicación con otros nodos.
2. **`Dispositivo del Usuario (Celular/PC):`** Este nodo representa el dispositivo (smartphone o computadora) utilizado por el operador para interactuar remotamente con el sistema. Ejecuta el artefacto `AppControlDVA.exe` (o su equivalente según la plataforma), que es la aplicación cliente desarrollada en Qt.
3. **`DVA Remoto:`** Representa cada una de las unidades de DVA físicas desplegadas en el terreno. Cada DVA remoto ejecuta su propio `FirmwareDVA.bin`, encargado de su operación autónoma y comunicación.

Las interacciones entre estos nodos se realizan a través de dos caminos de comunicación principales:

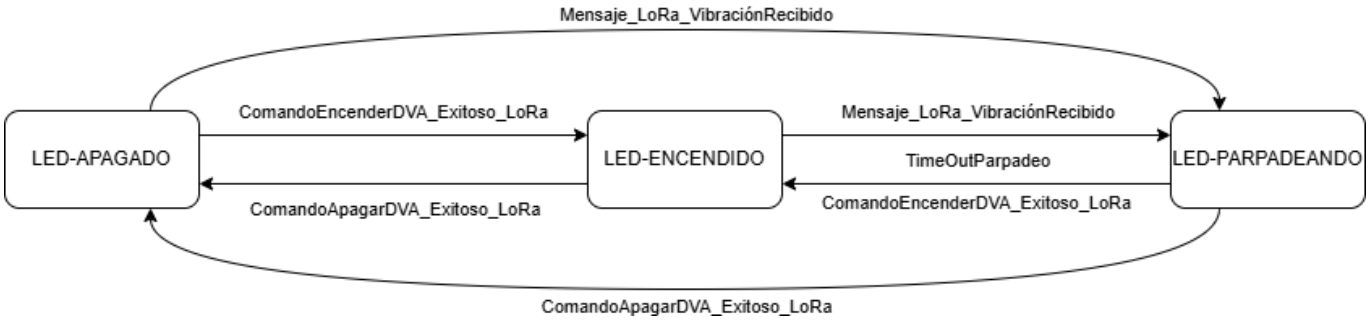
- Una conexión **«`Bluetooth SPP`»** se establece entre el `Dispositivo de Control ESP32` y el `Dispositivo del Usuario`. A través de este canal, la aplicación Qt envía comandos de control y recibe actualizaciones de estado y alertas.
- Una comunicación **«`LoRa`»** se utiliza entre el `Dispositivo de Control ESP32` y cada `DVA Remoto`. Este enlace permite al ESP32 enviar comandos de encendido/apagado a los DVAs y recibir de estos confirmaciones y alertas de vibración. La naturaleza de largo alcance y bajo consumo de LoRa es adecuada para este tipo de comunicación en campo.

3.5. Vista de Proceso

Esta vista aborda los aspectos dinámicos del sistema, analizando cómo los objetos interactúan en tiempo de ejecución, cómo se maneja la concurrencia y cómo se sincronizan los distintos procesos.

3.5.1. Diagrama de Estados

Describe el comportamiento dinámico de un único objeto a lo largo de su vida. Modela los diferentes estados por los que un objeto puede pasar y los eventos que causan las transiciones entre ellos.



Se puede utilizar como ejemplo, el flujo de encendido, apagado y parpadeo de los LEDs. Esto dependiendo de que acción este transcurriendo. Ya sea, apagar o encender el DVA, o que este nos emita una señal de vibración.

