

# Metodología para el desarrollo de transiciones visuales secuenciales en Android TV

Miguel Ángel Neyoy Ceniceros, Jesús Rodolfo Carvajal Lizárraga,

Juan Daniel Vázquez García, Modesto Alonso Alcaraz Duran

*Facultad de Informática Mazatlán Universidad Autónoma de Sinaloa Mazatlán, México*

## Resumen

Esta investigación establece una metodología estructurada para el desarrollo de aplicaciones con transiciones visuales secuenciales en el entorno de Android TV. El trabajo responde a la fragmentación del mercado de Smart TV y la necesidad de enfoques de desarrollo que aseguren rendimiento y compatibilidad. La metodología propuesta emplea Kotlin y el framework Jetpack Compose for TV, los cuales son recomendados por Google, y se validan mediante un prototipo de carrusel de imágenes. La implementación demostró la importancia de la gestión del estado reactivo (remember, mutableStateOf) para mantener la estabilidad de las transiciones visuales. Se confirmó la efectividad de la carga asíncrona de recursos remotos (librería Coil) para garantizar un flujo visual continuo, evitando bloqueos del hilo principal. Finalmente, el uso del operador módulo en la lógica de navegación garantiza secuencias cíclicas y estables. Esta metodología resultante proporciona una guía replicable para construir aplicaciones fluidas y coherentes en pantallas de gran formato.

**Palabras clave:** Android TV, Jetpack Compose, Kotlin, Smart TV.

## Abstract

This research establishes a structured methodology for developing applications with sequential visual transitions within the Android TV environment. The work addresses the fragmentation of the Smart TV market and the need for development approaches that ensure performance and compatibility. The proposed methodology employs Kotlin and the Jetpack Compose for TV framework, which are recommended by Google, and is validated through an image carousel prototype. The implementation demonstrated the importance of reactive state management (remember, mutableStateOf) in maintaining the stability of visual transitions. The effectiveness of asynchronous loading of remote resources (Coil library) was confirmed to ensure a continuous visual flow, avoiding main thread blocking. Finally, the use of the modulo operator in the navigation logic guarantees cyclic and stable sequences. This resulting methodology provides a replicable guide for building fluid and consistent applications on large-format screens.

**Keywords:** Android TV, Jetpack Compose, Kotlin, Smart TV.

## I. INTRODUCCIÓN

Las Smart TV han transformado el entretenimiento doméstico en una experiencia interactiva y conectada; con un mercado global en 2024 de aproximadamente 8.70 millones de unidades en América Latina y con una proyección calculada de crecimiento de 2025-2034 alcanzando los 13,85 millones de unidades [1].

Estos dispositivos no solo reproducen contenido tradicional, sino que integran aplicaciones de transmisión de contenido en línea(streaming), abriendo la posibilidad de conectarse a otros dispositivos físicos como teléfonos, computadoras o software por red, es decir, un ecosistema IoT [2]; a esto se le conoce como Smart TV. Impulsando un crecimiento exponencial en el desarrollo de software. Sin embargo, los diferentes actores del mercado de las Smart TV junto con su propio sistema operativo como lo son web OS(LG), Tizen OS(Samsung), Android TV(Google), Roku OS(Roku); generan diversos desafíos significativos [5] para los desarrolladores, incluyendo incompatibilidades, diferencias de rendimiento y portabilidad. En respuesta a esta fragmentación del mercado, las empresas han desarrollado su propio kit de desarrollo de software (SDK por sus siglas en inglés) ofreciendo entornos de programación específicos que buscan simplificar el desarrollo y reducir costos. Acorde al análisis del mercado de los televisores inteligentes realizado por Market Research Future [3], se observa un crecimiento exponencial, destacando a Android TV como una tendencia tecnológica adoptada por diversas empresas del mercado.

