Foto: Fernando Moura/Canva

Ecossistemas de CTVs e OTTs: Desafios técnicos e estratégias para compatibilidade multiplataforma

Por José Salustiano Fagundes de Souza

A revolução das TVs conectadas transformou o modo como consumimos conteúdo audiovisual. De aparelhos com vida útil de uma década, passamos para ecossistemas complexos de CTVs e OTTs que exigem compatibilidade entre múltiplas plataformas. A fragmentação de sistemas operacionais, linguagens e hardwares impõe desafios inéditos para desenvolvedores e empresas de streaming. Neste cenário, o autor analisa como estratégias de padronização, testes automatizados e uso inteligente de dado se tornam vitais e explica como plataformas líderes do setor como a Netflix e o Globoplay enfrentam esta nova era.

Houve uma época em que comprar um televisor era um investimento de longo prazo, sendo comum encontrar residências onde aparelhos de TVs permaneceram por quase uma década. Porém, isso tudo mudou com o processo de digitalização da televisão e, posteriormente, pela revolução que tornou a experiência televisiva baseada no acesso pela internet à aplicativos (***apps***) de ***software OTT*** (***Over-the-Top***), cada vez mais variados e complexos. Hoje as TVs são, por padrão, conectadas e inteligentes e para comprá-las tornou-se necessário analisar não apenas a qualidade do hardware, mas também a compatibilidade e a vida útil do ***software***.

Porém, ao contrário das plataformas móveis ou de ***desktop*** que possuem ambientes relativamente padronizados e estabilizados, as ***Smart TVs*** variam drasticamente por marca, região e ano de fabricação. Alguns modelos de entrada (mais baratos) podem até remover componentes-chave do sistema para economizar custos, o que afeta o comportamento dos ***apps*** de maneiras imprevisíveis. No geral, os sistemas operacionais (SOs) que vêm pré-instalados recebem atualizações por 4 anos, após esse período alguns ***apps*** podem apresentar problemas ou até mesmo pararem de funcionar.

Embora os termos ***Smart TVs*** e ***Connected TVs*** (CTVs) sejam frequentemente usados de forma intercambiável, é fundamental estabelecer uma distinção clara entre eles para um melhor entendimento. As ***Smart TVs*** são televisores com capacidade integrada de conectar-se à internet e acessar ***apps*** de streaming que vêm equipados com SOs distintos. Podemos considerar as ***Smart TVs*** como um subconjunto dos dispositivos de CTVs.

Por outro lado, o termo CTVs é mais utilizado para se referir a hardware externos, como ***sticks***, ***streaming boxes***, projetores de vídeos e ***consoles de videogame***, que habilitam a conectividade com a internet nos televisores, incluindo modelos mais antigos. ***Amazon***

Fire Stick, *Apple TV*, *Roku TV* e *Xbox 360* são exemplos de dispositivos que se conectam ao televisor - geralmente via HDMI - e fornecem uma plataforma que acessa conteúdo audiovisual pela internet. Enquanto as *Smart TVs* oferecem conveniência com sua natureza multifuncional, os dispositivos externos muitas vezes apresentam poder de processamento superior e podem modernizar televisores mais antigos sem a necessidade de uma substituição completa. Dessa forma, usuários podem optar por dispositivos externos para obter melhor desempenho, acessar ecossistemas específicos ou para evitar os ciclos de atualização frequentemente mais lentos, bem como a disponibilidade de *apps* limitadas para algumas plataformas de TV*s*.

Um mercado em crescimento: Diversidade, complexidade e fragmentação

Cada vez mais os consumidores dependem de CTVs para o entretenimento audiovisual e a flexibilidade, diversificação e natureza sob demanda dos serviços de *streaming* oferecidos podem ser considerados os principais atrativos que tornaram as TVs o habitat natural para *apps* de vídeos. Elas representaram aproximadamente 76,7% do tempo total da visualização global em 2022 de acordo com o *State of Streaming Reports* da *Conviva*. Em serviços *OTTs* como *Netflix* e *Disney*, estima-se que em 2025 as CTVs sejam responsáveis por mais de 50% da origem dos acessos nos EUA e em países da Europa, e de cerca de 35% a 40% em países da América Latina, onde existe ainda uma predominância de *Smartphones* devido a fatores como infraestrutura precária e alto custo.

Ao lado do México, o Brasil ocupa um lugar de destaque na América Latina: cerca de 6 em cada 10 usuários da internet no país acessaram a internet através de um televisor em 2023, segundo uma pesquisa elaborada pela IAB Brasil, um salto de 29% para 58% entre 2018 e 2023. Já um outro estudo realizado em novembro de 2024 pela *Samsung Ads Brasil* revelou que a região sudeste se destacava como a de maior consumo de conteúdo em TVs da empresa. O estudo mapeou os hábitos regionais de entretenimento e consumo com base em mais de 20 milhões de CTVs ativas no Brasil, alcançando mais de 81,3 milhões de potenciais consumidores.

Figura 1: O mercado brasileiro de CTVs possui uma grande quantidade e variedade de dispositivos. Foto: Foto de Oscar Nord na Unsplash

Porém, o crescimento e a evolução desses dispositivos trouxeram um cenário complexo para desenvolvedores, marcado pela diversidade de SOs - cada um deles com o seu próprio conjunto de ferramentas de desenvolvimento, diferentes processos de homologação de *apps*, linguagens de programação e diretrizes de interface - e por variações de hardware com chipsets (*SoC - System on a Chip*), processadores (*CPU*) que podem estar integrando ao *SoC* e *GPU* (Unidade de Processamento Gráfico) de tecnologias, fabricantes e desempenhos variáveis, principalmente se considerarmos as diferenças entre modelos de entrada (mais econômicos) e premium (mais caros e com alta performance).

