

Universidade Federal de Juiz de Fora  
Faculdade de Engenharia e Instituto de Ciências Exatas  
Bacharelado em Engenharia Computacional

**Pedro Campos Lima**

**Suporte a *MPD Events* em protótipo do *Persistent Media Player* da TV 3.0**

Juiz de Fora  
2026

Pedro Campos Lima

**Suporte a *MPD Events* em protótipo do *Persistent Media Player* da TV 3.0**

Trabalho de Conclusão de Curso apresentado ao corpo docente do curso de graduação em Engenharia Computacional da Faculdade de Engenharia e Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Bacharel em Engenharia Computacional.

Orientador: Prof. Dr. Marcelo Ferreira Moreno

Juiz de Fora

2026

Pedro Campos Lima,

Suporte a *MPD Events* em protótipo do *Persistent Media Player* da TV 3.0/ Pedro Campos Lima. - - 2026.

XIII, 54 p.: il.; 29, 7cm.

Orientador: Marcelo Ferreira Moreno

Trabalho de Conclusão de Curso (bacharelado) –  
Universidade Federal de Juiz de Fora, ICE/Engenharia.  
Bacharelado em Engenharia Computacional, 2026.

Referências Bibliográficas: p. 51 – 54.

1. TV 3.0. 2. Player de Mídia Persistente. 3.  
Eventos MPD. 4. Metadados. 5. DASH. I. Ferreira  
Moreno, Marcelo. II. Universidade Federal de Juiz de Fora,  
Bacharelado em Engenharia Computacional.

Pedro Campos Lima

**Suporte a *MPD Events* em protótipo do *Persistent Media Player* da TV 3.0**

Trabalho de Conclusão de Curso apresentado ao corpo docente do curso de graduação em Engenharia Computacional da Faculdade de Engenharia e Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial à obtenção do grau de Bacharel em Engenharia Computacional.

Aprovado em 15 de julho de 2026.

**BANCA EXAMINADORA**

---

Prof. Dr. Marcelo Ferreira Moreno - Orientador  
Universidade Federal de Juiz de Fora

---

Prof. Dr. Eduardo Barrére  
Universidade Federal de Juiz de Fora

---

Prof. Dr. Marcelo de Oliveira Costa Machado  
Universidade Federal de Juiz de Fora

*À minha família*

## AGRADECIMENTOS

Agradeço em primeiro lugar à minha família por todo apoio durante a minha jornada. Sou grato por todo incentivo, compreensão e suporte oferecidos ao longo dessa caminhada, que foram fundamentais para que eu pudesse superar os desafios encontrados e alcançar esta etapa. Agradeço por sempre acreditarem em mim e por estarem presentes em todos os momentos.

Também agradeço à equipe da BrainyIT e à RNP. Sem os conhecimentos e oportunidades oferecidas durante o meu processo de formação, não teria sido possível desenvolver grande parte das habilidades e experiências que contribuíram para a minha trajetória. Agradeço pela confiança, pelo aprendizado compartilhado e pelo ambiente de crescimento proporcionado ao longo desse período.

Agradeço às pessoas que fizeram parte de toda a minha jornada acadêmica, seja na UFJF ou nas demais instituições pelas quais passei. Agradeço pelos momentos compartilhados, pelas memórias construídas e por todas as contribuições que, de alguma forma, fizeram parte da minha formação. Além disso, sou grato aos amigos que conheci fora do ambiente acadêmico, pelo apoio, companheirismo e por tornarem essa caminhada mais significativa.

Por fim, sou grato à Universidade Federal de Juiz de Fora e ao professor Marcelo Moreno pela orientação, dedicação e parceria. Agradeço pelos conhecimentos compartilhados, pelo apoio ao longo do desenvolvimento deste trabalho e pela contribuição para minha formação acadêmica e profissional.

*“Nunca me diga as  
probabilidades.”*

*(Han Solo - Star Wars Episódio  
V - O Império Contra-Ataca)*

## RESUMO

O consumo de conteúdos audiovisuais é cada vez mais presente na sociedade. Principalmente devido ao avanço de tecnologias de transmissão de dados, como na própria *internet*, os produtores tradicionais desse tipo de conteúdo também precisam evoluir. Com isso, foi sugerido o padrão TV 3.0, uma nova forma de interagir e de pensar na televisão, abrindo novas possibilidades da interatividade do telespectador com o conteúdo. Este trabalho apresenta uma evolução no protótipo do *Persistent Media Player*, visando habilitar o processamento nativo de *Media Presentation Description (MPD) Events* em conformidade com a norma ABNT NBR 25608 (2025). O objetivo foi implementar um mecanismo de extração, estruturação e distribuição de eventos presentes em manifestos DASH para aplicações interativas. Testes experimentais foram realizados para medir a latência de entrega de eventos sob diferentes resoluções e cargas de manifesto. Os resultados demonstram que, em cenários de múltiplos eventos simultâneos, a tendência da latência observada é baixa, indicando a viabilidade técnica da implementação proposta para ambientes de transmissão ao vivo. Dessa forma, a extensão contribui para o ecossistema da TV 3.0, implementando um requisito exigido pela norma em protótipo aplicável a ambientes de desenvolvimento de aplicações interativas e a cenários de testes de conformidade.

**Palavras-chave:** TV 3.0. Player de Mídia Persistente. Eventos MPD. Metadados. DASH.

## ABSTRACT

The consumption of audiovisual content is increasingly present in society. Mainly due to the advancement of data transmission technologies, such as the internet itself, traditional producers of this type of content also need to evolve. In this context, TV 3.0 standard was proposed, introducing a new way of interacting with and conceptualizing television, enabling new possibilities for viewer interaction with content. This work presents an evolution of the Persistent Media Player prototype, aiming to enable the native processing of Media Presentation Description (MPD) Events in compliance with the standard ABNT NBR 25608 (2025). The objective was to implement a mechanism for extracting, structuring, and distributing events present in DASH manifests for interactive applications. Experimental tests were conducted to measure event delivery latency under different resolutions and manifest loads. The results demonstrate that, in scenarios with multiple simultaneous events, the observed latency tends to remain low, indicating the technical feasibility of the proposed implementation for live streaming environments. Therefore, the extension contributes to the TV 3.0 ecosystem by implementing a requirement established by the standard in a prototype applicable to interactive application development environments and compliance testing scenarios.

**Keywords:** TV 3.0. Persistent Media Player. MPD Events. Metadata. DASH.

## LISTA DE ILUSTRAÇÕES

|     |                                                                                                                                                      |    |
|-----|------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 2.1 | Arquitetura da TV 3.0 AoP . . . . .                                                                                                                  | 23 |
| 4.1 | Diagrama com os elementos e fluxo do ambiente de testes. . . . .                                                                                     | 37 |
| 5.1 | Evento de identificador 0 no <i>player</i> . . . . .                                                                                                 | 46 |
| 5.2 | Evento de identificador 1 no <i>player</i> . . . . .                                                                                                 | 47 |
| 5.3 | Maiores latências medidas para o primeiro experimento com resolução de 2K. .                                                                         | 47 |
| 5.4 | Menores latências medidas para o primeiro experimento com resolução de 2K.                                                                           | 48 |
| 5.5 | Comparação entre a linha de tendência das maiores latências e das menores<br>latências para o primeiro com resolução de 2K. . . . .                  | 48 |
| 5.6 | Comparação entre a linha de tendência das médias de maiores latências e<br>médias das menores latências para a resolução de 2K. . . . .              | 49 |
| 5.7 | Comparação entre a linha de tendência das médias de maiores latências e<br>médias das menores latências para todas as resoluções experimentadas. . . | 49 |
| 5.8 | Comparação entre tendência de latência com todos os eventos e descartando<br>eventos expirados para a resolução 2K. . . . .                          | 50 |

## LISTA DE TABELAS

|     |                                                                    |    |
|-----|--------------------------------------------------------------------|----|
| 2.1 | Descrição e atributos dos elementos de <i>MPD Events</i> . . . . . | 20 |
|-----|--------------------------------------------------------------------|----|

## LISTA DE ABREVIACÕES E SIGLAS

|        |                                                  |
|--------|--------------------------------------------------|
| ABNT   | Associação Brasileira de Normas Técnicas         |
| AoP    | <i>Application-oriented Platform</i>             |
| DASH   | <i>Dynamic Adaptive Streaming over HTTP</i>      |
| PMP    | <i>Persistent Media Player</i>                   |
| MPD    | <i>Media Presentation Description</i>            |
| NCL    | <i>Nested Context Language</i>                   |
| DSM-CC | <i>Digital Storage Media Command and Control</i> |
| SGAI   | <i>Server-Guided Ad Insertion</i>                |
| SSAI   | <i>Server-Side Ad Insertion</i>                  |
| CSAI   | <i>Client-Side Ad Insertion</i>                  |
| OTA    | <i>Over The Air</i>                              |
| OTT    | <i>Over The Top</i>                              |
| QoE    | <i>Quality of Experience</i>                     |
| LSTM   | <i>Long Short-Term Memory</i>                    |
| LOWESS | <i>Locally Weighted Scatterplot Smoothing</i>    |
| MPEG   | <i>Moving Picture Experts Group</i>              |
| HTTP   | <i>Hypertext Transfer Protocol</i>               |
| URI    | <i>Uniform Resource Identifier</i>               |
| XML    | <i>Extensible Markup Language</i>                |
| HTML   | <i>Hypertext Markup Language</i>                 |
| CSS    | <i>Cascading Style Sheets</i>                    |

## SUMÁRIO

|          |                                                                                       |           |
|----------|---------------------------------------------------------------------------------------|-----------|
| <b>1</b> | <b>INTRODUÇÃO .....</b>                                                               | <b>14</b> |
| 1.1      | MOTIVAÇÃO E CONTEXTUALIZAÇÃO .....                                                    | 14        |
| 1.2      | OBJETIVOS .....                                                                       | 16        |
| <b>2</b> | <b>FUNDAMENTAÇÃO TEÓRICA .....</b>                                                    | <b>18</b> |
| 2.1      | O PADRÃO MPEG-DASH E A ESTRUTURA DE MPD EVENTS .....                                  | 18        |
| 2.1.1    | Organização do Manifesto.....                                                         | 18        |
| 2.1.2    | MPD Events.....                                                                       | 19        |
| 2.2      | ARQUITETURA DE SOFTWARE DA TV 3.0 E O <i>PERSISTENT MEDIA</i><br><i>PLAYER</i> .....  | 22        |
| 2.2.1    | <i>Persistent Media Player</i> .....                                                  | 24        |
| 2.2.2    | O Requisito Funcional PMPFR-14 e a exposição de metadados ..                          | 24        |
| 2.3      | O FRAMEWORK GSTREAMER E O COMPONENTE DASHDEMUX2                                       | 25        |
| 2.3.1    | Arquitetura Baseada em Pipelines e Elementos.....                                     | 25        |
| 2.3.2    | O Componente <i>dashdemux2</i> .....                                                  | 26        |
| 2.4      | <i>WebServices</i> e Comunicação entre <i>Persistent Media Player</i> e Aplicação ... | 26        |
| 2.4.1    | O Protocolo <i>WebSocket</i> .....                                                    | 27        |
| 2.4.2    | Modelo de Publicação-Assinatura para Eventos do <i>Player</i> .....                   | 27        |
| <b>3</b> | <b>TRABALHOS RELACIONADOS.....</b>                                                    | <b>28</b> |
| <b>4</b> | <b>METODOLOGIA E IMPLEMENTAÇÃO.....</b>                                               | <b>31</b> |
| 4.1      | MODIFICAÇÃO NO GSTREAMER E MECANISMO DE <i>PARSING</i> ....                           | 31        |
| 4.1.1    | Modelagem de dados dos eventos .....                                                  | 31        |
| 4.1.2    | Extensão do <i>parser</i> XML do Manifesto DASH .....                                 | 31        |
| 4.1.3    | Gerenciamento de memória e ciclo de vida do elemento <i>Period</i> ..                 | 32        |
| 4.1.4    | Mecanismo de Exposição de Eventos via <i>bus</i> .....                                | 33        |