La disparidad de hardware y la gestión ineficiente de recursos en los sistemas operativos de televisores actuales provocan que las transiciones visuales complejas sufran de latencia y bloqueos en el hilo principal (UI). Esto resulta en una experiencia de usuario fragmentada y en un aumento considerable en los tiempos de desarrollo al intentar adaptar interfaces móviles a pantallas de gran formato. Surge entonces la siguiente pregunta de investigación: ¿Cómo se puede sistematizar el desarrollo de transiciones visuales en Android TV para garantizar un rendimiento estable y fluido independientemente de las limitaciones del hardware? Esta investigación es necesaria para estandarizar el uso de herramientas modernas (Jetpack Compose) en un entorno poco documentado como Android TV. Su importancia radica en la reducción de la deuda técnica asociada al mantenimiento de código XML heredado y en la optimización del consumo de memoria en dispositivos de gama media, lo cual impacta directamente en la calidad del producto final entregado al consumidor.

Se realizó una investigación aplicada con enfoque experimental. Se buscó resolver un problema práctico de la industria mediante la implementación y validación técnica de una nueva metodología de desarrollo, contrastando el desempeño esperado contra el observado en un entorno controlado. Por lo anterior esta investigación se planteó con el objetivo de proporcionar una metodología para desarrollar una aplicación. El resto del artículo se organiza de la siguiente manera: la Sección 2 detalla

los materiales y la configuración del entorno de pruebas; la Sección 3 describe la implementación del código y la lógica de navegación; la Sección 4 presenta los resultados obtenidos y la validación del prototipo; y finalmente, la Sección 5 ofrece las conclusiones y trabajo futuro.

## II. METODOLOGÍA

La metodología adoptada para esta investigación se divide en tres fases secuenciales: 1) Definición del Entorno Experimental y Selección de Herramientas, 2) Diseño de la Arquitectura de Transición Secuencial, y 3) Implementación y Validación del Prototipo. A continuación, se detalla cada fase.

### *II-A. Fase 1: Entorno Experimental*

Para garantizar la reproducibilidad de los resultados, se estableció un entorno de desarrollo basado en las recomendaciones oficiales de Google para "Building for TV". Se seleccionó Kotlin como lenguaje de programación por su interoperabilidad, manejo seguro de nulos y expresividad, lo cual permite escribir código más claro y seguro en comparación con Java. Para la interfaz de usuario, se implementó el framework Jetpack Compose for TV junto con las librerías de Leanback, facilitando la creación de interfaces optimizadas para pantallas de gran formato. Como entorno de desarrollo integrado (IDE), se utilizó Android Studio, aprovechando herramientas específicas como el sistema de compilación basado en Gradle y la función de edición en vivo (Live Edit) para la actualización inmediata de elementos en emuladores y dispositivos físicos. El equipo de cómputo utilizado para la experimentación y compilación del proyecto cuenta con las siguientes características técnicas, las cuales aseguran un rendimiento fluido del entorno de virtualización:

- Sistema Operativo: Windows 11 x64 Bits.
- Procesador: AMD Ryzen 7 8700F.
- Memoria: 16GB RAM DDR5.
- Gráficos: Tarjeta Gráfica GTX 1070.
- Almacenamiento: Unidad de Estado Sólido (SSD) de 480GB.

Para la replicación de esta metodología, es necesario cumplir con los estándares de hardware definidos por el SDK de Google. Los requisitos mínimos y recomendados para la configuración del entorno se detallan en la Tabla 1.

Tabla I  
TABLA 3X5 CON TEXTOS LARGOS USANDO TABULARX

Requisito	Mínimo	Recomendado
Sistema Operativo	Windows 10, MacOS 12, Cualquier distribución de Linux de 64 bits que sea compatible con Gnome, KDE o Unity DE; GNU C Library (glibc) 2.31 o versiones posteriores.	La versión más reciente de cada sistema operativo
RAM	Studio: 8 GB. Studio y Emulador: 16 GB.	32 GB
CPU(Procesador)	Se requiere compatibilidad con la virtualización. Microarquitectura de CPU posterior a 2017.	Microarquitectura de CPU más reciente
Espacio en el disco	Studio: 8 GB de espacio libre. Studio y emulador: 16 GB de espacio libre	Unidad de estado sólido con 32 GB o más
GPU(Tarjeta gráfica)	Studio: ninguno Studio y emulador: GPU con 4 GB de VRAM, como Nvidia GeForce serie 10 o posterior, o AMD Radeon RX 5000 o posterior con los controladores más recientes.	GPU con 8 GB de VRAM, como Nvidia GeForce serie 20 o posterior, o AMD Radeon RX6600 o posterior con los controladores más recientes.

