SISTEMA DE CONTROL PARA HORNO DE REFLUJO

Documentación Técnica de Software Android

Arquitectura: MVVM (Model-View-ViewModel)
Lenguaje: Kotlin / Jetpack Compose
Protocolo: TCP Sockets (Raw ASCII)

Departamento de Ingeniería en Automatización
9 de diciembre de 2025

Índice general

Capítulo 1

Visión General del Sistema

1.1 Introducción

El presente documento detalla la arquitectura de software, la dinámica de ejecución y los protocolos de comunicación de la aplicación móvil Reflow Oven Controller". Este sistema permite la supervisión remota y el control de lazo cerrado de un horno de soldadura mediante una interfaz gráfica reactiva y comunicación de red en tiempo real.

1.2 Alcance Técnico

El software ha sido diseñado bajo los principios de *Clean Architecture* simplificada, implementando el patrón **MVVM**. Sus responsabilidades principales incluyen:

1. **Gestión de Conectividad:** Establecimiento y mantenimiento de sockets TCP persistentes.
2. **Visualización de Datos:** Renderizado de gráficos de temperatura vs. tiempo a 60 FPS.
3. **Inyección de Perfiles:** Serialización y transmisión de etapas de soldadura (Soak, Reflow).
4. **Concurrencia:** Manejo de hilos en segundo plano para operaciones de E/S.

Capítulo 2

Arquitectura de Software

La aplicación se estructura en tres capas lógicas claramente diferenciadas para garantizar la escalabilidad y testabilidad.

2.1 Capa de Presentación (UI Layer)

Implementada utilizando **Jetpack Compose**. Esta capa es puramente reactiva; no contiene lógica de negocio, solo lógica de renderizado basada en estados.

- `MainActivity.kt`: Punto de entrada del sistema Android. Configura el tema y el contenedor de superficie.
- `DashboardScreen.kt`: Composable principal. Observa los flujos (*Flows*) del ViewModel y redibuja la interfaz cuando los datos cambian.
- `OvenChart.kt`: Wrapper de interoperabilidad que incrusta una vista de Android clásica (`MPAndroidChart`) dentro de la jerarquía de Compose.

2.2 Capa de Lógica de Negocio (ViewModel Layer)

`MainViewModel.kt` actúa como el cerebro de la aplicación.

- Transforma los datos crudos del repositorio en estados consumibles por la UI (`StateFlow`).
- Gestiona el ciclo de vida de las corrutinas (`viewModelScope`), asegurando que no haya fugas de memoria si la vista se destruye.
- Contiene la lógica de limitación del historial gráfico (buffer circular de 120 puntos).

2.3 Capa de Datos (Data Layer)

Responsable de la comunicación con el mundo exterior.

- `ReflowOvenRepository.kt`: Patrón repositorio que abstrae la fuente de datos.
- `WiFiService.kt`: Implementación de bajo nivel de `CommunicationService`. Maneja los `Sockets`, `PrintWriters` y `Scanners`.

Capítulo 3

Dinámica de Ejecución y Trazabilidad

Esta sección es el núcleo técnico del documento. Se desglosa la jerarquía de llamadas funcionales para los procesos críticos del sistema.

3.1 Traza 1: Ciclo de Monitoreo (Polling Loop)

Este proceso es responsable de mantener la gráfica y los textos de estado actualizados en tiempo real.

Traza de Ejecución: Actualización de Estado del Horno

Contexto: `MainViewModel` inicia una corrutina tras una conexión exitosa.