4. Patrones de Diseño

A continuación se mostraran los patrones de diseño utilizados para la realización de este proyecto por clase:

4.1. `ControlPrincipalDVA`

- **Rol General:** Orquestador central del sistema.
- **Patrones:**
  - **Patrón Command**:
    - **Rol de Cliente (Configurador):** En su constructor, `ControlPrincipalDVA` instancia los objetos `ComandoConcreto` (`ComandoEncenderSeleccionados`, `ComandoApagarSeleccionados`) y les pasa una referencia de sí mismo (`*this`) como el `Receiver`.
    - **Rol de Invocador (botones físicos):** La lógica en `ejecutarCiclo()` que detecta la pulsación de `botonEncender` o `botonApagar` (`if (botonEncender.fuePresionado())`) y después llama a `cmdEncender->ejecutar` actúa como un `Invoker`. La petición del usuario (pulsar el botón) se traduce en la ejecución de un comando.
    - **Rol de Receptor:** Los métodos como `estaSwitchSeleccionado(byte)`, `enviarComandoLoRaYEsperarAck(...)`, `encenderLedConfirmacion(int)`, `apagarLedConfirmacion(int)` son las acciones que los comandos concretos invocan en el receptor (`ControlPrincipalDVA` mismo, a través de la referencia `controlPrincipal_ref` que tienen los comandos). Estos métodos contienen la lógica de negocio real.
    - **Rol de Invocador (Para comandos de la App Qt):** Los métodos `ejecutarComandoEncender_App()` y `ejecutarComandoApagar_App()` son invocados por `ComunicadorBluetooth`. Estos métodos después iteran sobre `dvaSeleccion` y realizan las acciones, comportándose de manera similar a un `Invoker` que ejecuta la lógica de encendido/apagado (que internamente es similar a lo que haría un objeto comando, pero está directamente implementada acá).
  - **Patrón Observer**:
    - **Rol de Sujeto (Subject):** `ControlPrincipalDVA` mantiene el estado de los DVAs (implícitamente a través del estado de los LEDs de confirmación y las respuestas

LoRa) y el estado de los switches físicos. Cuando estos estados cambian, debe notificar a los observadores.

- **Patrón Facade**:
  - `ControlPrincipalDVA`, a través de la interfaz que expone a `ComunicadorBluetooth` (ej. `actualizarSeleccionDVA_App`, `ejecutarComandoEncender_App`, `enviarTodosLosEstadosViaBT`), actúa como una fachada para la app Qt, proporcionando una forma simplificada de interactuar con el subsistema complejo del control DVA (que incluye LoRa, LEDs, switches, etc.).

4.2. `Comando` (Interfaz)

- **Rol General:** Define una interfaz para ejecutar las operaciones.
- **Patrones:**
  - **Patrón Command (Interfaz Command)**:
    - Su "composición interna" es su declaración de interfaz.
    - *Métodos clave:* `virtual void ejecutar() = 0;` que obliga a las subclases a implementar la acción. El destructor virtual `virtual ~Comando()` permite la correcta destrucción de objetos concretos a través de un puntero a la interfaz.

4.3. `ComandoEncenderSeleccionados` / `ComandoApagarSeleccionados`

- **Rol General:** Implementaciones para las acciones de encender o apagar.
- **Patrones:**
  - **Patrón Command (ConcreteCommand)**:
    - *Atributos clave:* `controlPrincipal_ref: ControlPrincipalDVA&`. Este atributo almacena la referencia al `Receiver` de la acción.
    - *Métodos clave:* El constructor recibe y almacena la referencia al `Receiver`. El método `ejecutar()` implementa la acción:
      1. Itera sobre los switches físicos usando `controlPrincipal_ref.estaSwitchSeleccionado(i)`.
      2. Para cada DVA seleccionado, invoca la acción en el `Receiver` (ej. `controlPrincipal_ref.enviarComandoLoRaYEsperarAck(...)`, `controlPrincipal_ref.encenderLedConfirmacion(...)`).
    - *Colaboradores:* El `ControlPrincipalDVA` (como `Receiver`).

4.4. `ComunicadorBluetooth`

- **Rol General:** Manejar la comunicación bidireccional con la app Qt.
- **Patrones:**
  - **Patrón Observer (Observer)**:
    - *Método de actualización:* El método `enviarMensaje(const String& mensaje)` es invocado por el `Subject` (`ControlPrincipalDVA`) cuando hay un cambio de estado que necesita ser comunicado a la app. Este método toma los datos del `Subject` (pasados como argumento o formateados por el `Subject`) y los transmite.