Caminhando junto com o crescimento do número de dispositivos, existe uma batalha sendo travada neste momento entre grandes *players* de tecnologia que estão competindo para se tornarem o principal "*hub*" de conteúdo e influenciarem quais serviços os usuários devem priorizar. Afinal, por estar no dispositivo central das residências, ela é um ponto estratégico de controle sobre o usuário e o ecossistema de entretenimento, extraindo dados de uso e de comportamento (quais apps são mais usados, tempo gasto em cada conteúdo e horários de pico de uso para otimizar recomendações). É também capaz de mapear preferências de conteúdo (gêneros favoritos, histórico de buscas (sendo possível descobrir o que os usuários procuram em outros *apps* de *streaming*) e quais foram as interações com anúncios para o planejamento de uma publicidade mais direcionada); obter dados sobre dispositivos conectados (*Smartphones*, *Soundbars*, assistentes de voz como *Google Assistant*, *Bixby* e *Alexa*); e informações técnicas sobre o *hardware* que está sendo utilizado e a qualidade da conexão.

Em síntese, *Tizen*, *webOS* e *Google TV* não são apenas interfaces, são plataformas de negócios. Quem domina o SO das CTVs influencia o que você assiste, quais serviços prosperam e como as marcas podem monetizar a sua atenção.

Alguns dos principais SOs do mercado, com suas bases tecnológicas, linguagens suportadas, *Frameworks/SDKs* e principais dispositivos são listados na ***Tabela 1***.

SISTEMA OPERACIONAL	BASE TECNOLÓGICA	LINGUAGENS SUPORTADAS	FRAMEWORKS/ SDKS	DISPOSITIVOS PRINCIPAIS
Tizen (Samsung)	Kernel Linux, WebKitEFL, EFL	JavaScript, C#, HTML5, CSS	Tizen SDK, .NET/ Xamarin	TVs Samsung
webOS (LG)	Kernel Linux, WebKit, Enact	JavaScript, HTML5, CSS, C++	Enact Framework, LG SDK	TVs LG
Android TV	Kernel Linux, Android Runtime	Kotlin, Java, C++, HTML5	Android SDK, Jetpack, Leanback	Sony, Philips, Xiaomi
Google TV	Kernel Linux, Android Runtime	Kotlin, Java, C++, HTML5	Android SDK, Compose for TV	TCL, Chromecast
tvOS (Apple)	Kernel Darwin (Unix), UIKit, Metal	Swift, Objective-C, TVML	tvOS SDK, Xcode, SwiftUI	Apple TV
Roku OS	Kernel Linux, proprietário	BrightScript, SceneGraph (XML)	Roku SDK, SceneGraph	Roku Stick, TCL, Hisense
VIDAA	Kernel Linux, WebKit	HTML5, JavaScript, CSS	VIDAA Web App Framework	Hisense, Toshiba, JVC
Fire OS (Amazon)	Kernel Linux, baseado em Android	Java, Kotlin, C++, HTML5	Fire TV SDK, Android Studio, AVS	Fire TV Stick, Fire TV Cube

Tabela 1: Comparativo entre Sistemas Operacionais (2025). Fonte: Elaboração própria.

Componentes dos principais SOs apresentados

O ***Tizen*** é um SO de código aberto desenvolvido pela ***Linux Foundation*** em parceria com a Samsung. Começou a ser utilizado em modelos de TVs da empresa a partir de 2015. A sua base tecnológica inclui um motor de navegador ***open-source*** (***WebKitEFL***) que permite a renderização de conteúdo ***web*** e ***apps*** baseados em ***HTML5***; um conjunto de bibliotecas ***Enlightenment Foundation Libraries*** (EFL), que fornecem as funcionalidades gráficas e de interface do usuário, permitindo a criação de interfaces ricas e aceleradas por ***hardware***; o suporte ao desenvolvimento de ***apps*** usando ***C#*** através do ***Tizen .NET***, permitindo o uso do ***.NET Core*** e ***Xamarin.Forms***.

O uso ***HTML5***, ***CSS*** e ***JavaScript*** é amplamente suportado. A ***Samsung*** fornece um ***SDK*** (***Software Development Kit***) que inclui ***APIs JavaScript*** específicas para interagir com as funcionalidades da TV. Através do Tizen .NET, os desenvolvedores podem utilizar a linguagem ***C#*** e o ***framework .NET*** para criar ***apps*** nativos para a plataforma. Isso oferece acesso mais direto aos recursos do sistema e pode proporcionar uma significativa melhoria de desempenho.

O webOS é utilizado pela LG Electronics para as suas TVs e outros dispositivos de CTVs. A sua origem remonta ao Palm OS, um SO proprietário desenvolvido pela PalmSource para dispositivos móveis, incluindo PDAs e Smartphones. Em 2010, a Hewlett-Packard (HP) adquiriu a Palm e continuou o desenvolvimento. Em 2013, a LG adquiriu o webOS da HP e começou a aprimorá-lo para uso nas suas TVs no ano seguinte. A sua base tecnológica inclui um Kernel Linux, assim como o Android e o Tizen; um motor de navegador WebKit para renderizar o conteúdo web e os apps baseados em HTML5; o uso do Enact Framework, baseado em React e otimizado para o webOS; o Node.js para alguns de seus serviços de backend e o Luna Bus como sistema de barramento de mensagens interno, permitindo a comunicação entre diferentes componentes e serviços do SO.

As principais opções de linguagens para o desenvolvimento de ***apps*** são o ***HTML5***, ***CSS*** e ***JavaScript***, onde existem ***APIs*** específicas fornecidas pela LG; e o ***C/C++*** para ***apps*** nativos que exigem acesso de baixo nível ao hardware ou um desempenho gráfico mais otimizado. A versão mais recente do ***webOS*** é o 25.

O ***Android TV*** e ***Google TV***: o ***Android TV*** foi anunciado pelo ***Google*** em 2014 e os primeiros dispositivos com o sistema começaram a ser lançados em marcas de modelos da Sony e da Philips. Ele é construído sobre o ***Kernel Linux*** e compartilha muitas das mesmas características e componentes do ***Android*** para Smartphones e Tablets, mas com otimizações para a experiência em telas grandes e com uso de controle remoto. Seus principais componentes incluem o ***Android Runtime***, um ambiente de execução das aplicações; uma coleção de bibliotecas e ferramentas (***framework***) para a criação de ***apps***; uma interface de usuário projetada especificamente para TVs (***Leanback***), com foco em navegação simples com controle remoto.