|       |                                                                          |    |
|-------|--------------------------------------------------------------------------|----|
| 4.2   | INTEGRAÇÃO, TV 3.0 <i>WEBSERVICES</i> E PROTÓTIPO DO GINGA-HTML5 .....   | 33 |
| 4.2.1 | Captura e estruturação de eventos .....                                  | 34 |
| 4.2.2 | Distribuição de eventos via <i>WebServices</i> e <i>WebSockets</i> ..... | 35 |
| 4.2.3 | Protótipo do Ginga-HTML5 da TV 3.0 .....                                 | 36 |
| 4.3   | CENÁRIO DE TESTE E EXPERIMENTAÇÃO .....                                  | 37 |
| 4.3.1 | Servidor de <i>live streaming</i> DASH.....                              | 37 |
| 4.3.2 | Simulador de inserção de metadados .....                                 | 38 |
| 4.3.3 | Fluxo de Processamento no <i>Player</i> e Renderização em Tela .....     | 39 |
| 4.3.4 | Medição de latência no envio de eventos .....                            | 40 |
| 4.4   | AMBIENTE DE DESENVOLVIMENTO E COMPILAÇÃO .....                           | 41 |
| 4.4.1 | Especificações do ambiente de desenvolvimento.....                       | 41 |
| 4.4.2 | Gerenciamento de Dependências com <i>Docker</i> .....                    | 41 |
| 4.4.3 | Build com Meson e Ninja.....                                             | 42 |
| 4.4.4 | Ciclo de vida do contêiner e extração de artefatos.....                  | 42 |
| 5     | RESULTADOS E DISCUSSÕES .....                                            | 44 |
| 5.1   | AVALIAÇÃO DA TRANSMISSÃO DO EVENTO AO LONGO DOS COMPONENTES .....        | 44 |
| 5.1.1 | Verificação do <i>parsing</i> e extração no GStreamer.....               | 44 |
| 5.1.2 | Recepção dos eventos no protótipo do <i>Persistent Media Player</i> .    | 45 |
| 5.1.3 | Recepção e exibição dos eventos na interface gráfica.....                | 46 |
| 5.2   | ANÁLISE DE LATÊNCIA PARA ENTREGA DE EVENTOS .....                        | 47 |
| 6     | CONCLUSÃO .....                                                          | 51 |
| 6.1   | TRABALHOS FUTUROS .....                                                  | 51 |

# 1 INTRODUÇÃO

## 1.1 MOTIVAÇÃO E CONTEXTUALIZAÇÃO

Desde a consolidação da *internet*, a comunicação digital evolui de maneira acelerada, modificando a forma como a sociedade interage e consome informação. Nesse cenário, os aparelhos eletroeletrônicos estão cada vez mais conectados à rede mundial de computadores, o que trouxe novas oportunidades para a evolução da radiodifusão tradicional. É nesse contexto de transformação que o Brasil avança para a especificação e implementação da TV 3.0 (também denominada DTV+) (Fórum do Sistema Brasileiro de TV Digital Terrestre, 2026), a nova geração de televisão digital terrestre que adota uma arquitetura baseada em protocolos de internet, indo além dos métodos tradicionais de entrega de conteúdo exclusivamente pelo ar.

Esta evolução tecnológica converte o receptor de TV em uma plataforma na qual os fluxos de transmissão via *broadcast* e *broadband* se integram de maneira complementar e transparente para o telespectador. O grande diferencial dessa abordagem, estruturada sob o conceito de TV 3.0 *Application-oriented Platform* (TV 3.0 AoP), é o fato que a televisão passa a ser orientada a aplicações, conforme descrito por Moreno *et al.* (2023).

Importante para a viabilização dessa infraestrutura convergente com serviços *internet* está a adoção do padrão internacional MPEG-DASH (*Dynamic Adaptive Streaming over Hypertext Transfer Protocol (HTTP)*), padrão estabelecido para a entrega de conteúdos adaptativos via HTTP, especificado pelo padrão ISO/IEC 23009-1 (2022). Apesar da transmissão por radiodifusão terrestre não necessitar de tecnologias de *streaming* adaptativo, por usar um canal dedicado de comunicação unidirecional, a adoção do MPEG-DASH como padrão permite a sincronização entre os fluxos entregues por radiodifusão e *internet*, provendo a integração entre esses serviços.

Na arquitetura da TV 3.0, o protocolo DASH não apenas serve à função de entrega de áudio e vídeo, como também à entrega de um subconjunto de metadados, através de eventos. Conforme definido na Seção 5.10 da norma ISO, o DASH prevê métodos nativos para a entrega desses eventos sincronizados com o tempo de apresentação da mídia. Dentre

os métodos principais, destacam-se:

- *Media Presentation Description (MPD) Events*, eventos declarados de forma explícita dentro do elemento *EventStream* de um período (elemento *Period*), contendo informações para um cliente DASH ou aplicação;
- Eventos na Banda (*In-band Events*), sinalizados pelo elemento *InbandEventStream* e cujas mensagens brutas de dados ficam multiplexadas dentro dos próprios segmentos de mídia através de caixas *Event Message* (emsg).

O processamento dessas estruturas permite capturar os atributos dos eventos, como *schemeIdUri* (Uniform Resource Identifier (URI) que identifica o esquema do evento), *value* (que tem uma semântica definida pelos donos do *schemeIdUri*), *timescale* (unidade de escala de tempo) e *presentationTime* (momento de ocorrência do evento), possibilitando que interações sejam disparadas em sincronia com o fluxo audiovisual. A interatividade com conteúdos multimídia não é novidade da nova geração de televisão, dado que já existem formas de criar interações com os conteúdos na geração atual, como visto em ABNT NBR 15606-2 (2023). Usando a linguagem *Nested Context Language* (NCL), por exemplo, é possível montar apresentações de conteúdos com que o usuário pode interagir. Além disso, ela é especialmente voltada ao sincronismo entre mídias e à resposta a eventos em tempo de execução. A NCL possui a capacidade de ter a estrutura do documento hipermídia editada dinamicamente por meio de eventos de *stream Digital Storage Media Command and Control* (DSM-CC), permitindo que a aplicação se adapte ao fluxo da transmissão.

Para regulamentar este ecossistema de software da TV 3.0 no Brasil, a Norma ABNT NBR 25608 (2025) especifica os requisitos de codificação de aplicações e introduz um componente denominado *Persistent Media Player* (PMP). O PMP é o exibidor encarregado de realizar o gerenciamento da reprodução contínua e ininterrupta do conteúdo de mídia das emissoras, mantendo a estabilidade do vídeo e áudio mesmo durante as transições entre diferentes aplicações dentro do mesmo contexto do serviço DTV. O requisito funcional PMPFR-14 da norma estabelece também que o PMP deve suportar a

entrega de eventos de reprodução e de fluxo para as aplicações, incluindo explicitamente o processamento de *event streams* e *inband events*.

Apesar dessa demanda da norma, as ferramentas de reprodução e os *frameworks* de mídia de código aberto utilizados na atualidade, que poderiam fazer parte da prototipação do PMP, ainda apresentam lacunas no suporte completo a essas especificações. Por exemplo, o *framework* GStreamer (freedesktop.org, 2026), embora seja uma referência na decodificação e manipulação de fluxos multimídia, a implementação atual do componente *dashdemux2* (vinculado ao módulo *gst-plugins-good*) carece de suporte nativo capaz de extrair e processar os eventos declarados no manifesto por meio de (*MPD Events*).

Atualmente, o *framework* não realiza o parsing e a leitura dos elementos *EventStream* do MPD e seus elementos filhos *Event*, o que limita sua conformidade com as diretrizes da norma nacional e restringe a criação de ferramentas de validação do sistema. Diante dessa limitação, este trabalho foca no aprimoramento do suporte ao protocolo DASH pelo *framework* GStreamer, a fim de usá-lo em um protótipo do PMP.

## 1.2 OBJETIVOS

Este trabalho consiste em prototipar e avaliar o suporte a *MPD Events* em um *framework* multimídia que possa atuar como o PMP. Toma-se como base o *framework* de mídia de código aberto GStreamer, modificado para realizar o *parsing* e a exposição de *MPD Events*, alinhando-o aos requisitos exigidos para a TV 3.0. Para esse fim, foram definidos os seguintes objetivos:

- Estender as funções de leitura de manifesto do elemento *dashdemux2* do GStreamer para iterar sobre os elementos XML de *EventStream* e seus elementos filhos *Event*.
- Extrair e estruturar os atributos obrigatórios dos eventos em conformidade com a norma ABNT NBR 25608 (2025).
- Implementar uma camada de exposição baseada no TV 3.0 *WebServices* que permita a aplicações se inscreverem e receberem as notificações de eventos do *player* por meio de conexões via *WebSockets*.

- Integrar e avaliar a versão modificada do *framework* GStreamer com a prototipação da jornada do telespectador, desenvolvida no contexto do trabalho de Shuen *et al.* (2024).
- Medir a latência na entrega dos eventos para a *Broadcaster Application*, com e sem remoção de eventos expirados.

## 2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os conceitos e normas que sustentam o desenvolvimento deste trabalho. Inicialmente, detalha-se o padrão internacional MPEG-DASH e a estrutura de sinalização de eventos em manifestos. Em seguida, abordam-se as diretrizes da arquitetura de software da TV 3.0 no Brasil, com foco no *Persistent Media Player*. Por fim, são discutidos o funcionamento do *framework* GStreamer e os mecanismos de comunicação assíncrona baseados em *WebSockets*.

### 2.1 O PADRÃO MPEG-DASH E A ESTRUTURA DE MPD EVENTS

O padrão MPEG-DASH, apresentado no padrão ISO/IEC 23009-1 (2022), estabelece um formato de entrega de mídia baseado em segmentos adaptáveis sobre o protocolo HTTP. A arquitetura do DASH é dividida fundamentalmente em duas partes: a descrição da apresentação de mídia, contida no arquivo de manifesto denominado *Media Presentation Description* (MPD), e os segmentos físicos de mídia que contêm os fluxos de áudio, vídeo e dados estruturados.

#### 2.1.1 Organização do Manifesto

O manifesto MPD é um documento estruturado em formato XML (*Extensible Markup Language*) que mapeia a configuração do fluxo de transmissão. A estrutura do MPD possui uma hierarquia bem definida:

- *Period*: Representa um intervalo de tempo da transmissão de mídia. Um manifesto pode conter um ou mais períodos (por exemplo, dividindo o conteúdo principal de um bloco comercial).
- *AdaptationSet*: Contido dentro de um *Period*, agrupa os fluxos de mídia que contêm o mesmo conteúdo, mas codificados de formas distintas (por exemplo, um conjunto para vídeo, um para áudio em português e outro para áudio em inglês).
- *Representation*: Descreve uma versão alternativa do fluxo de mídia dentro de um

*AdaptationSet*, diferenciando-se por parâmetros de qualidade como taxa de bits (*bitrate*), resolução ou taxa de quadros.

Um exemplo simplificado de manifesto pode ser visto abaixo:

Código 2.1: Exemplo simplificado de MPD

```

1  ...
2  <Period id="0" start="PT0.0S">
3      <AdaptationSet id="0" contentType="video" startWithSAP="1"
          ↳ segmentAlignment="true" bitstreamSwitching="true" frameRate="30/1"
          ↳ maxWidth="1920" maxHeight="1080" par="16:9" lang="und">
4          <Resync dT="33333" type="0" />
5          <Representation id="0" mimeType="video/mp4" codecs="avc1.42c028"
          ↳ bandwidth="1904440" width="1920" height="1080" sar="1:1">
6              ...
7          </Representation>
8      </AdaptationSet>
9      <AdaptationSet id="1" contentType="audio" startWithSAP="1"
          ↳ segmentAlignment="true" bitstreamSwitching="true" lang="und">
10         <Resync dT="32000" type="0" />
11         <Representation id="1" mimeType="audio/mp4" codecs="ac-3"
          ↳ bandwidth="320000" audioSamplingRate="48000">
12             ...
13         </Representation>
14     </AdaptationSet>
15 </Period>
16 ...

```

### 2.1.2 MPD Events

A Seção 5.10 do padrão ISO/IEC 23009-1 (2022) introduz os mecanismos nativos para o transporte de metadados através de eventos. A norma define que a sinalização pode

ocorrer de forma multiplexada nos segmentos (*In-band Events*) ou de forma declarativa diretamente no XML do manifesto, denominados *MPD Events*.

Os *MPD Events* são estruturados por meio do elemento *EventStream*, que atua como um contêiner coletor de eventos e pode ser inserido diretamente no nível do *Period*. Para realizar o *parsing* completo desse elemento XML, o sistema deve mapear uma série de atributos e elementos filhos obrigatórios.