### *II-B. Fase 2: Configuración del Dispositivo de Prueba*

Para validar la metodología en un entorno controlado que simule las restricciones de hardware reales, se configuró un Dispositivo Virtual de Android (AVD) dentro de Android Studio. Se definió un perfil de televisión con resolución 1080p (1920x1080) y API 34. Un aspecto crítico de esta fase fue la limitación intencional de la memoria RAM a 2GB, con el objetivo de replicar las condiciones de estrés habituales en Smart TVs de gama media. La correcta configuración de este entorno, ilustrada en la Figura 2, es fundamental para detectar fugas de memoria o bloqueos en el hilo principal durante las pruebas de transición. Una vez inicializado el entorno, se ejecutó la validación de integridad mediante el despliegue de la aplicación base, confirmando la operatividad del sistema de renderizado gráfico en la Figura 3. Se configuró un Dispositivo Virtual de Android (AVD) con perfil de TV 1080p (1920x1080), API 34, y 2GB de RAM asignada para simular las limitaciones de hardware comunes en televisores de gama media tal como se aprecia en la ??.

### *II-C. Fase 3: Implementación de la Arquitectura*

La arquitectura base se fundamenta en... (PEGA AQUÍ LO DEL CÓDIGO BASE)...

### III. RESULTADOS

#### *III-A. Validación de la Propuesta*

Como resultado del proceso de investigación... (PEGA AQUÍ EL INICIO DE RESULTADOS)...

### IV. CONCLUSIONES

En conclusión, esta investigación logró establecer... (PEGA TUS CONCLUSIONES)...

The screenshot shows the Android Studio interface with the following details:

- Title Bar:** DemoApp > Version control
- Side Navigation Bar:** Shows the project structure under "Android":
  - app
  - manifests
  - kotlin.java
  - com.example.demoapp
  - ui.theme
  - MainActivity.kt
- Code Editor:** Displays the file `MainActivity.kt` with the following code:

```
package com.example.demoapp

import ...

class MainActivity : ComponentActivity() {
    @OptIn(markerClass = ExperimentalMaterial3Api::class)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            DemoAppTheme {
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    shape = RectangleShape
                ) {
                    Greeting(name = "Android")
                }
            }
        }
    }

    @Composable
    fun Greeting(name: String, modifier: Modifier = Modifier) {
        Text(
            text = "Hello $name!",
            modifier = modifier
        )
    }

    @Preview(showBackground = true)
    @Composable
    fun GreetingPreview() {
        DemoAppTheme {
            Greeting(name = "Android")
        }
    }
}
```
- Bottom Status Bar:** 45:28 LF UTF-8 4 spaces

Figura 1. Ventana principal de Android Studio

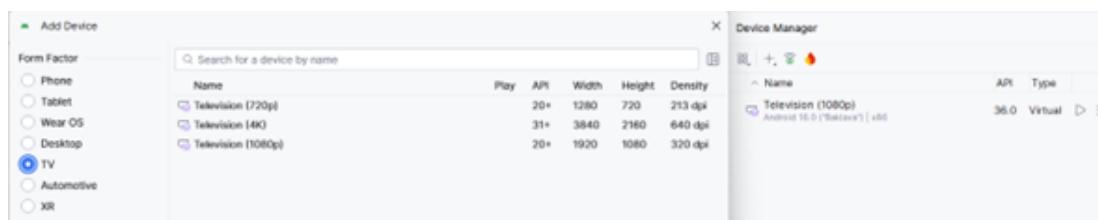


Figura 2. Configuración del emulador