O ***Google TV*** foi lançado em 2020 como uma evolução do ***Android TV***. Ele está disponível em dispositivos como

Chromecast e TVs da TCL e Sony, possui uma interface mais personalizada, com recomendações baseadas nos seus hábitos de visualização e combina conteúdo de vários serviços OTTs (YouTube, Netflix, HBO Max etc.) em uma única tela. Para o desenvolvimento de *apps* - embora o *Google TV* tenha novas *APIs* e recomendações de UI/UX e possibilite o uso do *Compose for TV* substituindo parcialmente o *Leanback* em aplicações que tenham exigências mais rígidas em termos de desempenho e integração de conteúdo - ele utiliza basicamente as mesmas tecnologias para o desenvolvimento *Android*: o *Java* e o *Kotlin*; o *Android SDK* e o *Android Studio* para criar *apps* otimizados para a interface *Leanback* do *Android TV*; o *Native Development Kit (NDK) (C/C++)*, para *apps* que exigem alto desempenho ou acesso de baixo nível ao *hardware*, utilizando *C* ou *C++*; e *Web Technologies* (*HTML5, CSS, JavaScript*) através de *WebView* ou de *frameworks* específicos, que encapsulam conteúdo web em um aplicativo *Android* nativo. Permite ainda integração com o *Google Assistant* para controle por voz e outras funcionalidades inteligentes e a transmissão de conteúdo de smartphones, tablets e computadores diretamente para a TV (*Chromecast built-in*).

O *FireOS*, atualmente na versão 8, foi desenvolvido pela Amazon para os seus dispositivos. Ele surgiu em 2014 (inicialmente projetado para uso pelo *Fire Phone*) e foi adaptado nos anos seguintes para uso no *Fire TV Stick*, *Fire TV Cube*, *Fire Tablets* e *Echo Show* (dispositivos com tela da linha Alexa).

Embora tenha como base um *Kernel Linux* que é o mesmo do Android, ele não é *Open Source* e teve várias modificações para priorizar o ecossistema da *Amazon*: serviços *Google* (*Play Store* e *GMS*) foram substituídos pelo Amazon Appstore e AWS; possui uma interface personalizada (*Amazon Silk Browser* e Alexa integrada) e foi otimizado para conteúdo Amazon (Prime Video, Music e Kindle).

O desenvolvimento de apps no FireOS é muito similar ao do Android TV, mas com algumas diferenças: possui extensões para controles remotos e voz (Fire TV SDK); permite o uso do Android Studio, mas com SDK da Amazon); e suporta Web Apps (HTML5/JS), porém de forma limitada. Recentemente, houve um aumento no suporte ao desenvolvimento de apps usando React Native para o FireOS. Algumas bibliotecas estão sendo desenvolvidas para facilitar a integração do React com APIs específicas do FireOS e existem comunidades e fóruns na internet que discutem e compartilham experiências sobre esse tema.

A Apple utiliza o *tvOS*, atualmente na versão 18, que é baseado no *iOS*. As primeiras gerações da Apple TV (anteriores à 4ª geração) utilizavam uma versão modificada do *macOS X* e, posteriormente, versões simplificadas do *iOS*. Com o lançamento da quarta geração da Apple TV, no final de 2015, o *tvOS* foi utilizado como um SO otimizado para a visualização em tela grande e interação com o controle *Siri Remote*.

O *tvOS* compartilha muitas das mesmas bases tecnológicas do *iOS*, tem um *Kernel* baseado em *Unix* (o *Darwin*) no seu núcleo; *Frameworks* e *APIs Cocoa Touch*, uma interface usada para construir apps que inclui *frameworks* como *UIKit* (adaptado para a tela de TV e controle remoto); uma *API* de gráficos de baixo nível (*Metal*) que oferece alto desempenho para renderização 3D e 2D; um motor de renderização web (*WebKit*) utilizado pelo navegador *Safari* e por visualizações web dentro de apps; o *AirPlay* para *streaming* sem fio de áudio e vídeo; e um *HomeKit* para controlar dispositivos domésticos inteligentes. Para o desenvolvimento são utilizadas as linguagens *Swift*, que é a principal recomendação da Apple para novos projetos; o *Objective-C* (embora o *Swift* seja preferível para novos apps, ainda há uma base expressiva de código em *Objective-C*); e o *Xcode*, o ambiente de desenvolvimento integrado (*IDE*) da *Apple*, juntamente com o *tvOS SDK* - que oferece ferramentas, documentação e bibliotecas necessárias para aproveitar os recursos específicos do tvOS - para criar, testar e depurar apps. Além disso, *frameworks* adaptados para o *tvOS*, como *SwiftUI* e *UIKit* são usados para construir a interface do usuário.

O *Roku OS* (atualmente na versão 14.5) é o SO desenvolvido pela *Roku* especificamente para seus dispositivos de *streaming*, mas que está presente também em CTVs de outros fabricantes como a TCL, AOC, Hisense e Sharp. O primeiro media player digital da empresa, o HD1000, que foi lançado em 2004, já utilizava o Roku OS. Também é baseado no *Kernel Linux* e tem uma arquitetura projetada para ser eficiente e rodar em hardwares com baixo consumo de energia e memória limitada, comuns em dispositivos de CTVs de entrada.

O desenvolvimento de "canais" para a plataforma envolve principalmente duas linguagens de programação: o *BrightScript*, que é proprietária da *Roku* e é usada para definir o comportamento e a lógica dos apps, combinando elementos de linguagens de script (*Python, BASIC, Ruby* e *Lua*), suportando tipagem dinâmica (onde o tipo de uma variável pode mudar durante a execução) e a possibilidade de declaração de tipos, como em *C* e *Java*; e o *SceneGraph*: e um *framework XML* orientado a objetos, também proprietário da *Roku*, que é utilizado para projetar a UI dos apps. Essas duas linguagens trabalham juntas de forma semelhante com o *HTML* e o *JavaScript*.