Tabela 2.1: Descrição e atributos dos elementos de *MPD Events*

| Elemento / Atributo       | Descrição                                                                                                                                                                             |
|---------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b><i>EventStream</i></b> | Elemento que agrupa eventos.                                                                                                                                                          |
| @schemeIdUri              | URI que identifica o esquema do evento.                                                                                                                                               |
| @value                    | A semântica e valor deste atributo deve ser definida pelos donos do esquema definido em @schemeIdUri.                                                                                 |
| @timescale                | Unidade de escala de tempo (unidades por segundo).                                                                                                                                    |
| <b><i>Event</i></b>       | Elemento filho de <i>EventStream</i> que representa uma instância individual de evento.                                                                                               |
| @id                       | Identificador numérico único para aquela instância específica de evento.                                                                                                              |
| @presentationTime         | O valor de tempo associado à ocorrência do evento, referenciado em relação ao início do <i>Period</i> e escalado pelo atributo @timescale.                                            |
| @duration                 | A duração do evento. O valor em segundos da duração é a divisão do valor desse atributo pelo @timescale                                                                               |
| @messageData              | A mensagem do evento, contendo os dados textuais relacionados a ele. Esse atributo atualmente está obsoleto, sendo recomendada a inserção da mensagem dentro do elemento <i>Event</i> |

Um exemplo fictício dos elementos *EventStream* e *Event* pode ser visto abaixo.

Código 2.2: Exemplo de EventStream e Event no MPD

```

1  ...
2  <Period id="0" start="PT0.0S">
3      ...
4      <EventStream schemeIdUri="urn:custom:product:selling:metadata"
5          ↪ timescale="1" value="best-seller">
6          <Event presentationTime="600" duration="30" id="0"
7              ↪ base64encoded="true">
8              eyJuYW11IjogJ2NhbWlzYS5scsICJ0ZW50IjogJ0JyYXNpbCcsICJ2YWx1ZSI6ICcwMD
9              AuMDAnLCAiY3VycmVuY3kiOiAncmVhbCd9
10             </Event>
11         </EventStream>
12     </Period>
13     ...

```

Esse evento poderia servir para um jogo de futebol ao vivo, em que o Brasil seria um dos times em campo. Nele, analisando os atributos da *EventStream* o *schemeIdUri*, com a semântica definida pelos donos da aplicação, indica para ela que é um metadado relacionado a um produto anunciado durante a partida. O campo *value* também definido pelos donos da aplicação, indica que é um dos produtos mais vendidos. Em relação aos atributos do *Event*, com o *presentationTime* é possível determinar que o evento aconteceria aos 600 segundos, ou 10 minutos, da transmissão. Pelo atributo *duration*, ele duraria 30 segundos e tem o identificador único *id* com valor 0. O evento está codificado em *base64*, como visto no atributo *base64encoded* e os metadados do evento estão na *string* inserida dentro do valor do elemento *Event*. Decodificando, essa string tem o valor:

```
{"name": 'camisa', "team": 'Brasil', "value": '000.00', "currency": 'real'}
```

## 2.2 ARQUITETURA DE SOFTWARE DA TV 3.0 E O *PERSISTENT MEDIA PLAYER*

Diferente das gerações anteriores de televisão digital, onde o middleware de interatividade rodava de forma acessória ao fluxo principal de vídeo, na TV 3.0 a experiência televisiva passa a ser integralmente mediada por aplicações multimídia sob os preceitos definidos pela TV 3.0 AoP, ABNT NBR 25608 (2025). Para a viabilização desse modelo, a arquitetura da TV 3.0 AoP é constituída de diversos componentes, sendo alguns deles importantes para o contexto deste trabalho.

O *Application Catalog* (seção 8 da ABNT NBR 25608 (2025)) é a principal interface de navegação dos telespectadores para poderem acessar as aplicações de radiodifusão. Ele deve listar as *Bootstrap Applications* anunciadas por serviços DTV, de forma que apenas os conteúdos relacionados a radiodifusão estejam disponíveis no catálogo. Por fim, ele é um meio centralizado de apresentação das emissoras disponíveis, de forma que o telespectador possa navegar e selecionar os conteúdos que deseja assistir.

As *Bootstrap Applications* (especificadas na seção 8.4 da ABNT NBR 25608 (2025)) são as aplicações iniciais instanciadas no *Application Catalog*. Essas aplicações são o ponto inicial para o telespectador interagir com os conteúdos de uma emissora. Elas são instanciadas durante a varredura de canais e controlam o PMP, apresentando o conteúdo selecionado. Para facilitar no processo de troca de serviços, são aplicações simples que gerenciam um grupo conciso de metadados.

Já as *Broadcaster Applications* (definidas na seção 6 da ABNT NBR 25608 (2025)) são aplicações personalizadas desenvolvidas pelas próprias emissoras. Enquanto as *Bootstrap Applications* são projetadas para serem simples e fornecerem funcionalidades essenciais, as *Broadcaster Applications* podem oferecer funcionalidades adicionais, principalmente voltadas para fornecer experiências personalizadas e também interativas para o telespectador. Cada emissora pode desenvolver várias *Broadcaster Applications*, e alterná-las automaticamente ou permitir que o telespectador selecione entre elas.

As máquinas de apresentação TV 3.0 Ginga-NCL (conforme o anexo A da ABNT NBR 25608 (2025)) e TV 3.0 Ginga-HTML5 (como descrito no anexo B da ABNT NBR 25608

(2025)) suportam linguagens de programação padronizadas e APIs para o desenvolvimento de *Broadcaster Applications*. A máquina TV 3.0 Ginga-NCL é um componente projetado para o processamento de documentos NCL, integrando também uma máquina Lua, para interpretação de objetos NCLua, objetos de mídia incorporados com código Lua. Enquanto isso, a máquina TV 3.0 Ginga-HTML5 é responsável por aplicações escritas em *Hypertext Markup Language* (HTML), *Cascading Style Sheets* (CSS) e *JavaScript*.

O TV 3.0 *WebServices* (especificado no anexo C da ABNT NBR 25608 (2025)) fornece um conjunto de APIs, responsáveis por prover o acesso a informações, recursos e funcionalidades da TV 3.0 AoP para as aplicações. Ele funciona como um intermediário que recebe requisições das aplicações e aciona os devidos componentes da arquitetura. Por exemplo, se uma aplicação quer receber informações de eventos que vierem via DASH, ela requisita a inscrição para o TV 3.0 *WebServices*, que aciona o PMP e este abre um *WebSocket* e retorna a URL para a aplicação se conectar e começar a receber esses eventos.

A seguir, é apresentado um diagrama da TV 3.0 AoP, com destaque nos componentes supracitados e no *Persistent Media Player*, que será detalhado posteriormente.

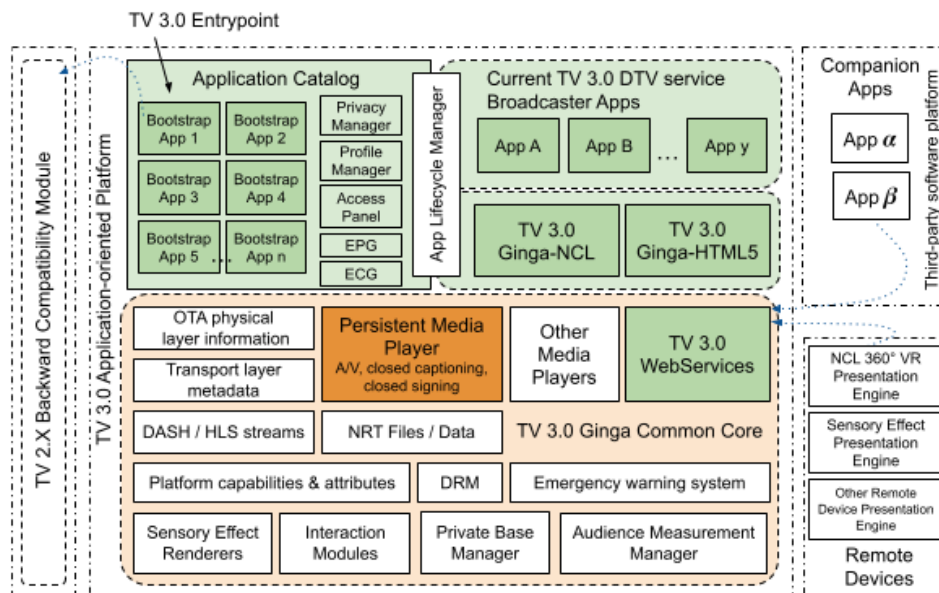


Figura 2.1: Arquitetura da TV 3.0 AoP

### 2.2.1 *Persistent Media Player*

O *Persistent Media Player* é o principal componente abordado neste trabalho. Dentro do modelo de arquitetura de receptores da TV 3.0 AoP, o PMP é o componente de software responsável por manter a exibição da mídia ininterrupta mesmo durante trocas de aplicações dentro do mesmo contexto de serviço DTV. Ele é o mecanismo unificado e de ciclo de vida contínuo encarregado de controlar a decodificação e exibição dos fluxos de áudio e vídeo de radiodifusão (*over-the-air* - OTA) ou banda larga (*over-the-top* - OTT).

O PMP opera de forma independente do ciclo de vida das aplicações interativas. Dessa forma, se a sinalização de uma emissora indicar o chaveamento entre sua aplicação inicial (*Bootstrap Application*) e uma aplicação interativa (*Broadcaster Application*), os fluxos de vídeo e áudio atuais não sofram interrupções ou descontinuidades.

### 2.2.2 O Requisito Funcional PMPFR-14 e a exposição de metadados

Para que as *Broadcaster Applications* consigam criar recursos interativos (como votações em tempo real, compra de produtos exibidos na tela ou troca dinâmica de anúncios comerciais), o PMP deve expor informações de eventos de DASH, especificamente da classe *dash-stream* de acordo com a norma ABNT NBR 25608 (2025), para as aplicações. Outro uso relevante de eventos é mostrado em SCTE 35-2 (2026), sendo ele a sinalização de oportunidades de inserção de anúncios, o que para emissoras de TV pode ser um uso significativo, dado que tais conteúdos comerciais podem ser escolhidos de acordo com o perfil do telespectador.

Além desses usos, vale ressaltar o uso de eventos para comando de edição NCL, como destacado na seção A.3.22 da ABNT NBR 25608 (2025). A máquina de apresentação TV 3.0 Ginga-NCL inclui a *Editing Commands API*, e esta permite as *Broadcaster Applications* manipularem bases privadas, controlem ciclos de vida de *Broadcaster Applications* e alterem as estruturas do documento NCL. Então, via *MPD Events*, uma aplicação pode receber um comando NCL e solicitar a adição de um documento a uma base privada, parada e/ou início da reprodução de um documento, por exemplo.

O requisito PMPFR-14 estabelecido na Seção 7.6.2 da norma previamente citada,

determina também que o PMP deve possuir a capacidade de extrair e repassar os fluxos de eventos (*event streams*) e eventos na banda para as aplicações, uma vez que inscritas, via *WebServices*, para receber notificações de eventos. A especificação técnica prevê que o *player* exponha um *web socket* em que a aplicação possa se conectar para receber as informações dos eventos recebidos pelo *player*.

## 2.3 O FRAMEWORK GSTREAMER E O COMPONENTE DASHDEMUX2

Para implementar um *player* para demonstrações e visando também a suíte de testes aderente aos requisitos da TV 3.0, o uso de frameworks de processamento multimídia existentes se torna uma solução viável. O GStreamer é uma biblioteca de código aberto escrita em linguagem C e estruturada sob o sistema de objetos GObject.

### 2.3.1 Arquitetura Baseada em Pipelines e Elementos

A arquitetura do GStreamer fundamenta-se no conceito de *pipelines* de processamento. Um *pipeline* é um grafo direcionado composto por múltiplos blocos funcionais conectados chamados Elementos (GstElement). Os dados fluem sequencialmente entre esses elementos através de pontos de conexão denominados Pads (GstPad). A seguir são apresentados alguns tipos de elementos do *pipeline* do GStreamer <sup>1</sup>:

- *Source Elements*: responsáveis por capturar os dados brutos de arquivos em disco ou de conexões de rede, receber e fornecer dados para o pipeline.
- *Transform Elements*: elementos que transformam *input buffers* em *output buffers* de acordo com os elementos de origem e destino. Exemplos são conversores de áudio, conversores de vídeo e filtros.
- *Sink Elements*: Renderizam a mídia na tela, saídas de áudio ou gravam o resultado em disco.

---

<sup>1</sup>Disponível em: <https://gstreamer.freedesktop.org/documentation/application-development/basics/elements.html>. Acesso em: 20 jul. 2026.