1. **`MainViewModel`**:
   - Llama a `repository.getOvenState()`.
   - Inicia bloque `collect` (observador) sobre el flujo retornado.
2. ↪ **`ReflowOvenRepository`**:
   - Redirige la llamada a `communicationService.getOvenState()`.
3. ↪ **`WiFiService`** (Hilo: `Dispatchers.IO`):
   - Entra en bucle `while (isConnected)`.
   - `sendCommand("STATUS?\n")`: Escribe en el Socket.
   - `reader.nextLine()`: **Bloqueo** esperando respuesta del hardware.
   - *Parsing*: Divide la cadena recibida por delimitadores ";".
   - Instancia `OvenState(temp, target, stage...)`.
   - `emit(ovenState)`: Envía el objeto a través del flujo.
   - `delay(1000)`: Espera no bloqueante antes del siguiente ciclo.
4. ← *Retorno a* **`MainViewModel`** (Hilo: `Main`):
   - Recibe `OvenState`.
   - Actualiza `_ovenState.value` → Dispara recomposición de Textos.
   - Llama a función interna `addGraphPoint(temp)`.
     - Añade punto a `_tempHistory`.
     - Si `size >120`, elimina el índice 0 (FIFO).
     - Actualiza `_tempHistory.value`.
5. ← *Reacción en* **`DashboardScreen`**:
   - `collectAsState` detecta cambio en `tempHistory`.
   - Pasa los nuevos datos al composable `OvenChart`.
6. ↪ **`OvenChart`**:
   - Bloque `update`: Recibe los puntos nuevos.
   - Actualiza el `LineDataSet` de la librería MPAndroidChart.
   - Llama a `chart.notifyDataSetChanged()` e `invalidate()`.

3.2 Traza 2: Inyección de Perfil (Comando START)

Proceso desencadenado cuando el usuario selecciona un perfil y pulsa "Start".

Traza de Ejecución: Envío de Perfil de Soldadura

Contexto: Interacción de usuario en `DashboardScreen`.

1. `DashboardScreen`:
   - Callback `onProfileSelected`.
   - Llama a `viewModel.startOven(reflowProfile)`.
2. ↪ `MainViewModel`:
   - `launch`: Inicia nueva corrutina.
   - Limpieza: `_tempHistory.value = emptyList()`.
   - Reinicio: `timeIndex = 0f`.
   - Llama a `repository.startOven(profile)`.
3. ↪ `ReflowOvenRepository`:
   - Delega a `communicationService.startOven(profile)`.
4. ↪ `WiFiService` (Hilo: `Dispatchers.IO`):
   - **Serialización**: Transforma el objeto `ReflowProfile` en cadena.
   - Construye string: `"START;` ` stage.temp + ";` ` stage.time ...`
   - Llama a `sendCommand(string)`.
     - `writer.print(command)`.
     - `writer.flush()`: Fuerza el envío del paquete TCP inmediatamente.

Capítulo 4

Especificación del Protocolo de Comunicación

El sistema utiliza un protocolo basado en texto ASCII sobre TCP/IP. La aplicación actúa como Cliente TCP y el Horno como Servidor TCP.

4.1 Formato de Comandos (Cliente → Horno)

Los comandos finalizan con un carácter de nueva línea (`\n`).

Comando	Descripción y Estructura
`STATUS?`	Solicita el estado actual. No requiere parámetros.
`STOP`	Detiene el proceso de calentamiento inmediatamente.
`START;<Stages>`	Inicia el horno con un perfil específico. La estructura de `<Stages>` es una lista serializada de temperatura y duración. *Ejemplo:* `START;Soak;150;90;Reflow;245;30`
`PROFILE;<Stages>`	Carga un perfil en memoria sin iniciar el proceso inmediatamente. Sigue la misma serialización que START.

Tabla 4.1: Tabla de Comandos de Control

4.2 Formato de Respuesta (Horno → Cliente)

El horno responde al comando `STATUS?` con una cadena delimitada por punto y coma (;).

Estructura de Trama STATUS

```
STATUS;CurrentTemp;TargetTemp;StageName;TimeElapsed
```

Mapeo de Datos en `WiFiService.kt`:

- `parts[0]` → Cabecera ("STATUS")
- `parts[1]` → `currentTemperature` (Float)

- `parts[2]` → `targetTemperature` (Float)
- `parts[3]` → `stage` (String, e.g., "Soak")
- `parts[4]` → `timeElapsed` (Long)

Capítulo 5

Gestión de Concurrencia y Memoria

5.1 Uso de Corrutinas

Kotlin Coroutines se utiliza para evitar el bloqueo del *Main Thread* (UI Thread).