O *VIDAA*, desenvolvido pela *VIDAA USA* (uma subsidiária da *Hisense*) e lançado em 2014, foi projetado como uma plataforma integrada para acessar conteúdo multimídia, incluindo serviços de streaming, TV ao vivo e apps em TVs. Além da Hisense ele é utilizado por outros fabricantes de televisores como a Toshiba, Vestel (que possui marcas como Daewoo, Regal, Hitachi, Telefunken e JVC) e a Loewe. A principal linguagem utilizada para o desenvolvimento é o HTML5 e tecnologias web relacionadas como *JavaScript* e *CSS*. A plataforma fornece um *framework* para *web apps* e possui *APIs* proprietárias que permitem a interação com funcionalidades específicas do SO e do hardware das TVs.

Desenvolver APPS WEB, nativo ou híbrido?

Para essa pergunta não existe uma resposta única "certa". A melhor escolha dependerá dos objetivos específicos do projeto. A decisão de utilizar linguagens web, nativa ou híbrida (***web+nativa***) no desenvolvimento deve levar em consideração diversos fatores estratégicos, técnicos e de negócio.

O uso de linguagens web é quase sempre mais eficiente para se alcançar com um único código-fonte (ou com pequenas adaptações) o maior número possível de usuários em diferentes SOs de CTVs. Elas oferecem ciclos de desenvolvimento mais rápidos e atualizações mais fáceis, já que, em tese, uma única atualização pode ser implantada para todas as plataformas simultaneamente, simplificando a manutenção e a correção de bugs. A sua utilização tem sido suficiente para a grande maioria dos apps de ***streaming*** e conteúdo que são predominantemente ***I/O-bound*** (Dispositivos para entrada e saída de dados). O desempenho das ***webviews*** (componente que permite incorporar conteúdo web em aplicações nativas) em TVs é robusto o suficiente para uma boa experiência do usuário. Com as ferramentas e ***frameworks*** certos, é possível criar experiências visuais muito atraentes e fluidas. O acesso a ***APIs*** e funcionalidades nativas é intermediado pelas camadas ***web*** dos ***SDKs***.

Em alguns casos, se o foco é uma plataforma específica onde o desempenho máximo é um diferencial competitivo (ex.: atender a uma base de dispositivos mais antigos ou com limitações de desempenho), ou se há uma necessidade de funcionalidades muito específicas de hardware de uma única plataforma e do processamento de vídeo em tempo real com ***codecs*** otimizados, o desenvolvimento nativo deve ser considerado.

Outro diferencial é o fato delas permitirem acesso irrestrito a todas as ***APIs*** e funcionalidades do sistema operacional e hardware da TV - essencial para requisitos muito específicos de integração com o dispositivo - e a criação de ***UIs*** com a máxima fluidez e responsividade, especialmente em transições e animações complexas, pois têm controle direto sobre o processo de renderização. Porém, elas possuem um processo de atualização mais complexo, pois cada versão nativa (por plataforma) precisa ser compilada, testada e submetida individualmente às lojas de aplicativos, o que pode prolongar o tempo de lançamento de novas funcionalidades ou correções. Exigem também a alocação no projeto de desenvolvedores com conhecimento específico das linguagens (***C/C++***) e dos ***SDKs*** nativos de cada fabricante, que são mais nichados e difíceis de encontrar.

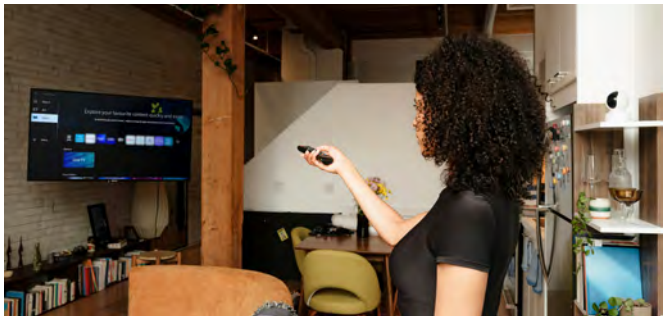

Figura 2: O desafio de escolher a melhor abordagem para o desenvolvimento de apps em CTVs. Foto de Samsung Memory na Unsplash

A ***Tabela 2*** apresenta linguagens nativas e frameworks nos principais SOs de CTVs:

SISTEMA OPERACIONAL	LINGUAGENS NATIVAS	FRAMEWORKS & SDKS
Tizen (Samsung)	C, C# (.NET via Tizen Native)	Tizen Native SDK, .NET for Tizen, Tizen Web SDK
webOS (LG)	C, C++ (via Native SDK)	webOS Native SDK, Enact Framework, webOS Web SDK, PDK (Plug-in Development Kit) para componentes nativos de C/C++
Android TV	Kotlin, Java, C++	Android TV SDK, NDK Leanback, Compose for TV
Google TV	Kotlin, Java, C++	Android TV SDK, Jetpack, Google Cast SDK, Compose for TV, Google TV Experience APIs
tvOS (Apple)	Kernel Darwin (Unix), UIKit, Metal	tvOS SDK, Xcode, UIKit, SwiftUI, AVFoundation, SpriteKit/SceneKit
Roku OS	Kernel Linux, proprietário	Roku SDK, SceneGraph, BrightScript Engine
Fire OS (Amazon)	Kotlin, Java, C++	Amazon Fire TV SDK, Android SDK, Fire App Builder, Web App Starter Kit.

Tabela 2: Linguagens Nativas & SDKs para CTVs. Fonte: Elaboração própria.

O orçamento e os recursos disponíveis são aspectos importantes a serem considerados. Linguagens web, que são mais acessíveis e rápidas, possuem uma vasta disponibilidade de desenvolvedores formados e permitem a reutilização de código entre plataformas, resultam em custos de desenvolvimento e manutenção significativamente menores. Já as linguagens nativas exigem equipes especializadas para cada plataforma, aumentando o tempo e o custo de desenvolvimento, além de um esforço de manutenção muito maior para múltiplos códigos-fonte.

Desenvolver de forma híbrida (combinando tecnologias web e nativas) para CTVs, embora não seja a abordagem mais comum ou diretamente suportada por todos os fabricantes de forma padronizada, pode oferecer um conjunto de vantagens. Essa abordagem permite a facilidade da reutilização do código, economizando tempo e custo e uma manutenção mais simplificada. Embora a interface principal seja web, o "***container***" nativo permite que o aplicativo acesse funcionalidades de hardware ou ***APIs*** específicas da TV que talvez não estejam disponíveis diretamente para aplicativos puramente web. Como parte da aplicação que exige alta performance pode ser implementada nativamente e integrada com a interface web, é possível encontrar um equilíbrio entre agilidade de desenvolvimento e desempenho.