A comunicação de controle entre os elementos de um pipeline e a aplicação que gerencia o *player* é feita através de um canal de comunicação de mensagens chamado *Bus*. O *Bus* do GStreamer permite o envio assíncrono de notificações de fim de fluxo (*End-of-Stream*), mensagens de erro e metadados customizados.

### 2.3.2 O Componente *dashdemux2*

No GStreamer, as funcionalidades de streaming adaptativo baseadas no protocolo DASH são gerenciadas pelo elemento *dashdemux2*, localizado no módulo de plugins de alta qualidade *gst-plugins-good*.

A função primária do *dashdemux2* é realizar o download do manifesto MPD, a leitura inicial da árvore XML utilizando bibliotecas internas de parsing e coordenar as requisições HTTP dos segmentos de mídia apropriados para a banda de rede disponível.

No entanto, em sua versão nativa padrão, o elemento *dashdemux2* ignora as ramificações de nós XML que não correspondam diretamente a áudio ou vídeo. Os elementos *EventStream* e *Event* não são processados ou armazenados nas estruturas de dados internas em C do plugin. Consequentemente, o framework impossibilita que qualquer aplicação cliente tome conhecimento da existência de eventos declarados no manifesto DASH, o que não está em conformidade com o requisito PMPFR-14 da norma brasileira.

## 2.4 WEBSERVICES E COMUNICAÇÃO ENTRE *PERSISTENT MEDIA PLAYER* E APLICAÇÃO

A especificação da TV 3.0 AoP determina o desacoplamento entre o *Persistent Media Player* e o ambiente de execução *web* das aplicações. Por isso uma infraestrutura de rede local foi proposta para permitir a comunicação entre a aplicação e o PMP. Nesse contexto, incluído no *TV 3.0 Ginga Common Core*, o *TV 3.0 WebServices* faz a ponte de comunicação entre ambos os lados.

### 2.4.1 O Protocolo *WebSocket*

Diferente do modelo tradicional de comunicação HTTP que opera com base no modelo de requisição e resposta iniciado obrigatoriamente pelo cliente, o protocolo *WebSocket*, padronizado pela IETF através da RFC 6455, estabelece um canal de comunicação bidirecional, persistente e de baixa latência sobre uma única conexão TCP.

Uma conexão *WebSocket* inicia-se por meio de uma requisição *HTTP Upgrade* do cliente. Uma vez validada a requisição, o servidor deve responder ao cliente, que também deverá validar a resposta do servidor. Feito isso, a conexão *full-duplex* é aberta, eliminando os cabeçalhos HTTP e permitindo a troca contínua de dados de ambos os lados.

### 2.4.2 Modelo de Publicação-Assinatura para Eventos do *Player*

Para o cenário de notificação de eventos extraídos de manifestos DASH, o uso de conexões baseadas em *WebSockets* se mostra útil ao implementar o padrão de arquitetura de software conhecido como Publicação-Assinatura (Publish-Subscribe ou Pub-Sub):

- Publicador (*Publisher*): O *player* multimídia em Python integrado ao GStreamer atua como o publicador. Assim que o parser estendido do GStreamer identifica e extrai as informações de um novo *Event* no XML do manifesto, ele repassa essa mensagem via *WebSocket*.
- Assinantes (Subscribers): As Broadcaster Applications em execução assinam o tópico correspondente aos fluxos de eventos e conectam-se ao servidor *WebSocket* responsável por entregar as mensagens contendo informações do evento.

Dessa forma, sempre que um novo evento é capturado pelo *player*, o servidor *WebSocket* entrega o conteúdo do *payload* de maneira assíncrona para a aplicação conectada, eliminando a necessidade de consultas repetitivas de requisições HTTP, economizando recursos computacionais.

### 3 TRABALHOS RELACIONADOS

O envio de metadados durante transmissões tem sido estudado de maneira contínua. Os eventos no DASH representam uma forma de realizar esse tipo de envio, e agora é a forma adotada pela implementação da TV 3.0.

Giladi *et al.* (2026) apresenta uma análise sobre a *Server-Guided Ad Insertion* (SGAI) aplicada ao padrão DASH em sua 6ª edição. A proposta consiste no meio termo entre *server-side ad insertion* (SSAI) e *client-side ad insertion* (CSAI), buscando maior escalabilidade, padronização e flexibilidade. O artigo fala sobre o funcionamento dos *Alternative MPD Events*, mecanismo introduzido no MPEG-DASH que permite a substituição dinâmica de conteúdos durante a reprodução, possibilitando casos como inserção personalizada de anúncios, substituição de conteúdo e retorno ao fluxo principal de transmissão.

Mekuria *et al.* (2023) mostra uma abordagem para melhorar a interoperabilidade da sinalização SCTE-35 em *workflows Over The Top* (OTT) baseados em MPEG-DASH, com foco na inserção e substituição dinâmica de anúncios. O trabalho descreve o uso de *EventStream* e *Event* do DASH para transportar informações no manifesto MPD, permitindo a identificação de pontos de inserção de conteúdo e anúncios durante a reprodução adaptativa.

Altamimi e Shirmohammadi (2019) propõem um mecanismo cooperativo entre cliente e servidor para otimização de transmissões MPEG-DASH, utilizando aprendizado por reforço para realizar adaptação de *bitrate* com foco em qualidade de experiência e justiça na alocação de banda. A proposta utiliza o elemento *EventStream* do manifesto MPD para permitir atualizações dinâmicas no manifesto, mantendo compatibilidade com clientes DASH existentes. Nessas atualizações, o servidor disponibiliza para o cliente alternativas de *bitrate* de acordo com as condições da rede. Os resultados demonstram melhorias na *Quality of Experience* (QoE) e na distribuição de recursos da rede em comparação com métodos tradicionais.

Em Altamimi e Shirmohammadi (2020), os autores aprimoraram a versão de 2019 ao adicionar uma camada de *Long Short-Term Memory* (LSTM) na rede neural do agente,

para prever as dinâmicas da rede a longo prazo. Também foi alterado o algoritmo de treinamento do servidor, a fim de melhorar a qualidade com menos ações do servidor. Avaliado em cenários com clientes rodando o mesmo algoritmo de adaptação, e algoritmos diferentes, o novo modelo reduziu as ações de controle do servidor em até 22% e equilibrou a qualidade da experiência entre os usuários em até 10%.

O trabalho de Perkis (2021) traz uma solução de mídia interativa baseada em MPEG-DASH utilizando elementos *EventStream* e *Event*. A proposta utiliza esses eventos para sincronizar ações em uma segunda tela com a reprodução do vídeo principal, permitindo o disparo de conteúdos adicionais, como vibrações, áudios, vídeos e interações com o usuário. Além disso, os autores exploram o uso desses eventos para criar narrativas interativas, nas quais escolhas realizadas pelo usuário podem alterar o fluxo da reprodução por meio da atualização do manifesto. O estudo demonstra que os eventos do MPEG-DASH podem ser utilizados como mecanismo de sincronização de metadados temporais para aplicações multimídia interativas.

Ibrahim *et al.* (2017) apresentam uma abordagem para personalização de gráficos em transmissões de TV utilizando eventos *inband* do MPEG-DASH. A proposta consiste em separar os elementos gráficos do vídeo principal, permitindo que sejam renderizados no lado do cliente e personalizados de acordo com o perfil do usuário, região ou contexto da transmissão. Para isso, os autores utilizam esses eventos inseridos nos segmentos de mídia por meio de mensagens **emsg**, que funcionam como gatilhos para que o *player* busque e sobreponha os gráficos durante a reprodução. O trabalho demonstra uma implementação baseada em *players* DASH, mostrando que essa técnica permite manter a qualidade dos gráficos independente da qualidade do vídeo e possibilita experiências mais flexíveis e personalizadas em serviços de *streaming*.

Enquanto os trabalhos citados anteriormente mostram formas de se usar os eventos do DASH, o presente trabalho traz uma forma de extrair os eventos a partir de um MPD em um protótipo do *Persistent Media Player* da TV 3.0 e disponibilizá-lo no formato definido pela forma. Além disso, este trabalho traz a implementação do protótipo da máquina de apresentação Ginga-HTML5 na forma de um *browser web* customizado com

fundo transparente, permitindo com que o HTML seja renderizado à frente do PMP sem obstruí-lo.

## 4 METODOLOGIA E IMPLEMENTAÇÃO

### 4.1 MODIFICAÇÃO NO GSTREAMER E MECANISMO DE *PARSING*

Para se aproximar da aderência ao requisito PMPFR-14 da TV 3.0, foi necessário modificar diretamente o código-fonte do plugin de demultiplexação adaptativa, criando novas estruturas e ajustando a lógica interna. A implementação no GStreamer dividiu-se em quatro etapas: modelagem orientada a objetos usando o sistema GObject, extensão do parser XML, gerenciamento de memória e o mecanismo de notificação via *bus*.

#### 4.1.1 Modelagem de dados dos eventos

Como o GStreamer adota o sistema de tipos GObject, as estruturas de dados para mapear as especificações do padrão ISO/IEC 23009-1 (2022) precisaram ser criadas do zero. Para isso, foram desenvolvidos novos módulos funcionais compostos por arquivos de cabeçalho e arquivos com as lógicas de programação.

Os arquivos *gstmpdeventstreamnode.h* e *gstmpdeventstreamnode.c* contêm a estrutura de dados e as lógicas referentes ao elemento *EventStream* do MPD. Ela armazena os atributos presentes na *EventStream* original, como *schemeIdUri*, *value* e *timescale*, além de gerenciar a associação com seus respectivos elementos filhos, no caso, os elementos *Event*.

Já a estrutura e lógicas para o elemento *Event* estão nos arquivos *gstmpdeventnode.h* e *gstmpdeventnode.c*. Nela são armazenados os dados dos eventos que serão posteriormente expostos, como *id*, *presentationTime*, *duration*, *messageData* e o valor contido dentro do elemento *Event*.

#### 4.1.2 Extensão do *parser* XML do Manifesto DASH

Originalmente, o decodificador ignorava tais elementos do XML. Com as novas estruturas de dados devidamente modeladas, a lógica para extrair os dados de eventos do manifesto foi incorporada ao arquivo *gstmpdparser.c* e ao cabeçalho correspondente *gstmpdparser.h*.

Primeiro, a implementação da função encarregada de realizar o *parsing* do nó *Event* foi

feita. Chamada *gst\_mpdparser\_parse\_event\_node*, essa função busca os atributos textuais do elemento e popula os campos internos da estrutura *GstMpdEventNode*.

Em seguida, a função *gst\_mpdparser\_parse\_event\_stream\_node* foi implementada para realizar o *parsing* do elemento *EventStream*. Ela realiza a leitura dos atributos do elemento e inicia um laço de repetição para iterar sobre cada *Event* filho, chamando a função *gst\_mpdparser\_parse\_event\_node* para construir o conjunto de eventos associados.

Por último, foi modificada a função responsável por processar o *Period* do arquivo MPD. Ela passou a chamar a rotina *gst\_mpdparser\_parse\_event\_stream\_node*, fazendo com que ao ser lido o *Period*, o GStreamer verifique a existência de fluxos *MPD Events* acoplados àquele intervalo de tempo.

#### 4.1.3 Gerenciamento de memória e ciclo de vida do elemento *Period*

Por se tratar de um desenvolvimento em linguagem C de baixo nível, o ciclo de vida dos dados injetados na memória deve ser gerenciado cautelosamente, a fim de evitar vazamentos de memória que poderiam degradar a estabilidade do *player* ao longo do tempo.

O *framework* do GStreamer possui funções bem definidas para lidar com o gerenciamento de memória, e essas funções foram aproveitadas e adaptadas para os novos fluxos. O arquivo *gstmpdperiodnode.h* foi modificado, uma vez que ele representa o período e agora passa a receber uma lista de elementos *Glist \*EventStreams*. Essa lista armazena dinamicamente múltiplos elementos de fluxo de eventos à medida que eles são detectados pelo parser XML. No arquivo *gstmpdperiodnode.c*, no construtor do componente o ponteiro da lista `self->EventStreams` é explicitamente configurado como *NULL*.

Na rotina de destruição do componente (*gst\_mpd\_period\_node\_finalize*), foi adicionada uma rotina de limpeza de memória. Antes do *Period* ser desalocado pelo GStreamer, o sistema percorre a lista, liberando os recursos, *buffers* de texto e *structs* de dados vinculados a cada *EventStream* e *Event* processados.