- *Atributos clave:* `SerialBT` (para enviar los datos), `controlPrincipal_ptr` (aunque no se usa para *recibir* la notificación del patrón Observer, sino para delegar comandos de la app).

4.5. Otras Clases (`Boton`, `PanelSwitches`, `DisplayLEDs`, `ComunicadorLoRa`)

Respecto a las clases restantes, su composición interna está más orientada a encapsular la interacción con hardware específico o una funcionalidad concreta, en lugar de implementar patrones de comportamiento complejos por sí mismas.

4.6. Diagramas por Patrones

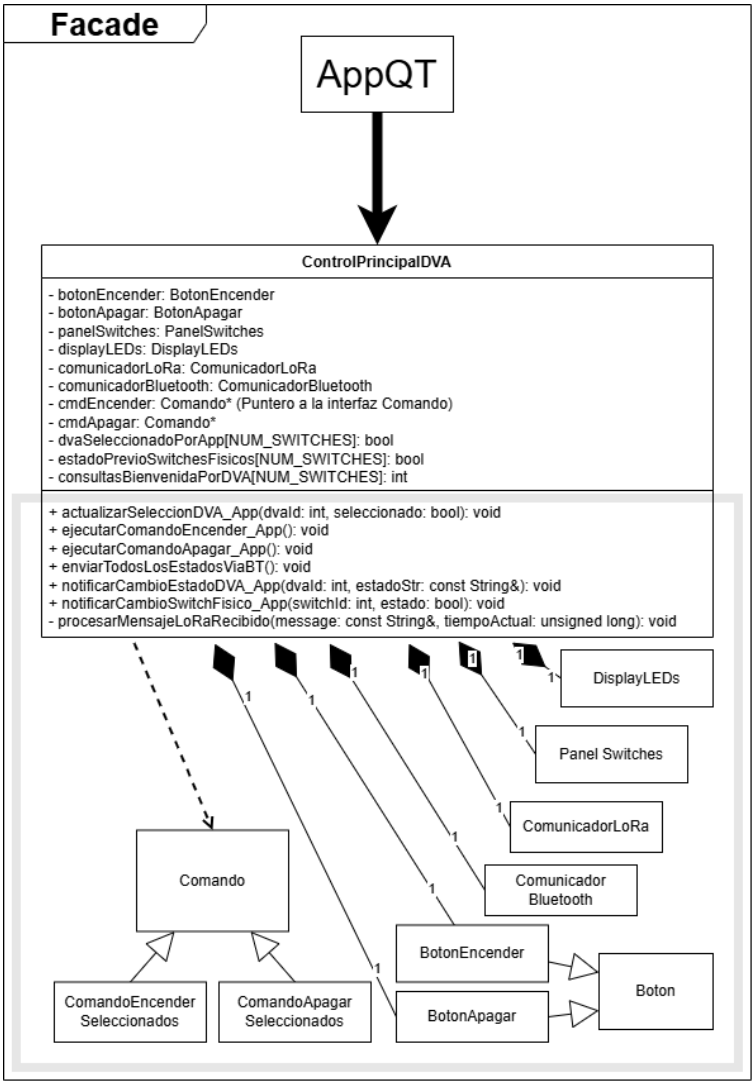


Figura 6: Facade

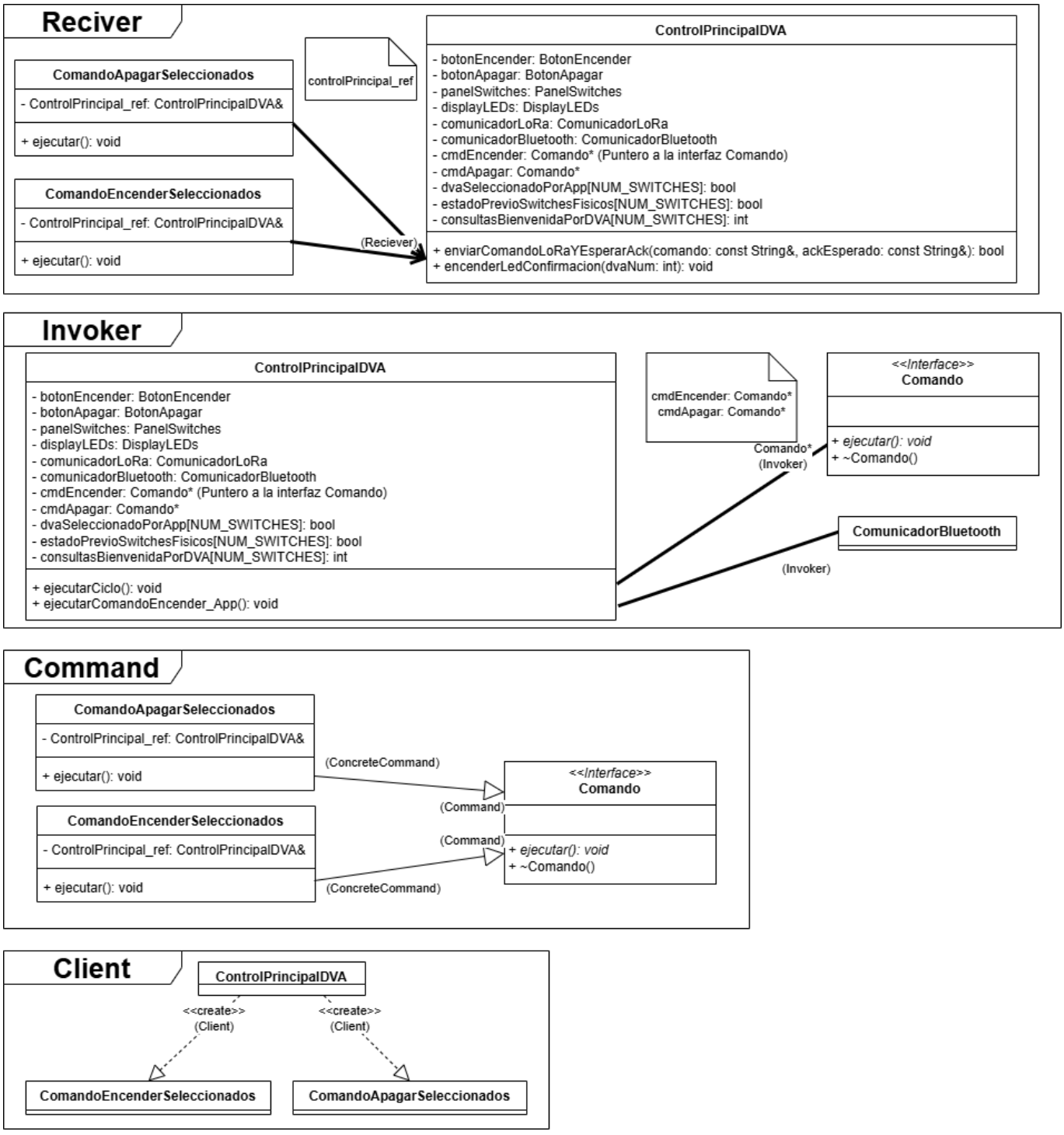


Figura 7: Command

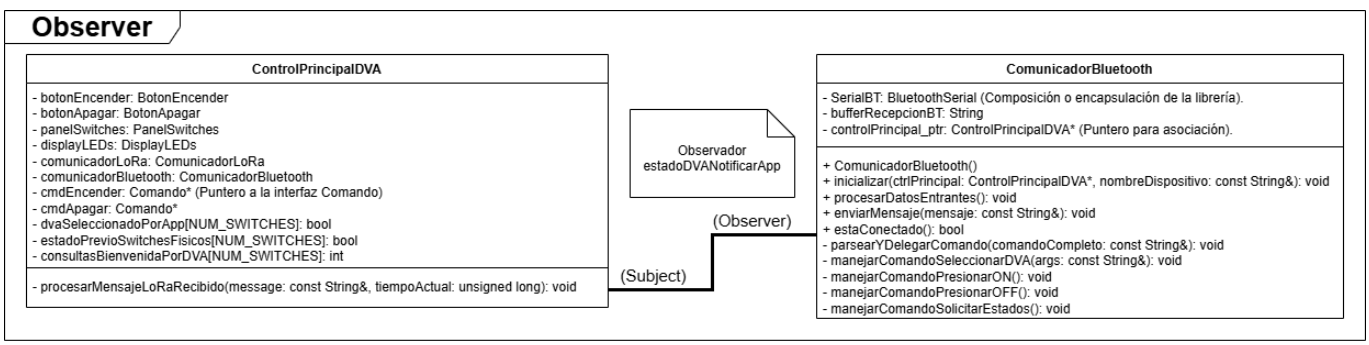
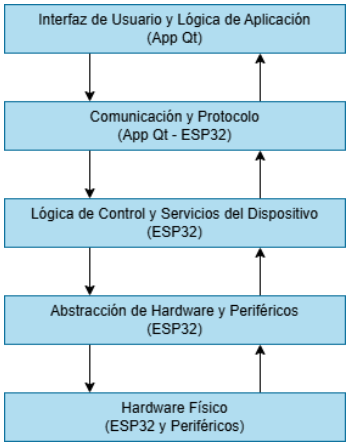


Figura 8: Observer

5. Estilo Arquitectónico de Capas Jerárquicas (Sistema Completo)

El sistema de Control DVA, incluyendo tanto el dispositivo ESP32 como la aplicación Qt, fue realizada utilizando un estilo arquitectónico de capas jerárquicas. Realmente no se