Na prática para TVs, a abordagem mais comum que se aproxima do "híbrido" é o uso de aplicações "***Hosted Web"*** com componentes nativos integrados. Em casos mais avançados, um aplicativo pode ter uma base nativa que gerencia aspectos críticos (como um player de vídeo personalizado ou acesso a ***tuners*** de TV) e uma interface de usuário construída com tecnologias web que se comunicam com esses componentes nativos. Essa é a verdadeira essência do híbrido para TVs.

Para os próximos anos é importante ressaltar que, com a implantação da TV 3.0 no Brasil e a evolução do seu ***middleware*** (DTVI+), uma nova camada de desafios será adicionada para os desenvolvedores de CTVs que precisarão conhecer tecnologias híbridas e complementares para a criação de apps. A formação de recursos para trabalhar nesse cenário, que possui um ecossistema conceitualmente fechado, será um grande desafio para empresas que planejam fazer negócios com essa nova tecnologia.

Os desafios da fragmentação

Toda essa fragmentação de SO, linguagens e ***SDKs,*** somadas a presença de hardware com ***chipsets*** de baixo desempenho em modelos mais antigos (que gera falhas), dificulta a criação e a manutenção de ***software*** e exige que sejam feitas múltiplas versões de um mesmo ***app***, aumentando custos e tempo de desenvolvimento.

Segundo Abreu et al. (2022), o cenário de fragmentação do ecossistema televisivo demanda soluções tecnológicas e de negócios que unifiquem a experiência do espectador, integrando diferentes fontes de conteúdo em uma única interface. Os autores destacam quatro principais divisões, percebidas pela indústria e pela academia há alguns anos, como elementos disruptivos que impactam na experiência do usuário e nos modelos de negócios. São elas:

- A queda no consumo de canais lineares e um aumento significativo no conteúdo sob demanda, incluindo serviços de ***VOD*** e plataformas ***OTT***;
- O fato de as operadoras de TV por assinaturas tradicionais oferecerem pacotes combinados (TV, internet e telefone), enquanto os serviços OTT fornecem conteúdo de forma desagregada, com modelos de assinatura mais flexíveis permitindo a escolha da rede de acesso;
- A seleção de conteúdo por curadores humanos contrastando com a recomendação algorítmica;
- A necessidade de considerar o consumo de conteúdo audiovisual para além do da TV, incluindo smartphones, tablets e computadores, dispositivos onde a experiência televisiva também acontece.

Empresas como Netflix, Disney+, HBO Max e Globoplay investem continuamente em soluções para garantir uma compatibilidade e desempenho consistente de seus *apps*. **Eles já entenderam que uma abordagem única para todos os CTVs simplesmente não funciona.** A empresa que deseja oferecer uma experiência ***OTT*** fluida e de alta qualidade precisa ter uma estratégia de controle de qualidade muito bem estruturada, fazer ajustes finos e específicos para cada plataforma e um monitoramento contínuo do desempenho. E o trabalho precisa começar por um bom projeto de ***design***, criando uma ***interface*** de usuário de fácil navegação e responsiva, que funcione bem em diversas telas.

Garantir uma experiência uniforme em diferentes plataformas é um desafio e as grandes empresas de ***streaming*** têm investido em tecnologias ***Cross-Platform***, otimização de desempenho e testes automatizados para garantir uma experiência consistente. A adoção de ***PWAs*** (***Progressive Web Apps***), de ***frameworks*** unificados e de ***ABR*** (***Adaptive Bitrate Streaming***) são tendências que podem reduzir a complexidade no ecossistema de TVs conectadas. A HBO Max adotou uma arquitetura modular para reduzir retrabalho e a Netflix, Prime Video e Globoplay utilizam componentes baseados em React Native para TV permitindo uma maior reutilização de código.

O case da Netflix: inovação e open-source

Para entender como os líderes do setor operam com seu ambiente de desenvolvimento, investigamos artigos sobre o case de sucesso da Netflix. A empresa é uma referência não apenas pelo seu pioneirismo - que redefiniu a experiência televisiva na era da convergência digital e mostrou caminhos a serem seguidos - mas também pela qualidade de entrega dos seus serviços. Afinal, quase duas décadas de investimentos em streaming permitiram que a Netflix se destacasse por ter uma interface de usuário bem resolvida; usar algoritmos sofisticados para recomendação de conteúdo com base no histórico de visualização e preferências dos usuários; oferecer uma qualidade no suporte a streaming em 4K e HDR, usando tecnologias que minimizam o buffering, mesmo em conexões lentas; e por ter uma prática de atualizações frequentes da sua plataforma, introduzindo novas funcionalidades e melhorando a UX para aumentar o engajamento.

A Netflix utiliza diferentes abordagens para o desenvolvimento, dependendo da plataforma. No Android TV/Google TV usa o Kotlin e o Java (para versões mais antigas), Jetpack Compose (UI) e Netflix Android Client (algumas libs são Open-Source e estão disponíveis no GitHub); para o Tizen e webOS utiliza JavaScript/HTML5, React.js e Enact Framework (otimizado para o webOS); no tvOS o Swift e TVML (***TV Markup Language***) para apps baseados em templates; e o BrightScript é utilizado para o Roku. A empresa contribui com as comunidades open-source disponibilizando várias bibliotecas, como a Zuul, um sistema de API Gateway desenvolvido pela Netflix que atua como um ponto central para todas as solicitações de API que chegam ao sistema, antes de serem encaminhadas para os microserviços internos; Hystrix, uma biblioteca de código aberto que ajuda a lidar com latência e falhas em sistemas distribuídos. Foi projetada para isolar pontos de acesso em sistemas remotos, serviços e bibliotecas de terceiros, evitando falhas em cascata e habilitando a resiliência em sistemas distribuídos onde a fala é inevitável. Metaflow, uma ferramenta de orquestração que tem código aberto desde 2019. Ela foi originalmente desenvolvida para atender às necessidades de desenvolvedores e cientistas de dados que trabalham em projetos exigentes de Machine Learning, IA e dados na Netflix. A solução trata de todo o fluxo de trabalho de ciência de dados, do protótipo à implantação do modelo, e fornece integrações internas aos serviços de nuvem da AWS; e o Polly.js, uma biblioteca JavaScript que permite gravar, reproduzir e simular interações HTTP de chamadas de API de uma aplicação sem a necessidade de um servidor real.