#### 4.1.4 Mecanismo de Exposição de Eventos via *bus*

A última etapa da adaptação do GStreamer consistiu em pegar as estruturas de dados populadas pelo parser e disponibilizá-las para fora das rotinas internas do *plugin*. Essa lógica foi implementada no arquivo do demultiplexer adaptativo de DASH do framework: *gstdashdemux.c*. Em vez de criar canais de comunicação paralelos, a solução aproveitou a infraestrutura já consolidada do GStreamer, conhecido por *Bus*. O *bus* é a camada do GStreamer responsável por encaminhar mensagens das *threads* do fluxo de reprodução para uma aplicação em outro contexto de *threads*.

Uma nova rotina foi desenvolvida, denominada *gst\_dash\_demux\_expose\_mpd\_events*: função criada com o propósito de extrair as *EventStreams* do período atual e encapsular seus dados em uma mensagem genérica de elemento do GStreamer (*GstMessage* do tipo *ELEMENT*). Então, essa mensagem é injetada no *bus*, que expõe o metadado para a aplicação externa que gerencia o *player*.

A função de leitura original do manifesto, *gst\_dash\_demux\_process\_manifest*, foi modificada para chamar a nova rotina de exposição. Isso assegura que, em vídeos sob demanda ou no carregamento inicial da transmissão, todos os eventos declarados no arquivo MPD sejam extraídos e enviados para o *bus*. No cenário de transmissões em tempo real, o manifesto MPD é atualizado periodicamente. Diante disso, a função *gst\_dash\_demux\_update\_manifest\_data*, responsável por processar as atualizações do MPD em transmissões ao vivo também passou a chamar a rotina *gst\_dash\_demux\_expose\_mpd\_events*.

Essa integração faz com que a solução funcione para vídeos sob demanda (VoD) ou programações ao vivo, já que tanto os *MPD Events* no início de um vídeo quanto os inseridos no meio de uma transmissão ao vivo sejam capturados pelo parser e disponibilizados para a aplicação que gerencia o *player* do GStreamer.

## 4.2 INTEGRAÇÃO, TV 3.0 *WEBSERVICES* E PROTÓTIPO DO GINGA-HTML5

Após a implementação da lógica de extração no GStreamer, o projeto foi integrado ao protótipo do *Persistent Media Player* previamente existente. Esse protótipo é responsável

por receber as mensagens do *bus*, estruturar os dados no formato previsto pela norma ABNT NBR 25608 (2025) e distribuí-los para as aplicações inscritas no *WebSocket* de distribuição de eventos da classe *dash-stream*.

Para se aproximar do que é previsto para a TV 3.0, foi implementado o monitoramento de mensagens do tipo *ELEMENT* na função que escuta as mensagens do *bus* e a função que disponibiliza um *WebSocket* para que aplicações externas se conectem e recebam os eventos.

A fim de suportar a execução de aplicações em HTML, no caso deste trabalho as *Bootstrap* e *Broadcaster Applications*, foi implementado um protótipo da máquina de apresentação Gingga-HTML5 em *Python*, usando a biblioteca *PySide6*<sup>1</sup>, que permite acesso ao *framework* Qt 6 (2026). Desse *framework* foi usado o *Qt WebEngine*<sup>2</sup>, que por sua vez utiliza o motor do Chromium (2026) para fazer aplicações baseadas em *browser*. Ele é responsável por executar o código das aplicações HTML e foi implementado com o fundo transparente, permitindo com que o vídeo permaneça visível mesmo com os componentes HTML na tela, viabilizando a personalização dos conteúdos gráficos das aplicações.

#### 4.2.1 Captura e estruturação de eventos

O componente do reprodutor foi desenvolvido em *Python* utilizando a biblioteca PyGObject, que expõe os métodos nativos do GStreamer. A inicialização do *player* realiza o carregamento dos arquivos binários modificados (.so) contidos no diretório de compilação do *framework*, o que faz o pipeline utilizar o elemento *dashdemux2*.

A integração utiliza uma instância do plugin de alto nível *playbin3*, que abstrai grande parte da construção do *pipeline* de reprodução. Para interceptar as mensagens geradas em C, o *player Python* escuta o *bus* do sistema através do método `self.bus.connect("message", self.bus.call)`. A função de retorno monitora continuamente o fluxo de sinais.

Quando um sinal do tipo *Gst.MessageType.ELEMENT* é enviado via *bus*, o *player* verifica se a estrutura de dados interna possui o nome *dash-event*, definido na

<sup>1</sup>Disponível em: <https://doc.qt.io/qtforpython-6/gettingstarted.html>. Acesso em: 20 jul. 2026.

<sup>2</sup>Disponível em: <https://doc.qt.io/qt-6/qtwebengine-overview.html>. Acesso em: 20 jul. 2026.

implementação customizada no GStreamer.

Ao capturar a mensagem, a rotina chama uma função de preparação. Essa função realiza uma consulta ao pipeline (*query\_position*) para determinar a posição atual de reprodução da mídia em segundos no momento do recebimento do evento, injetando esse valor no atributo *currentTime*.

Os dados extraídos da estrutura enviada pelo GStreamer são convertidos em um dicionário Python contendo as chaves: *schemeIdUri*, *eventTime*, *value*, *currentTime*, *duration*, *id* e *data*. Esse dicionário é inserido em uma fila em memória (*self.awaiting\_dash\_events*), que funciona como um *buffer* de mensagens pendentes de transmissão.

#### 4.2.2 Distribuição de eventos via *WebServices* e *WebSockets*

Para isolar a execução do *player* de mídia das aplicações externas, cumprindo o princípio de desacoplamento da TV 3.0, a camada de *WebServices* foi implementada baseada no *framework FastAPI* (Ramírez (2026)).

Essa camada roda na porta 8000 e atua como o ponto de entrada para as aplicações do DTV+. Quando uma aplicação Broadcaster Application deseja escutar notificações, ela submete uma requisição REST para o *endpoint* */tv3/notifications/dash-stream*. Este serviço intercepta o pedido e faz o encaminhamento interno para a API de controle do *player*.

O servidor de controle do *player* roda na porta 1342. Ele recebe a requisição e gera um canal de comunicação persistente. A resposta para a camada de *WebServices* é um JSON contendo o *handle* de acesso e a URL de conexão ao *WebSocket* aberto pelo *player*.

A aplicação cliente utiliza a URL recebida para abrir uma conexão *WebSocket* direta com o *player*, que é gerenciada pela rotina assíncrona *handle\_dash\_stream\_notifications*. Essa função monitora continuamente o *buffer* de eventos (*self.awaiting\_dash\_events*). Assim que um novo evento é adicionado, o servidor faz a retirada do elemento da fila e dispara um envio no formato JSON através do canal *WebSocket* ativo, permitindo que a aplicação web receba o evento.

### 4.2.3 Protótipo do Ginga-HTML5 da TV 3.0

Um dos desafios da TV 3.0 é garantir que os componentes visuais do HTML (menus interativos, pop-ups, catálogos) fiquem sobrepostos ao vídeo, sem obstruir o fluxo do PMP. Para solucionar essa restrição, foi desenvolvido uma aplicação para servir como a máquina de apresentação Ginga-HTML5 utilizando o módulo *PySide6* e o motor de renderização *Chromium* (*QtWebEngine*).

O protótipo foi projetado para atuar como uma tela transparente de tamanho 1920x1080 *pixels* posicionada à frente do *Persistent Media Player*. A renderização é viabilizada por meio de configurações fornecidas pelo módulo usado para construção do protótipo.

No bloco de inicialização, o *script* altera o formato padrão de superfície gráfica através da classe *QSurfaceFormat*, alocando um buffer de canal *Alpha* de 8 bits (*setAlphaBufferSize(8)*), permitindo que o fundo do protótipo seja transparente.

A janela principal recebe a diretiva *FramelessWindowHint* para evitar com que a janela tenha bordas, barra de títulos ou molduras nativas do sistema operacional. Além disso, aplica-se a *flag WindowStaysOnTopHint*, configurando o protótipo para permanecer sempre no topo de todas as janelas do sistema, o que faz com que a interface HTML esteja sempre à frente dos outros elementos. Para que o vídeo do *GStreamer* seja visível por trás das marcações HTML, o atributo de *background* da janela é configurado como transparente (*Qt.GlobalColor.transparent*).

Essas configurações instruem o motor do *Chromium* a renderizar de forma opaca apenas os elementos HTML textuais ou gráficos que possuam cores declaradas em CSS. Todas as regiões vazias, fundos de página e áreas sem preenchimento da aplicação deixam de ser renderizadas, permitindo que o vídeo gerado pelo PMP passe pelo navegador. Essa funcionalidade é um requisito definido no Anexo B, seção B.4.1 da ABNT NBR 25608 (2025), e o protótipo do Ginga-HTML5 desenvolvido nesse trabalho é aderente a esse requisito.

### 4.3 CENÁRIO DE TESTE E EXPERIMENTAÇÃO

Para avaliar a implementação, foi desenvolvido um ambiente experimental de testes. O propósito desse cenário é simular uma transmissão de TV 3.0 ao vivo, englobando desde a geração dinâmica de eventos no lado da emissora até a recepção de áudio, vídeo e eventos payload na interface do telespectador.

O ambiente de testes é composto por um servidor de streaming DASH ao vivo, um injetor de metadados, que envia os dados para gerar eventos, o *player* com o GStreamer personalizado, a camada de *WebServices* e o navegador transparente, operando conforme detalhado na Figura 4.

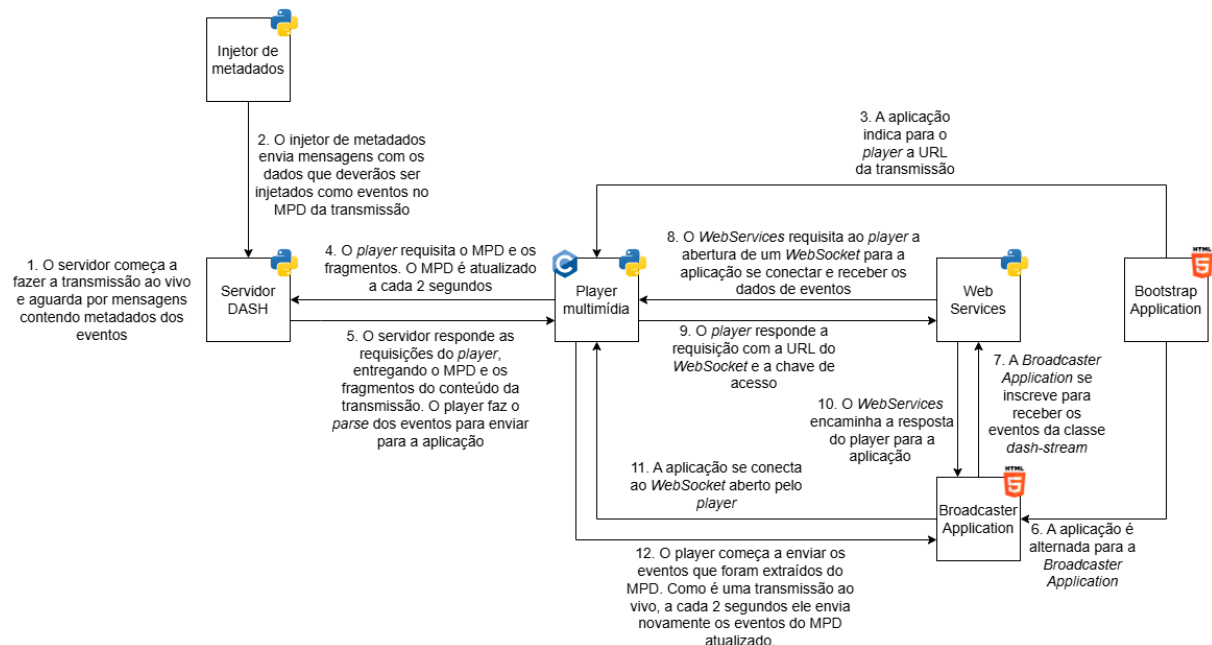


Figura 4.1: Diagrama com os elementos e fluxo do ambiente de testes.

#### 4.3.1 Servidor de *live streaming* DASH

O componente de backend encarregado de simular a transmissão ao vivo foi construído em Python utilizando o *framework web* Flask (Pallets Project (2026)). Este servidor tem duas funções principais operando simultaneamente: gerenciar o pipeline de codificação de vídeo e manipular o arquivo de manifesto DASH.