- **Dispatchers.IO:** Utilizado exclusivamente en `WiFiService`. Optimizado para operaciones de bloqueo de E/S como `socket.read()` o `writer.flush()`.
- **ViewModelScope:** Define el alcance de las corrutinas. Si el usuario cierra la aplicación o cambia de pantalla, este scope cancela automáticamente las peticiones de red pendientes, evitando *memory leaks*.

5.2 Optimización de Gráficos

Dado que el horno puede operar por largos periodos, la acumulación de puntos en la gráfica representa un riesgo de consumo de memoria.

```
private fun addGraphPoint(temp: Float) {
    val currentList = _tempHistory.value.toMutableList()
    currentList.add(Entry(timeIndex, temp))
    timeIndex += 1f

    // Protección de Memoria: Ventana deslizante
    if (currentList.size > 120) {
        currentList.removeAt(0)
    }
    _tempHistory.value = ArrayList(currentList)
}
```

Listing 5.1: Lógica de Buffer Circular en MainViewModel

Esta lógica asegura que la aplicación mantenga un consumo de RAM constante (O(1)) independientemente de la duración del proceso de soldadura.

Capítulo 6

Diccionario de Funciones y Lógica Operativa

En este capítulo se desglosa la totalidad de las funciones implementadas en el sistema, detallando su lógica interna, dependencias y efectos colaterales en el flujo de ejecución.

6.1 Módulo: MainViewModel (Lógica de Negocio)

Este componente orquesta la interacción entre la UI y los datos. Todas las funciones aquí se ejecutan inicialmente en el *Main Thread* pero delegan operaciones pesadas a corrutinas.

connect(ip, port)

Propósito: Inicia el intento de conexión con el hardware y, si tiene éxito, arranca el monitoreo.
Lógica de Llamadas:

1. Lanza corrutina en `viewModelScope`.
2. Invoca `repository.connect(ip, port)`.
3. Recolecta el flujo booleano resultante.
4. *Si es True:*
   - Cancela trabajos anteriores (`ovenStateJob?.cancel`).
   - Reinicia gráficas (`_tempHistory = empty`).
   - Inicia nueva corrutina interna que llama a `repository.getOvenState()` para comenzar el polling.
5. *Si es False:* Cancela cualquier job activo.

disconnect()

Propósito: Cierra la conexión y limpia el estado de la UI.
Lógica de Llamadas:

1. Cancela la corrutina de monitoreo (`ovenStateJob`).
2. Invoca `repository.disconnect()` (función suspendida).
3. Fuerza el estado `_isConnected` a `false`.
4. Resetea `_ovenState` a valores por defecto (Idle).

startOven(profile)

Propósito: Envía un perfil de soldadura y ordena el inicio del proceso.
Lógica de Llamadas:

1. Limpia inmediatamente el historial gráfico (`_tempHistory`).
2. Resetea el contador `timeIndex` a 0.
3. Lanza corrutina que invoca `repository.startOven(profile)`.

addGraphPoint(temp)

Propósito: Función auxiliar privada para gestión de memoria de la gráfica.
Lógica Interna:

1. Clona la lista actual de puntos.
2. Crea un nuevo objeto `Entry(x, y)`.
3. Verifica si `lista.size >120`. Si es cierto, ejecuta `removeAt(0)` (FIFO).
4. Reasigna la lista al `StateFlow`, notificando a la UI.