Um conjunto de soluções ***open-source*** usadas pela Netflix podem ser encontradas no *Netflix Open Source Software Center*.

Entre as melhores práticas se destacam o uso de:

- *Dynamic Optimizer,* um *framework* de otimização de codificação de vídeo desenvolvido pela Netflix. Seu objetivo é melhorar a qualidade do vídeo percebida pelo espectador, reduzindo a taxa de bits necessária para manter essa qualidade. Ele utiliza uma abordagem baseada em percepções humanas, considerando como os espectadores reagem a diferentes qualidades de vídeo. Se a conexão piora, o player automaticamente reduz a resolução gradualmente, aumenta o buffer mínimo e prioriza áudio contínuo O seu algoritmo utiliza:
- *Buffer-based Adaptation*, para monitorar o ***buffer*** e evitar interrupções;
- *Throughput-based Adaptation*, faz ajustes com base na velocidade real da rede;
- *Hybrid Approach*, que combina ambos para decisões mais precisas.
- Ferramentas internas para testes A/B em larga escala:
- *A/B Testing Framework*, para a criação, execução e análise de testes de maneira eficiente suportando múltiplos testes simultaneamente;
- *Data Pipeline*, uma infraestrutura que coleta e processa dados em tempo real, permitindo que a equipe de desenvolvimento analise rapidamente os resultados dos testes;
- *Metric Tracking*, para monitorar e medir as métricas de desempenho relevantes durante os testes. Isso inclui dados quantitativos (como visualizações e cliques) e qualitativos (feedback dos usuários); e
- *Statistical Analysis Tools*, que ajudam na análise estatística dos resultados dos testes, garantindo que as conclusões sejam baseadas em dados confiáveis.
- Arquitetura de microsserviços: A Netflix adota uma arquitetura onde a aplicação é estruturada como uma coleção de serviços menores e independentes. Isso permite escalabilidade, flexibilidade e implantações independentes.
- *Cloud Computing*: A Netflix foi pioneira no uso extensivo da AWS para sua infraestrutura de nuvem. Eles utilizam vários serviços da AWS para diversos serviços, como os de armazenamento, computação e entrega de conteúdo.
- Player personalizado: o player de vídeo da Netflix é um dos componentes mais sofisticados de sua plataforma, desenvolvido para oferecer a melhor experiência possível em milhares de dispositivos diferentes. Ele é construído como um sistema modular que se adapta dinamicamente às capacidades do dispositivo e condições de rede. Seus principais componentes são:

- Camadas de *Player*, com *UI Layer* (controles de *playback* personalizáveis por dispositivo e elementos interativos como menus *X-Ray*, que mostra metadados em tempo real, cenas com atores específicos e trivia sobre a produção);
- *Business Logic Layer*, com gerenciamento de DRM e lógica de negócios (controle parental, contagem de tempo assistido);
- *Playback Engine*, com decodificação de vídeo/áudio e *Adaptive Bitrate*;
- *Network Layer*, com protocolos padrão de streaming (HLS, DASH) e conexão com a *CDN Open Connect*.
- Uso de DRM padrão de mercado (*Widevine*, *FairPlay* e *PlayReady*) e do *Netflix MSL*, um protocolo proprietário para comunicação segura.
- Uso de otimizações específicas por dispositivos: *Player* baseado em *JavaScript* com *hardware acceleration* para o *Tizen*; uso de *APIs* nativas de decodificação para o *webOS*; *AVPlayer* com extensões customizadas para o *iOS*; e *ExoPlayer* modificado com *Prefetching* inteligente e decodificação por hardware prioritária.

Inovações exclusivas como o uso de:

- *Perceptual Quality Optimization* para análise em tempo real de artefatos de compressão, fluidez de movimento e detalhes em cenas escuras;
- Ferramentas de *downloads offline* para criptografia por dispositivo, gerenciamento de espaço inteligente e atualizações automáticas de episódios.

Alguns aspectos de performance podem ser facilmente percebidos na experiência dos usuários. **Licciardello et al.** (2020) coletaram dados em um artigo sobre um desses aspectos, o comportamento da interação dos *players* de vídeo em resposta a variações controladas na largura de banda. Foram testados vários vídeos de serviços em diferentes condições de rede que demonstraram:

- uma ampla gama de comportamentos em relação à manutenção de *buffer*, adaptação à largura de banda e eficiência no uso da largura de banda disponível pelo *Adaptive Bitrate*;
- que muitos algoritmos priorizam a estabilidade em vez de mudanças rápidas na qualidade do vídeo, sugerindo uma preferência por uma experiência de visualização mais suave, em vez de mudanças frequentes de resolução;
- que algumas plataformas, como o Youtube, utilizavam menos de 60% da largura de banda disponível, levantando questões sobre a eficiência dos algoritmos.

Ao encontrarmos alguns dados de testes obtidos com a Netflix, como um *Startup Time* menor que 1.5 segundos em até 90% dos dispositivos; *Rebuffering Rate* menor que 0.5%, quando a média do mercado é de 1 a 2%; e um *Bitrate Switching* que faz ajustes a cada 2 a 10 segundos, enquanto outros *players* levam de 15 a 30 segundos, entendemos o motivo de existir um certo senso comum de que existe uma superioridade da plataforma da Netflix em relação a muitos de seus concorrentes.