O servidor instancia um subprocesso em background executando um *pipeline* do

FFmpeg (Michael Niedermayer, 2026). O comando recebe como entrada um arquivo de vídeo local no formato MP4 em um laço infinito (*-stream\_loop -1*) e o codifica utilizando o *codec* H.264 para vídeo e cópia direta do *codec* de áudio original para o áudio. O FFMpeg fragmenta essa mídia em blocos de 2 segundos, configurando as propriedades para uma transmissão de baixa latência ao vivo (*-f dash -live 1*), gerando como saída inicial o arquivo de controle *manifest.mpd*. O vídeo utilizado ao longo dos testes foi o "Big Buck Bunny" (Blender Foundation (2008)).

O servidor expõe uma rota REST baseada no método HTTP POST. Esse *endpoint* funciona como um canal de escuta para que sistemas de automação externos enviem requisições com um corpo no formato JSON, contendo metadados para gerar eventos. Cada *payload* recebido é armazenado em uma lista em memória (*metadata\_array*).

Em uma *thread* paralela, o servidor executa um laço de monitoramento. Ele lê o manifesto gerado pelo FFMpeg e, utilizando a biblioteca de manipulação XML *xml.etree.ElementTree*, injeta o elemento *EventStream* dentro da tag *Period* atual. A *thread* percorre a lista de metadados recebidos pelo *endpoint* REST e insere os elementos filhos *Event* com seus respectivos atributos. O resultado é salvo em um novo arquivo denominado *live-manifest.mpd*, que é atualizado constantemente e servido para o *player*.

#### 4.3.2 Simulador de inserção de metadados

Para simular as ações de uma emissora, como a entrada de um comando de publicidade dinâmica, um aviso jornalístico ou um gatilho de interatividade, foi desenvolvido um *script* em *Python* denominado como injetor de metadados.

O *script* opera em um laço infinito para continuamente enviar dados de eventos para o servidor. Após uma espera inicial de 10 segundos (tempo necessário para o servidor estabilizar a geração dos primeiros blocos de vídeo), o injetor começa a enviar metadados de eventos a cada 1 segundo. A cada iteração do ciclo, o *script* constrói um objeto contendo os atributos requeridos pela norma (como *presentationTime*, *value*, *duration* e *id*). Para verificar o *parser* personalizado do *dashdemux2* na leitura dos campos de carga útil do evento, o injetor alterna o local em que ele será escrito, abrangendo as

possibilidades da norma: em rodadas ímpares ele envia o texto de teste no corpo do *Event*, e em rodadas pares ele injeta o texto dentro do atributo específico *messageData*.

O objeto é enviado via requisição HTTP POST para o *endpoint /metadata* do servidor *Flask*, simulando o comportamento de um sistema de pós-produção injetando dados na transmissão em tempo real.

### 4.3.3 Fluxo de Processamento no *Player* e Renderização em Tela

Com o ambiente transmissão operando, recebendo metadados e transformando-os em *MPD Events*, o próximo passo acontece no *player* desenvolvido para o trabalho.

O fluxo é chamado inicialmente abrindo o catálogo de aplicações, como seria em um televisor aderente à TV 3.0. Ao selecionar uma aplicação, ela é aberta, com o navegador transparente a frente do *player*, como esperado. No cenário de teste, selecionando a aplicação da TV Brasil abre a *Bootstrap Application*, que ainda não recebe eventos. A partir do momento em que é trocado para a *Broadcaster Application*, ela se registra e passa a receber os eventos que estão no *player*.

À medida que novos metadados são injetados e o servidor atualiza o arquivo *XML*, a rotina em C modificada no componente *gstdashdemux.c* identifica os nós *EventStream* e *Event*. Ao realizar o *parse* desses elemento, ele despacha a mensagem do tipo *dash-event* via *bus*. O *player Python* captura essa mensagem do *bus* via *PyGObject*, adiciona o evento à fila interna de notificações e o envia de através do canal *WebSocket* mantido com a aplicação web.

O navegador web customizado recebe o objeto JSON contendo o *payload* do evento através da conexão *WebSocket*. Como a interface HTML está posicionada de maneira fixa à frente da tela e configurada com transparência (canal Alpha ativado), o script *JavaScript* da aplicação consegue capturar a mensagem e renderizar o texto do *payload* diretamente sobre o vídeo em execução.

#### 4.3.4 Medição de latência no envio de eventos

Além do *parse* e construção do fluxo de envio dos eventos, foram coletados dados e feita uma análise sobre a latência nesse envio. Apesar de em um receptor real não existir a disputa de recursos para envio de eventos e decodificação de vídeo, já que haverá um *hardware* dedicado para isso, essa análise se mostra relevante uma vez que esse protótipo pode rodar em ambientes variados, possuindo ou não *hardware* dedicado para decodificação de vídeo.

Para avaliar a latência ao enviar os eventos, foram salvos em arquivos de *log* o momento em que o MPD é requisitado pelo *player*. Após o *parse* ser finalizado, cada evento é enviado para a aplicação e ela responde via *web socket* indicando que recebeu o evento. Então, o momento da recepção dessa resposta no player é, também, salvo em um arquivo de log.

Esse teste foi feito primeiramente mantendo todos os eventos recebidos no servidor durante 11 minutos de uma transmissão ao vivo, para as resoluções de 360p, 480p, 720p, 1080p e 2K (1440p). Depois, foi selecionado o caso com maiores latências e feito o teste novamente, mas removendo os eventos cujo *presentationTime* já aconteceu há mais de 1 minuto, ou seja, foram removidos os eventos expirados do manifesto. Essa remoção acontece no servidor, uma vez que ele salva todos os metadados recebidos do injetor de metadados e os escreve no MPD periodicamente. Então, os eventos expirados não são reescritos nos novos manifestos que serão disponibilizados para o PMP.

Para cada uma das resoluções avaliadas nos 2 cenários, foram realizados 3 testes de medição do tempo total da latência. Os dados dos arquivos de *logs* foram organizados em *dataframes* usando a biblioteca *Pandas* (pandas development team (2020)).

Com essas informações, foram extraídas e comparadas as médias de tempo do recebimento da resposta do primeiro e do último evento para cada MPD de cada uma das resoluções testadas. Com isso, devido a natureza do comportamento dos valores obtidos, foi utilizado o método de regressão paramétrica LOWESS (Locally Weighted Scatterplot Smoothing) Seabold e Perktold (2010), obtendo a curva que melhor se adequa ao conjunto de dados. Essas informações foram usadas para a geração de gráficos usando a biblioteca *Matplotlib* (Hunter (2007)).

## 4.4 AMBIENTE DE DESENVOLVIMENTO E COMPILAÇÃO

Para a realização das modificações no *framework* GStreamer e posterior integração com a camada de *WebServices*, foi configurado um ambiente de compilação isolado. O uso de ferramentas de containerização foi adotado para facilitar a reprodutibilidade do *build*, evitar conflitos de dependências com o sistema operacional hospedeiro e padronizar o fluxo de compilação dos artefatos modificados.

### 4.4.1 Especificações do ambiente de desenvolvimento

As etapas de desenvolvimento, compilação e execução experimental foram conduzidas em uma ambiente equipado com as seguintes especificações:

- Sistema Operacional: Ubuntu 24.04 LTS (*Noble Numbat*);
- Processador: Intel Core i7-1265U de 12<sup>a</sup> Geração (12 núcleos, 2 *threads* por núcleo);
- Memória RAM: 32 GB DDR4.

### 4.4.2 Gerenciamento de Dependências com *Docker*

A fim de evitar a poluição do sistema hospedeiro e garantir um ambiente limpo, utilizou-se a tecnologia de contêineres Linux através da plataforma *Docker* (Docker Inc. (2026)).

Um arquivo de configuração *Dockerfile* baseado na imagem oficial do Ubuntu 24.04 foi construído para automatizar o provisionamento do ambiente. A execução desse arquivo realiza a compilação e disponibiliza a versão do GStreamer que será utilizada no protótipo do *Persistent Media Player*. O processo de preparação dessa versão foi estruturado em três etapas:

1. Instalação de ferramentas base: Inclusão de pacotes necessários para compilação como *build-essential*, *git*, *pkg-config*, além do analisador léxico *flex* e o analisador sintático de propósito geral *bison*, que são pré-requisitos do GStreamer.
2. Provisionamento de bibliotecas: Instalação de dependências relacionadas a cabeçalhos e bibliotecas para compilação, como *libglib2.0-dev*, que provisiona

cabeçalhos relacionados à GLib, *libsoup2.4-dev*, uma biblioteca de cliente/servidor HTTP em C, projetada especificamente para o ecossistema GNOME, *libjson-glib-dev*, para lidar com arquivos do tipo JSON, componentes do ecossistema Wayland/X11 para gerenciamento da exibição gráfica no Linux e drivers gráficos Mesa.

3. Ambiente Python: Criação de um ambiente virtual Python (*venv*) interno para a instalação e atualização das ferramentas de *build* Meson e Ninja.

#### 4.4.3 Build com Meson e Ninja

O GStreamer utiliza o sistema de geração de arquivos de compilação Meson combinadamente com o sistema de *build* de código aberto Ninja. No *Dockerfile*, a configuração do projeto foi realizada por meio do utilitário *meson setup*, aplicando as diretivas de compilação que habilitam os módulos necessários ao escopo deste trabalho. Destacam-se os seguintes parâmetros configurados:

- *-Dgst-plugins-good:adaptivedemux2=enabled*: diretiva que habilita a compilação do módulo de demultiplexação adaptativa, onde reside o componente *dashdemux2* alvo das alterações de código.
- *-Dpython=enabled* e *-Dintrospection=enabled*: parâmetros que ativam a geração dos metadados de introspecção do GObject (*GObject Introspection*), permitindo que as alterações realizadas no GStreamer fiquem acessíveis para os *bindings* em Python do protótipo do PMP da UFJF.
- *-force-fallback-for=glib,gobject-introspection*: diretiva que força o download e compilação local dessas dependências, assegurando a consistência das bibliotecas manipuladas pelo *framework*.

#### 4.4.4 Ciclo de vida do contêiner e extração de artefatos

Com o ambiente devidamente configurado, o ciclo de compilação e extração dos binários alterados se dá através da execução sequencial de comandos via interface de linha de

comando (CLI). A imagem *Docker* (Docker Inc. (2026)) foi gerada forçando a reconstrução completa via parâmetro *-no-cache* para garantir a incorporação de modificações feitas no código-fonte local do GStreamer copiado para o diretório */tmp/gstreamer*:

```
docker build -t gstreamer-static:final . --no-cache
```

Após a finalização do processo de compilação gerenciado pelo Ninja dentro do contêiner, o ciclo de execução consiste em remover qualquer instância obsoleta do contêiner de execução e instanciar um novo ambiente baseado na imagem recém-gerada:

```
docker rm gstreamer
```

```
docker run --name gstreamer -it gstreamer-static:final bash
```

Por fim, com o contêiner mantido em execução ativa em segundo plano, utiliza-se um terminal secundário na máquina hospedeira para extrair o diretório de compilação contendo os binários e bibliotecas dinâmicas geradas (*builddir*):

```
docker cp gstreamer:/tmp/gstreamer/builddir .
```

Após a extração, o ambiente hospedeiro tem acesso a versão customizada do GStreamer, pronta para ser vinculada e executada pela aplicação do *player* multimídia.

## 5 RESULTADOS E DISCUSSÕES

Este capítulo apresenta a análise dos resultados obtidos a partir dos experimentos do ambiente de teste. O fluxo foi monitorado por meio da coleta de logs em três partes distintas do *pipeline* de dados: no GStreamer, no protótipo do *Persistent Media Player* e no console de desenvolvimento da *Broadcaster Application*. Além disso, foi feita uma análise de latência entre a requisição do manifesto pelo *player* e a resposta da aplicação sobre a chegada dos eventos.