hizo pensando en este estilo arquitectónico, ya que se realizo antes de darse la clase donde se presentaba, pero es el estilo arquitectónico que más se le parece. Este estilo organiza el sistema en un conjunto de capas, donde cada una provee servicios a la capa inmediatamente superior y utiliza los servicios de la capa inmediatamente inferior. La interacción entre capas adyacentes se define mediante protocolos o interfaces claras.



Para el sistema, se identifican las siguientes capas principales, abarcando desde la interacción con el usuario en la aplicación Qt hasta el hardware físico del ESP32:

Capa 1: Interfaz de Usuario y Lógica de Aplicación (App Qt)

- *Responsabilidades:* Esta es la capa más alta, con la que el usuario interactúa directamente. Se encarga de presentar la información del estado de los DVAs y los switches, capturar las acciones del usuario (selección de DVAs, comandos de encender/apagar, configuración de parámetros LoRa), y la lógica de presentación y validación de la aplicación cliente.
- *Clases/Componentes Clave (Qt):* Archivos QML para la interfaz visual (`Main.qml`), la clase `AppController` (actuando como ViewModel o Presenter para la lógica de la UI y el estado de la aplicación cliente).
- *Interacción:* Utiliza los servicios de la Capa de Comunicación para enviar comandos al dispositivo ESP32 y recibir actualizaciones de estado. Provee la interfaz al usuario final.

Capa 2: Comunicación y Protocolo (App Qt - ESP32)

- *Responsabilidades:* Implementa el protocolo de mensajes definido para la comunicación entre la aplicación Qt y el ESP32. Gestiona el establecimiento y mantenimiento de la conexión (Bluetooth), el formateo de los mensajes salientes y el parseo de los mensajes entrantes. Actúa como el puente entre los dos sistemas principales.
- *Clases/Componentes Clave:*
  - En la App Qt: La clase `BluetoothManager` (para Bluetooth).
  - En el ESP32: La clase `ComunicadorBluetooth`.
  - El propio medio de comunicación (ondas Bluetooth).