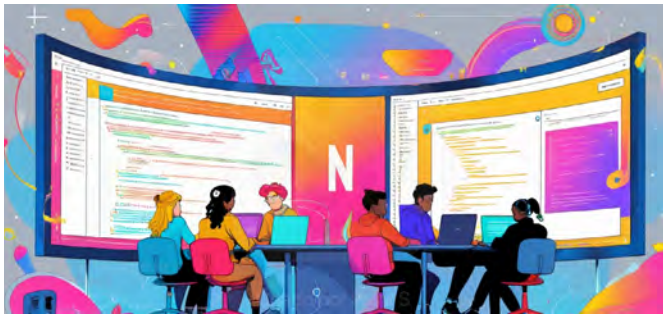

Figura 3: Desenvolvimento da Netflix é referência e contribui com a comunidade open-source/Foto: Autor

Panorama nacional

Olhando para o mercado brasileiro, temos um *gap* trazido pela implantação da TV digital que fez com que muitas emissoras criassem uma barreira com relação ao uso da internet nos televisores. Durante mais de 60 anos as transmissões do sinal de TV foram um território de domínio exclusivo de empresas incumbentes que tiveram origens nas áreas de rádio e jornalismo.

No Brasil, assim como aconteceu em outros países, a transformação disruptiva que foi trazida pela internet para o setor de radiodifusão foi vista com resistência e desconfiança, principalmente por tirar da zona de conforto empresas que não tinham cultura de inovação, uma mentalidade que infelizmente prevaleceu por muitos anos e impediu que novas formas de se fazer e assistir televisão fossem experimentadas. Somente mais recentemente estamos assistindo a um movimento maior em direção ao *streaming* por parte de emissoras e canais, principalmente devido ao crescimento dos *Fast Channels*. Mas ainda existe um longo caminho a ser trilhado e a maioria dos serviços se esforçam para manter o equilíbrio entre a necessidade de investimentos constantes em infraestrutura e recursos tecnológicos e a viabilidade econômica dos serviços.

Ao olharmos do ponto de vista de maturidade e resiliência, dois serviços OTTs 100% brasileiros se destacam: o *Globoplay* e o *Looke*, ambos criados em

2015. O primeiro está presente nos principais modelos de dispositivos de CTVs, foi criado e é mantido por uma equipe interna responsável pelo desenvolvimento, que atuam em três frentes de batalha: a) ***Web/Backend***, que utiliza ***Javascript***, ***React***, ***NodeJS*** e o ***Jarvis***, uma ***GraphQL API Gateway*** que é a base de todo ***backend*** do serviço; b) ***Mobile*** (incluindo o ***Android TV*** e o ***tvOS***), utilizando as linguagens nativas ***Kotlin*** e ***Swift***; e c) CTVs, abrangendo mais de 15 plataformas diferentes. Devido à sua complexidade, nesse tipo de desenvolvimento, conforme vimos anteriormente, são utilizados formatos de apps diferentes, como ***HTML***, Nativo e *Roku (BrightScript)*.

O app do ***Looke*** oferece um catálogo com mais de 10.000 conteúdos de diferentes gêneros e países, incluindo títulos originais e exclusivos, está presente no México e pode ser assinado através das plataformas da Amazon Prime Video, Claro TV+ e Vivo Play. Foi desenvolvido e é mantido por uma equipe interna que utilizam hoje o mesmo conjunto de linguagens, ***frameworks*** e ferramentas compatíveis com as que foram citadas nesse artigo. Ele pode ser acessado nos principais modelos de CTVs a partir de 2013, em dispositivos móveis, web e pelo console de videogame ***xBox***.

Foto de Kevin Woblick na Unsplash

Rumo às boas práticas

É possível elencar algumas boas práticas comuns para mitigar os principais desafios mencionados neste artigo. São elas:

- **Design e Experiência do Usuário (*UX/UI*)**
  - **Design responsivo e minimalista**: adotar interfaces com alto contraste, fontes legíveis a distâncias variadas, fundos escuros para melhorar a legibilidade em ambientes com pouca luz e navegação simplificada.
  - **Design adaptativo**: usar interfaces que se adaptem à resolução, memória e capacidade de processamento dos dispositivos, especialmente para modelos de entrada de marcas populares na América Latina.
  - **Estratégia de *fallback* para dispositivos *legacy***: suporte limitado a dispositivos antigos com alternativa leve via *web/PWA*.
- **Desempenho e Eficiência**
  - **Otimização de desempenho:** uso eficiente de recursos (memória e processamento, imagens, animações e chamadas de rede para garantir uma experiência fluida).
  - **Uso de tecnologias de distribuição de conteúdo:** implementar *CDNs (Content Delivery Networks)* e balanceadores de carga para fazer uma entrega eficiente de vídeos, reduzindo a latência e melhorando a qualidade do *streaming*, especialmente durante picos de acesso.
  - **Adotar *frameworks* multiplataforma com *fallback* inteligente:** evitar duplicação de código mantendo desempenho nativo.
- **Testes e Qualidade**
  - **Testes em diversos dispositivos:** realizar testes em hardware real e laboratórios com uma ampla gama de dispositivos representativos da base de usuários de CTVs para identificar e corrigir problemas específicos de hardware ou SO, garantindo maior compatibilidade e desempenho consistente.
  - ***CI/CD com testes automatizados específicos para CTVs:*** criar pipelines de teste que incluam automação e ferramentas para diferentes resoluções e comportamentos.
- **Monitoramento, Observabilidade e Dados**
  - **Telemetria:** fazer uma coleta contínua de dados sobre o uso das aplicações para identificar padrões de comportamento, detectar problemas e implementar melhorias na experiência do usuário.
  - **Observabilidade orientada por métricas de *UX***: uso de dashboards com *TTFF* (*Time To First Frame*), *Rebuffering Ratio* e engajamento por dispositivo, para decisões baseadas em dados.
  - **Estratégia de *fallback* para dispositivos *legacy***: definir limites claros para suporte a modelos com mais de 5 anos e oferecer versão web ou *PWA* (*Progressive Web App*) adaptada como alternativa leve.
- **Desenvolvimento e Arquitetura**
  - **Adotar *frameworks* multiplataforma com *fallback* inteligente**: recomenda-se o uso de abstrações baseadas em *React Native for TV*, que permitem desempenho nativo sem duplicar código excessivamente.