6.2 Módulo: WiFiService (Comunicación de Bajo Nivel)

Clase encargada de la manipulación directa de Sockets y Streams. Todas las funciones operan bajo el contexto `Dispatchers.IO`.

connect(ip, port)

Propósito: Establece el túnel TCP.
Lógica Interna:

1. Instancia `java.net.Socket(ip, port)`.
2. Vincula `PrintWriter` al *OutputStream* (para enviar comandos).
3. Vincula `Scanner` al *InputStream* (para leer respuestas).
4. Emite `true` si no hay excepciones; de lo contrario captura excepción y emite `false`.

getOvenState()

Propósito: Bucle infinito (mientras haya conexión) de petición-respuesta.
Lógica de Llamadas:

1. Verifica `socket.isConnected`.
2. Llama a método privado `sendCommand("STATUS?")`.
3. Bloquea hilo esperando `reader.nextLine()`.
4. Procesa string raw: `"STATUS;25.5;150.0;Soak;120"`.
5. Mapea a objeto `OvenState`.
6. `emit(state)` hacia el repositorio.
7. `delay(1000)` para evitar saturación de red.

sendCommand(cmd)

Propósito: Envío físico de bytes por la red.
Lógica Interna:

1. Verifica nulidad del `writer`.
2. Ejecuta `writer.print(cmd)`.
3. Ejecuta `writer.flush()` obligatoriamente para vaciar el buffer de salida del sistema operativo.
4. Maneja posibles `IOException` logueando el error.

sendProfile(profile)

Propósito: Serialización de objeto complejo a protocolo ASCII.
Lógica Interna:

1. Itera sobre `profile.stages`.
2. Construye string usando `joinToString` con delimitador punto y coma.
3. Prefija el comando "PROFILE;".
4. Llama a `sendCommand()`.

6.3 Módulo: ReflowOvenRepository (Patrón Fachada)

Actúa como intermediario puro para aislar la implementación de red.

Funciones Pasarela

(`connect`, `disconnect`, `startOven`, etc.)
Lógica: Estas funciones no contienen lógica de negocio propia. Su única función es reenviar los parámetros recibidos hacia la instancia de `CommunicationService`. Esto permite que, en el futuro, si se cambia WiFi por Bluetooth, el `ViewModel` no requiera modificaciones, solo se inyecta un servicio diferente en este repositorio.

6.4 Módulo: DashboardScreen (Interfaz de Usuario)

Funciones Composable que gestionan la renderización y captura de eventos.

ConnectionDialog

Propósito: Modal para ingreso de credenciales de red.
Lógica de Eventos:

1. Mantiene estado local temporal para IP y Puerto (`remember`).
2. Al pulsar Çonnect":
   - Valida conversión de puerto a Int.
   - Ejecuta callback `onConnect(ip, port)`.
   - Cierra el diálogo.

CustomProfileDialog

Propósito: Formulario para creación de perfiles ad-hoc.
Lógica Interna:

1. Captura 5 campos de texto (Nombre, Temp Soak, Tiempo Soak, Temp Reflow, Tiempo Reflow).
2. Al confirmar, instancia un nuevo objeto `ReflowProfile`.
3. Devuelve el objeto creado al padre mediante `onProfileCreated(it)`.

OvenChart (Update Block)

Propósito: Actualización reactiva de la vista de Android clásica.
Lógica de Llamadas:

1. Se ejecuta cada vez que cambia la lista `points`.
2. Si la lista no está vacía:
   - Obtiene referencia al `LineDataSet` existente.
   - Inyecta la nueva lista de puntos.
   - Notifica cambios a la librería gráfica para forzar el repintado (`invalidate`).
3. Si la lista está vacía, limpia el gráfico (`chart.clear`).

Capítulo 7

Conclusión

La arquitectura presentada desacopla exitosamente la lógica de control crítica (Wi-Fi/TCP) de la interfaz de usuario moderna (Compose). La distribución de llamadas demuestra un flujo de datos unidireccional (UDF), donde los eventos fluyen hacia arriba (UI $\rightarrow$ ViewModel $\rightarrow$ Repositorio) y los datos fluyen hacia abajo mediante flujos reactivos, resultando en un sistema robusto, mantenible y profesional.