- *Interacción:* Provee servicios de envío y recepción de mensajes formateados a la "Capa de Interfaz de Usuario y Lógica de Aplicación (App Qt)" y a la "Capa de Lógica de Control y Servicios del Dispositivo (ESP32)". Utiliza los servicios de comunicación de bajo nivel del sistema operativo (Qt) o del hardware (ESP32).

Capa 3: Lógica de Control y Servicios del Dispositivo (ESP32)

- *Responsabilidades:* Es el cerebro del dispositivo ESP32. Contiene la lógica principal para controlar los DVAs, procesar los comandos recibidos (tanto de los botones físicos como de la app remota), gestionar el estado de cada DVA, coordinar la comunicación LoRa con los DVAs en campo, y manejar las interacciones con los periféricos de entrada/salida.
- *Clases/Componentes Clave (ESP32):* `ControlPrincipalDVA`, las clases de `Comando` y sus derivadas (`ComandoEncenderSeleccionados`, `ComandoApagarSeleccionados`), y `ComunicadorL`
- *Interacción:* Utiliza los servicios de la "Capa de Comunicación" para interactuar con la app Qt y los servicios de la "Capa de Abstracción de Hardware" para interactuar con los componentes físicos. Provee la lógica de control principal del sistema.

Capa 4: Abstracción de Hardware y Periféricos (ESP32)

- *Responsabilidades:* Provee una interfaz simplificada y de más alto nivel para interactuar con los componentes de hardware específicos del ESP32. Encapsula los detalles de bajo nivel del manejo de los switches (multiplexor), el display de LEDs (controlador HT16K33), y las librerías base de comunicación (LoRa, Bluetooth a nivel de librería Arduino si no estuviera en la capa de Comunicación).
- *Clases/Componentes Clave (ESP32):* `PanelSwitches`, `DisplayLEDs`, `Boton` (y derivadas), las librerías de Arduino para I2C, SPI, GPIO, y la librería LoRa de bajo nivel.
- *Interacción:* Ofrece servicios a la "Capa de Lógica de Control y Servicios del Dispositivo". Interactúa directamente con la "Capa de Hardware Físico".

Capa 5: Hardware Físico (ESP32 y Periféricos)

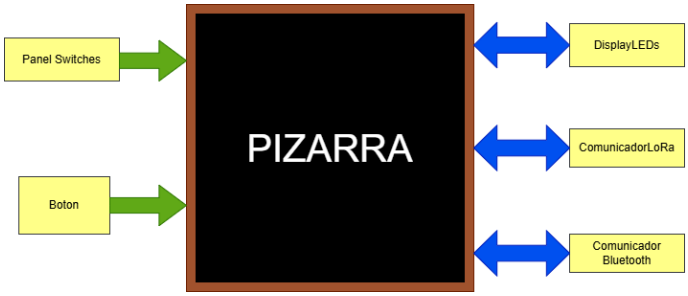
- *Responsabilidades:* Es la capa más baja, compuesta por los componentes electrónicos reales.
- *Clases/Componentes Clave:* El microcontrolador ESP32-WROOM-32, los switches físicos (palancas), los LEDs, el multiplexor, el controlador HT16K33, el módulo LoRa SX1278, antenas, batería, etc.
- *Interacción:* Es la base sobre la cual operan todas las capas de software del dispositivo ESP32.

También se pudo haber resuelto con otros estilos arquitectónicos, como el de **Datos Compartidos** versión pizarrón. Se mantendrían ciertas clases pero cada una escribiría o tomaría datos de este, no interactuarían entre si, sino que dependiendo que haya en la pizzarra, cambiarían de comportamiento. Por ejemplo:

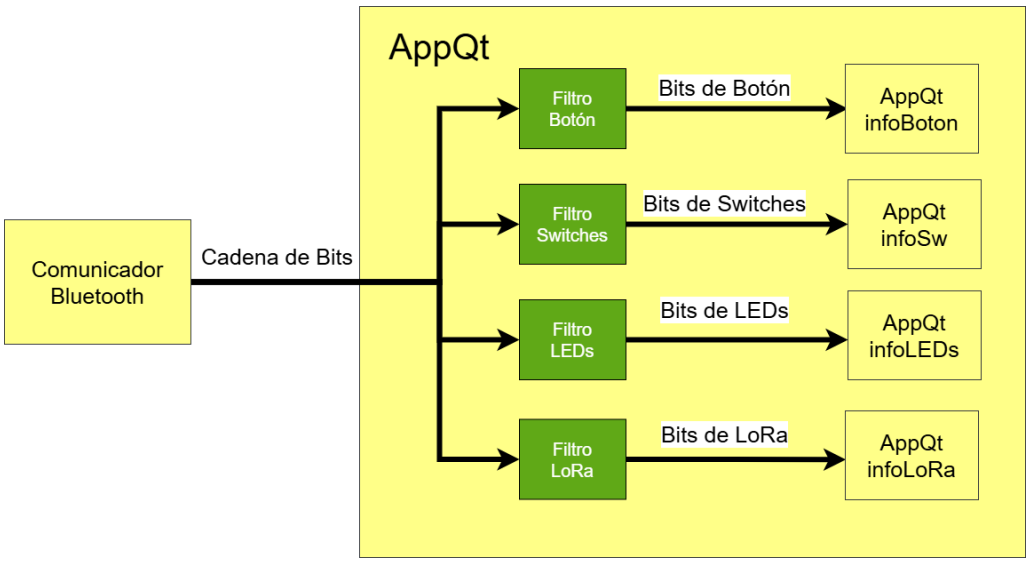
- **Botón:** Podría escribir un vector de tamaño 2 (apagado, encendido) que indique cual botón fue seleccionado, 0 para no seleccionado y 1 para seleccionado.
- **PanelSwitches:** Podria escribir un vector de tamaño 8 (cantidad máxima de DVAs) en el que indique cual es el estado de cada switch, con 0 para deseleccionado y 1 para seleccionado.

- **ComunicadorLoRa:** Tomaría los datos que dejan el `Botón` (para saber si hay que mandar encender o apagar) y de `PanelSwitches` cuales DVA realizaran la acción. Además debería de tomar los datos de `ComunicadorBluetooth` en caso de que se realice alguna acción de ahí. Escribiría los mensajes mandados por el Controlador DVA, esto ya se puede hacer de varias formas, por ejemplo, los mensajes que nos envía el Controlador son siempre los mismos (confirmación de encender y apagar, y el de vibración), podemos hacer que en la pizarra escribir 2 bits que representes las 3 opciones.
- **DisplayLEDs:** Leería los datos dejados por `PanelSwitches` para la primera barra y los datos dejados por `ComunicadorLoRa` para la segunda barra. Quizás también debería de escribir los estados de las barras, para la sincronización con `ComunicadorBluetooth`.
- **ComunicadorBluetooth:** Debería de tomar los datos de `DisplayLEDs`. Debería de escribir una combinación de `Botones` y `PanelSwitches`.

El diagrama quedaría así:



También se podría utilizar el estilo arquitectónico de **Tubos y Filtros** en la parte de la app en Qt. Esto tendría sentido en caso de que se len envíe más información a la app. Por ejemplo, no solo se enviaría el estado de las barras de los leds, sino también, cuando se realice alguna acción en el Hardware o llegue algún mensaje. Como lo he dicho antes en el pizarrón, se podría enviar una cadena de bits que contengan la información del estado de los botones, de los Leds, de los Switches y mensaje LoRa. Acá entrarían los tubos y los filtros, que filtrarían esta cadena de bits separandolo por sección y luego filtrando lo que no es destacable.
A continuación se mostrará un diagrama que visualice esto:



6. Proceso de Desarrollo

Analizar esto es complicado, debido a ser un proyecto previo a la realización de esta materia, lo que provoco grandes cambios en la planificación de este, además de que todavía no se llego a un feedback con el usuario y esto recién sería un prototipo.
Podríamos pensarlo como el **Modelo de Desarrollo en Cascada** analizando las etapas:

1. **Especificación de requerimientos o requisitos:** Al empezar con el proyecto se hablo con el usuario para establecer requisitos.
2. **Diseño del software:** Se pensó para ser utilizado con Arduino-IDE, con expectativa de pasarlo a ESP-IDF, y con la idea que sea usado solo como Hardware. Gracias a esta materia se agrego la idea de usarlo tambien por applicación, siendo Qt.
3. **Construcción o Implementación del software:** Inicialmente estaba hecho sin clases en un único archivo de Arduino-IDE, luego se cambio para que funcione con clases. Actualmente estariamos acá.
4. **Integración**
5. **Pruebas (o validación)**
6. **Despliegue (o instalación)**
7. **Mantenimiento**

También, al no llegar aún a un prototipo al que probar, se podría pensar como **Desarrollo Evolutivo**. Este proyecto me da una gran libertad, realmente solo se me pidió encender DVA a distancia que estén a una distancia máxima de 70 metros y enterrados como máximo 50cm, por lo que yo agregue varíos de los requisitos para que sea más amigable al usuario, como agregar unos LEDs de visualización o la aplicación en Qt. A lo largo del desarrollo se me fueron ocurriendo ideas para mejorar el proyecto, además, con la app en Qt puedo realizar otras ideas que me pueden facilitar la realización de pruebas, como puede ser modificar los parámetros del LoRa (la forma en la que envía los mensajes) a distancia sin la necesidad de desarmar el prototipo y reprogramarlo. Hasta este momento soy yo el que esta explorando los requisitos hasta llegar a un sistema final (**Desarrollo Exploratorio)**, pero a la vez, ya se los requisitos del usuario y estoy trabajando para mejorar la calidad de estos (**Desarrollo en prototipos)**.

7. Conclusiones

El desarrollo del sistema “Control de DVA” ha permitido integrar soluciones de hardware y software bajo una arquitectura coherente y documentada. La aplicación del modelo de “4+1 Vistas” de Philippe Kruchten resultó fundamental para descomponer la complejidad del problema, permitiendo abordar por separado los desafíos de la interfaz de usuario, la lógica de control embebida y la comunicación inalámbrica.

A través de la Vista Lógica, se demostró cómo el uso de Patrones de Diseño no es meramente teórico, sino que resuelve problemas prácticos de mantenibilidad. Específicamente, el patrón Command permitió desacoplar la interfaz física (botones) de la lógica de ejecución, facilitando la futura incorporación de nuevos comandos sin modificar el hardware. Asimismo, el patrón Observer fue clave para mantener la consistencia entre el estado del dispositivo físico y la interfaz gráfica en Qt.

En cuanto a la estructura general, la elección del Estilo Arquitectónico de Capas Jerárquicas validó su eficacia para este tipo de sistemas embebidos, permitiendo una clara separación de responsabilidades entre la abstracción del hardware (ESP32) y la lógica de presentación (App Qt).

Finalmente, el análisis del proceso de desarrollo reveló que, en proyectos donde los requisitos de hardware y usuario se descubren progresivamente, un enfoque Evolutivo y basado en Prototipos es superior al modelo en Cascada tradicional, ya que permite validar tempranamente decisiones críticas como el alcance de la comunicación LoRa y la usabilidad de la aplicación móvil.

8. Bibliografía

- Kruchten, P. (1995). Architectural Blueprints-The “4+1” View Model of Software Architecture. IEEE Software, 12(6), 42-50.
- Bass, L., Clements, P., & Kazman, R. (2003). Software Architecture in Practice (2nd Ed.). Addison-Wesley.
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley.
- Miles, R., & Hamilton, K. (2006). Learning UML 2.0. O’Reilly Media.
- Stroustrup, B. (1997). The C++ Programming Language (3rd Ed.). Addison-Wesley.
- Deitel, H. M., & Deitel, P. J. (2008). C++ Cómo Programar (6ta Ed.). Pearson Educación.