Considerações finais

O desenvolvimento para CTVs e dispositivos de streaming representa hoje um dos maiores desafios técnicos e de negócio para plataformas OTT. A fragmentação de sistemas operacionais, a diversidade de hardwares e a necessidade de manter uma experiência fluida e responsiva em centenas de dispositivos exigem investimentos contínuos em arquitetura modular, testes automatizados, telemetria e suporte multicanal.

Empresas líderes como a Netflix demonstram que uma abordagem centrada na padronização de componentes, uso intensivo de dados e investimentos em tecnologias como ***Adaptive Bitrate Streaming*** e ***Dynamic Optimizer*** são fundamentais para escalar com qualidade.

No cenário brasileiro, serviços como Globoplay e Looke mostram que é possível alcançar boa cobertura e performance, mas a complexidade do ecossistema ainda impõe barreiras técnicas significativas. O futuro do desenvolvimento OTT nas CTVs depende da colaboração entre fabricantes, provedores de conteúdo e desenvolvedores para construção de um ecossistema mais interoperável, com APIs mais abertas e suporte a tecnologias como ***PWAs*** e ***React Native for TV***.

A realização de acordos e parcerias é fundamental para se movimentar nesse novo ecossistema de televisão, mídia e entretenimento que foi formado. Todos podem ser, ao mesmo tempo, concorrentes e parceiros comerciais. Provedores de serviços de ***OTT video streaming*** precisam do relacionamento com fabricantes e detentores de SOs para manterem e evoluírem as suas plataformas; por sua vez, fabricantes precisam ter ofertas de apps de conteúdos (não só em quantidade, mas também com qualidade) que possam fazer a diferença na hora de um consumidor escolher um dispositivo de CTV. Ao estudarmos cases do setor fica muito claro que não existe sucesso sem a realização de alianças estratégicas.

Para os próximos anos, recomenda-se ainda que empresas que têm no seu ***core*** de negócios a oferta de serviços OTT estabeleçam critérios claros de suporte por dispositivo; adotem arquitetura modular e ***frameworks*** reutilizáveis; invistam em monitoramento de dados e testes automatizados - ter um laboratório com os principais modelos de CTVs utilizados na base é recomendável – e, sobretudo, que estejam abertas às inovações que podem, quando percebidas, mensuradas e utilizadas no tempo certo, agregar valor aos seus negócios.

Referências

1. CONVIVA. State of Streaming Reports, 2022. Disponível em: https://www.conviva.com/research/. Acesso em: 14 maio 2025.
2. IAB BRASIL. Pesquisa sobre Comportamento de Consumo em Connected TVs. São Paulo, 2023. Disponível em: https://iabbrasil.com.br. Acesso em: 14 maio 2025.
3. SAMSUNG ADS BRASIL. Estudo sobre Hábitos Regionais de Consumo de Conteúdo em CTVs, 2024. Disponível em: https://www.samsungads.com/latam/pt. Acesso em: 14 maio 2025.
4. ABREU, J. A. et al. Fragmentação no Ecossistema de Televisão Conectada: desafios e oportunidades. Revista Brasileira de Mídia Digital, v. 10, n. 2, p. 45–67, 2022.
5. WIKIPEDIA. I/O Bound. Disponível em: https://pt.wikipedia.org/wiki/I/O_bound. Acesso em: 15 maio 2025.
6. NETFLIX TECHNOLOGY BLOG. Engineering and Tech Blog. Los Gatos: Netflix. Disponível em: https://netflixtechblog.com/. Acesso em: 17 maio 2025.
7. NETFLIX. Zuul: Edge Service in the Cloud. Netflix Tech Blog, 2013. Disponível em: https://netflixtechblog.com/zuul-edge-service-in-the-cloud-ab3af5c31e54. Acesso em: 17 maio 2025.
8. NETFLIX. Hystrix: Latency and Fault Tolerance for Distributed Systems. Netflix Tech Blog, 2012. Disponível em: https://netflixtechblog.com/hystrix. Acesso em: 17 maio 2025.
9. NETFLIX. Introducing Metaflow: Human-Centric Framework for Data Science. Netflix Tech Blog, 2019. Disponível em: https://netflixtechblog.com/metaflow. Acesso em: 18 maio 2025.
10. POLLY.JS. Record, replay, and stub HTTP interactions. GitHub, 2023. Disponível em: https://github.com/Netflix/pollyjs. Acesso em: 18 maio 2025.
11. NETFLIX. Open-source Software Center. Disponível em: https://netflix.github.io/. Acesso em: 18 maio 2025.
12. BYTEPLUS. Does Netflix use adaptive bitrate streaming? A deep dive into streaming technology. Disponível em: https://www.byteplus.com/en/topic/122703?title=does-netflix-use-adaptive-bitrate-streaming-a-deep-dive-into-streaming-technology. Acesso em: 19 maio 2025.
13. LICCIARDELLO, Melissa; GRUNER, Maximilian; SINGLA, Ankit. Understanding video streaming algorithms in the wild. 2020. Disponível em: https://arxiv.org/pdf/2001.02951. Acesso em: 22 mai. 2025.
14. GLOBOPLAY. Tecnologia e arquitetura do app Globoplay. Disponível em: https://globoplay.globo.com. Acesso em: 19 maio 2025.
15. ---. Os bastidores tech do Globoplay. 2021. Disponível em: https://globoplay-tecnologia.medium.com/os-bastidores-tech-do-globoplay-8cf5e39dcf38. Acesso em: 20 maio 2025.
16. LOOKE. Plataforma brasileira de streaming. Looke, 2025. Disponível em: https://www.looke.com.br. Acesso em: 20 maio 2025.

José Salustiano Fagundes de Souza

possui duas décadas de experiência com a criação e o desenvolvimento de projetos de aplicações de software para TVs. Foi membro do Conselho do Fórum SBTVD-T, tendo participado dos esforços para a implantação do componente de interatividade na TV digital. Atualmente coordena o CTV Experience Lab do Claro tv+, integra o Conselho Deliberativo da SET, onde conduz um GT sobre OTT Video Streaming e participa do Programa de Pós-Graduação em Meios e Processos Audiovisuais da USP.

Contato: salustiano@set.org.br