### 5.1 AVALIAÇÃO DA TRANSMISSÃO DO EVENTO AO LONGO DOS COMPONENTES

#### 5.1.1 Verificação do *parsing* e extração no GStreamer

A primeira etapa consistiu em verificar que o elemento *dashdemux2* foi capaz de iterar sobre o manifesto *live-manifest.mpd*, identificar a tag *EventStream* e extrair os atributos dos nós *Event*. Os eventos gerados no servidor foram:

```
{'presentationTime': 10, 'value': 1, 'duration': 5, 'id': 0,
'messageData': '', 'base64encoded': 'true', 'data': 'message 1'}
```

```
{'presentationTime': 20, 'value': 2, 'duration': 18, 'id': 1,
'messageData': 'message 2', 'base64encoded': 'true', 'data': ''}
```

Durante a execução da live, o terminal do GStreamer registrou as seguintes mensagens de aviso (*WARN*) geradas:

```
0:00:09.712969079      296569      296862 WARN      dashdemux2
gstdashdemux.c:1204:gst_dash_demux_expose_mpd_events:<dashdemux2-0>
schemeIdUri urn:metadata, timescale 1, eventTime 10, base64encoded true,
duration 5, id 0, messageData , value metadata, content message 1
```

```
0:00:19.721590952      296569      296862 WARN      dashdemux2
```

```
gstdashdemux.c:1204:gst_dash_demux_expose_mpd_events:<dashdemux2-0>
schemeIdUri urn:metadata, timescale 1, eventTime 20, base64encoded true,
duration 18, id 1, messageData message 2, value metadata, content
```

Os *logs* do GStreamer mostram sucesso na implementação do *parser* frente à flexibilidade do XML de teste injetado pelo cliente de automação.

No Evento 0 (id 0), o *payload* do evento foi enviado como o texto interno do elemento *Event*, sendo capturado corretamente pelo GStreamer no campo *content* com valor *message 1*, enquanto o atributo *messageData* permaneceu vazio.

No Evento 1 (id 1), o comportamento inverteu-se: o *payload* foi inserido diretamente como um atributo do elemento, fazendo com que o *parser* a extraísse no campo *messageData* com o valor *message 2*, deixando o valor do *Event*, *content*, vazio.

Essa variação demonstra a conformidade com a norma brasileira do código em C desenvolvido com relação ao conteúdo do evento que pode ser enviado via *messageData* ou valor interno do *Event*, já que se mostrou apto a varrer e extrair os metadados independentemente da estratégia adotada pelo gerador do fluxo DASH.

Além disso, a presença dos outros elementos requisitados pela norma indicam o correto *parsing* do evento pela implementação adicionada ao GStreamer.

### 5.1.2 Recepção dos eventos no protótipo do *Persistent Media Player*

Uma vez injetados no *bus* do GStreamer, os eventos precisam ser interceptados pelo *player*. O script *player.py*, usando *PyGObject*, capturou as mensagens do tipo *ELEMENT* e registrou o seguinte comportamento no console da aplicação:

```
{'schemeIdUri': 'urn:metadata', 'eventTime': 10, 'value': 'metadata',
'duration': 5, 'id': '0', 'data': 'message 1', 'base64encoded': 'true',
'currentTime': 9.882944}

{'schemeIdUri': 'urn:metadata', 'eventTime': 20, 'value': 'metadata',
'duration': 18, 'id': '1', 'data': 'message 2', 'base64encoded': 'true',
'currentTime': 13.994901}
```

Os *logs* do *Python* confirmam que a adaptação no GStreamer realizou com sucesso o transporte dos dados extraídos do MPD. É possível notar que as informações são as mesmas presentes na camada anterior, o que mostra que as informações extraídas foram devidamente passadas para o protótipo do PMP.

### 5.1.3 Recepção e exibição dos eventos na interface gráfica

O protótipo do Ginga-HTML5 utilizou um navegador customizado baseado em *PySide6* sobreposto à janela de renderização do vídeo do PMP.

A ativação do canal *Alpha* e a remoção das molduras do sistema operacional permitiram que a página HTML *web* recebesse os objetos enviados pelo *player* no formato JSON e realizasse a renderização textual dos conteúdos desses objetos diretamente sobre a tela.

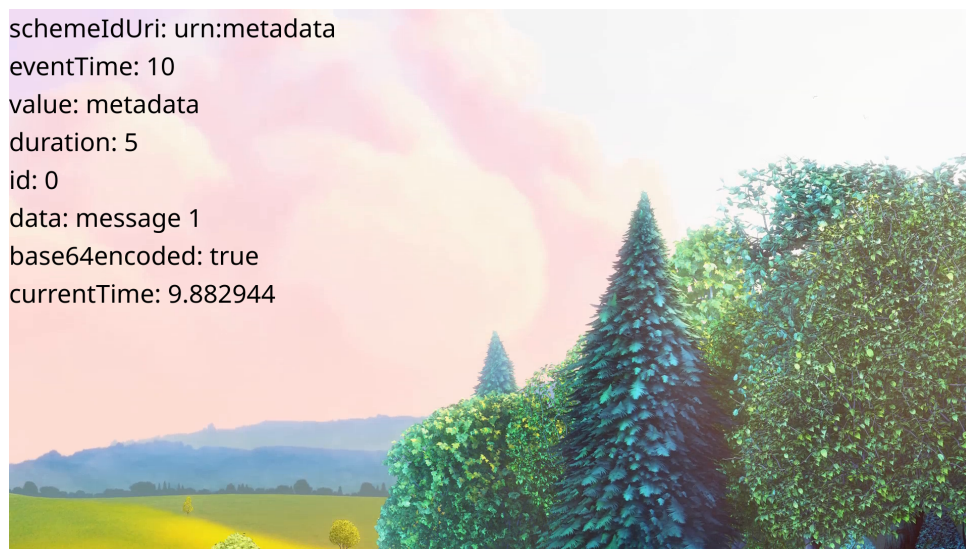


Figura 5.1: Evento de identificador 0 no *player*.



Figura 5.2: Evento de identificador 1 no *player*.

## 5.2 ANÁLISE DE LATÊNCIA PARA ENTREGA DE EVENTOS

A seguir, são apresentados gráficos das menores e maiores latências obtidas, com as linhas de tendências respectivas e depois a comparação entre elas para a resolução de 2K e, por último, o gráfico comparativo entre as linhas de tendência de cada resolução.

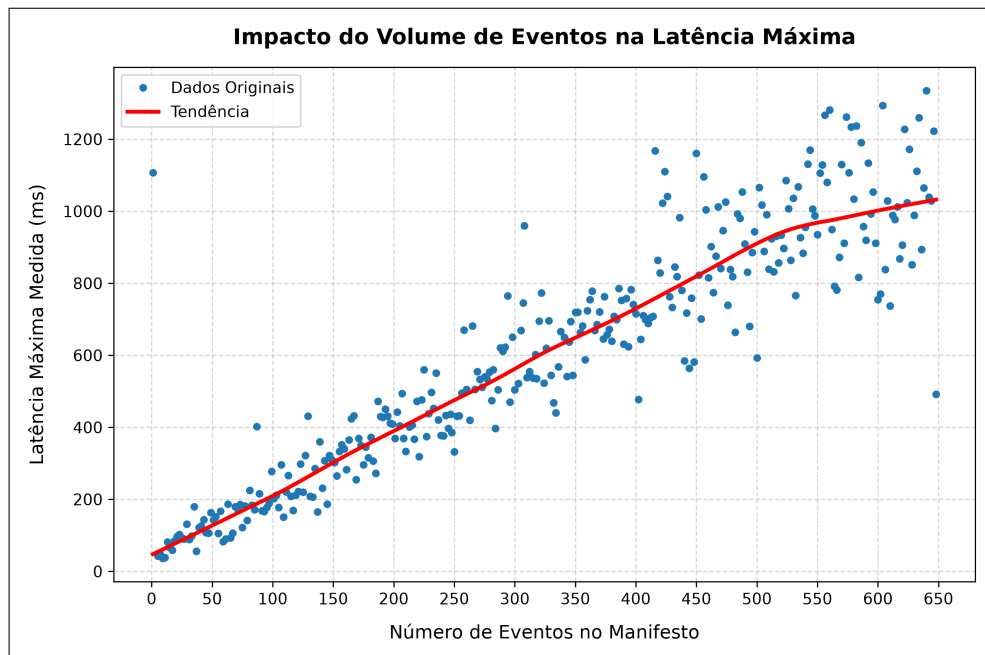


Figura 5.3: Maiores latências medidas para o primeiro experimento com resolução de 2K.

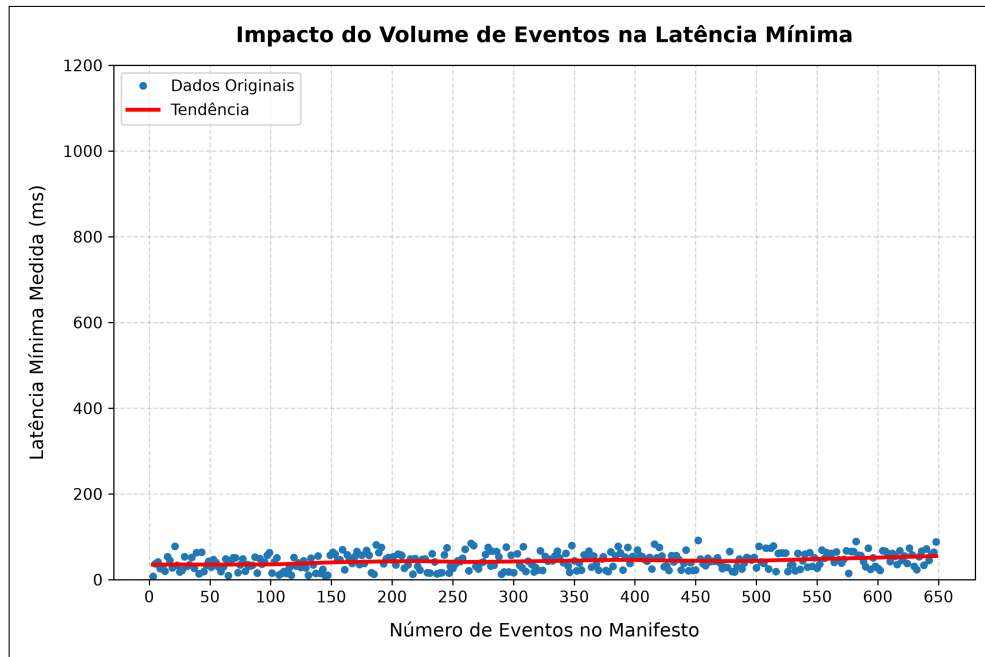


Figura 5.4: Menores latências medidas para o primeiro experimento com resolução de 2K.

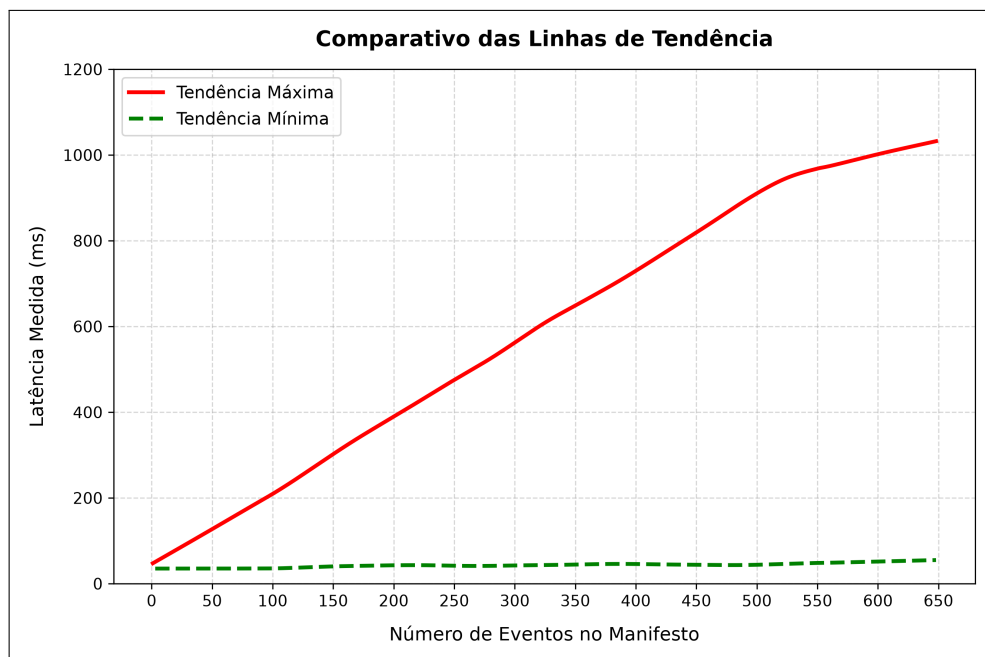


Figura 5.5: Comparação entre a linha de tendência das maiores latências e das menores latências para o primeiro com resolução de 2K.

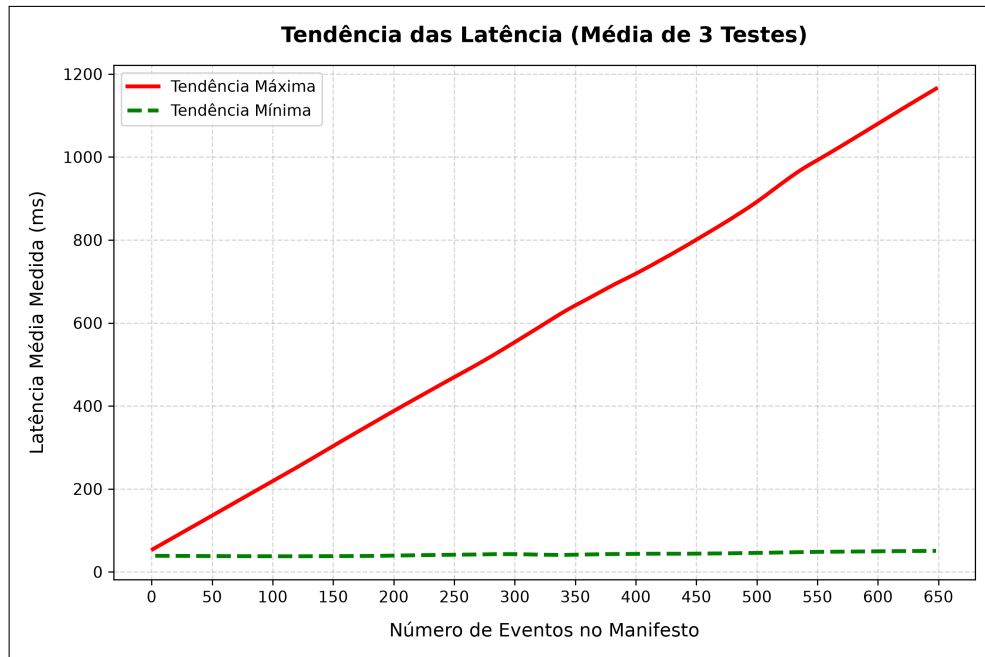


Figura 5.6: Comparação entre a linha de tendência das médias de maiores latências e médias das menores latências para a resolução de 2K.

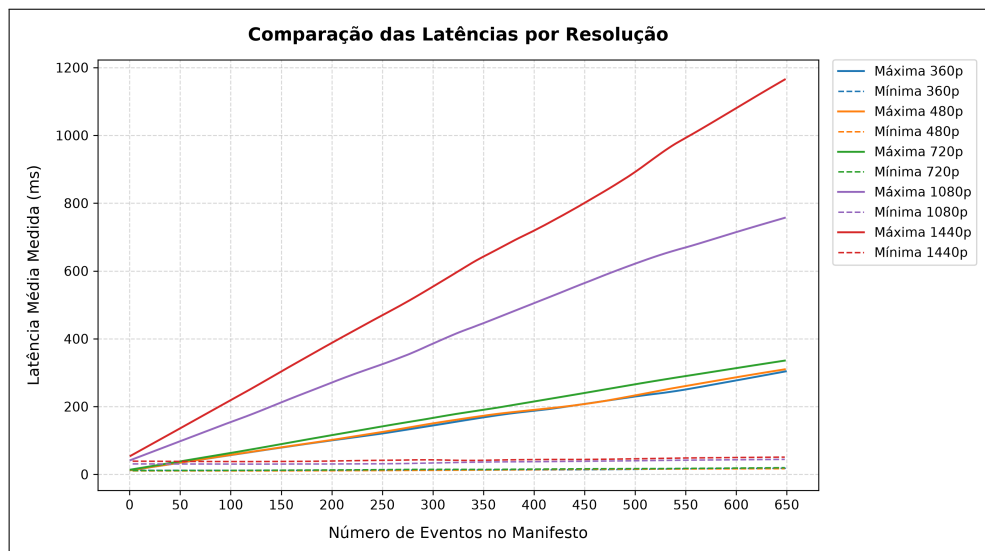


Figura 5.7: Comparação entre a linha de tendência das médias de maiores latências e médias das menores latências para todas as resoluções experimentadas.

Para as maiores latências medidas, é importante ressaltar o começo do gráfico para um comportamento inesperado, um evento com latência superior a 1 segundo. A explicação para este evento se deve ao primeiro manifesto recebido já conter eventos. Dessa forma, ao fazer a primeira requisição para o servidor, ao receber a resposta, o GStreamer faz a

instanciação de todos os elementos para reproduzir o conteúdo, até que finalmente envia os eventos para o *player Python* e por fim para a aplicação HTML.

Principalmente por usar o *playbin3*, que abstrai e simplifica o *pipeline* para reprodução do conteúdo, a preparação dos elementos corretos para reprodução é um pouco mais longa, acarretando na demora do envio da primeira leva de eventos. Além disso, um fluxo alternativo mais eficiente é chamado para a atualização do MPD durante a transmissão.

A seguir é apresentado o gráfico com a remoção dos eventos expirados.

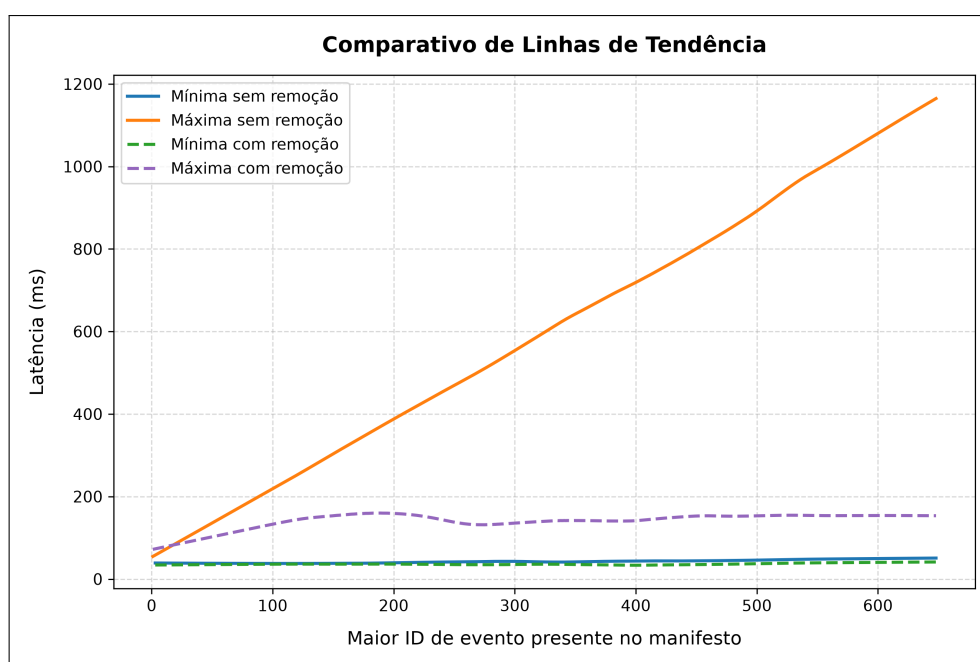


Figura 5.8: Comparação entre tendência de latência com todos os eventos e descartando eventos expirados para a resolução 2K.

Como visto no gráfico, a adição de uma lógica para remover os eventos expirados do manifesto fez com que a latência estabilizasse abaixo de 200 milissegundos, enquanto manter todos os eventos no manifesto faz com que a latência aumente cada vez mais.

Assumindo que os eventos estão ordenados cronologicamente, para o cenário de uma transmissão ao vivo, o acúmulo de eventos que já foram exibidos no manifesto degrada a eficiência de entrega de novos eventos. Isso pode fazer com que haja a dessincronização de eventos com a transmissão, prejudicando a qualidade da experiência do telespectador. Portanto, é importante que a emissora faça o devido controle dos eventos que serão transmitidos e remoção de eventos que já não serão mais parte da transmissão.

## 6 CONCLUSÃO

O presente trabalho atingiu o objetivo de estender o framework GStreamer para suportar nativamente a leitura e o processamento de *MPD Events*, aproximando o *player* das exigências da TV 3.0 no Brasil. Através da modificação do componente *dashdemux2* e da implementação de um mecanismo de exposição de eventos via *bus* do GStreamer, foi possível garantir que metadados declarativos em manifestos DASH sejam capturados tanto em vídeos sob demanda quanto em fluxos ao vivo. A integração desse framework modificado com a camada de *WebServices* e a utilização de *WebSockets* permitiu o desacoplamento entre o reprodutor de mídia e as aplicações, atendendo aos requisitos de arquitetura da norma ABNT NBR 25608 (2025). Além disso, a implementação do navegador customizado com renderização transparente demonstrou a viabilidade de criar *overlays* de interface sobre o conteúdo de mídia, outro aspecto exigido pela norma. Os resultados obtidos mostraram a capacidade de extração, estruturação e distribuição assíncrona de eventos com latência aceitável, principalmente no caso de remoção de eventos expirados do MPD.

### 6.1 TRABALHOS FUTUROS

Como trabalhos futuros, sugere-se a expansão dessa implementação para o suporte a *In-band Events* além de testes de carga para cenários com resoluções maiores como 4K. Também, sugere-se a containerização dos componentes desenvolvidos, ou seja, fazer as imagens das aplicações desenvolvidas para promover portabilidade e isolamento na execução dos componentes. Dessa forma, a realização de testes em máquinas diferentes é facilitada, assim como promover cenários com limitações de recursos computacionais e congestionamento de rede.

Além disso, para ainda maior aderência à norma ABNT NBR 25608 (2025), pretende-se fazer a implementação dos momentos de envio de eventos definidos na seção C.6.6.9. A implementação atual contempla apenas o envio de eventos com *dispatchMode onReceive*, enquanto a norma prevê o *dispatchMode onStart*, que define que os eventos devem ser notificados quando são esperados que iniciem.

## REFERÊNCIAS

- ABNT NBR 15606-2, 2023. Norma ABNT NBR 15606-2, Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações.
- ABNT NBR 25608, 2025. TV 3.0 — Codificação de aplicações. Norma ABNT NBR 25608, ABNT - Associação Brasileira de Normas Técnicas.
- Altamimi, S., Shirmohammadi, S., Client-Server Cooperative and Fair DASH Video Streaming. In: In Proceedings of the 29th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '19), 2019. Association for Computing Machinery, New York, NY, USA, pp. 1–6.
- Altamimi, S., Shirmohammadi, S., 2020. QoE-fair DASH video streaming using server-side reinforcement learning. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM) 16 (2s), 1–21.
- Blender Foundation, 2008. Big Buck Bunny. <https://peach.blender.org/>, acessado em: 19 jun. 2026.
- Chromium, 2026. The Chromium Projects. Google, acessado em: 19 jun. 2026. Disponível em: <https://www.chromium.org/>
- Docker Inc., 2026. Docker Engine. Disponível em <https://www.docker.com>, acessado em: 19 jun. 2026.
- freedesktop.org, 2026. GStreamer. Acessado em: 19 jun. 2026. Disponível em: <https://gstreamer.freedesktop.org>
- Fórum do Sistema Brasileiro de TV Digital Terrestre, 2026. TV 3.0 Project. Acessado em: 24 jun. 2026. Disponível em: [https://forumsbtvd.org.br/tv3\\_0/](https://forumsbtvd.org.br/tv3_0/)

- Giladi, A., Levy, N., Griebel, T., Server-Guided Ad insertion in 6th edition of MPEG DASH. In: Proceedings of the 2026 Mile-High Video Conference (MHV '26), 2026. Association for Computing Machinery (ACM), Denver, CO, USA, pp. 59–65.
- Hunter, J. D., 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9 (3), 90–95.
- Ibrahim, M., El Essaili, A., Lohmar, T., D'allonnes, A. R., TV graphics personalization using in-band events. In: Adjunct Proceedings of the 15th European Interactive TV Conference (ACM TVX 2017), In-Programme Personalisation for Broadcast (IPP4B) Workshop, 2017.
- ISO/IEC 23009-1, 2022. Information technology — Dynamic adaptive streaming over HTTP (DASH) — Part 1: Media presentation description and segment formats. Standard ISO/IEC 23009-1, ABNT - Associação Brasileira de Normas Técnicas.
- Mekuria, R., Syed, Y., Hughes, G., Robust SCTE 35 in the OTT workflow. In: In Proceedings of the 2nd Mile-High Video Conference (MHV '23), 2023. Association for Computing Machinery, New York, NY, USA, pp. 27–33.
- Michael Niedermayer, A. K., 2026. FFmpeg. Acessado em: 19 jun. 2026. Disponível em: <https://ffmpeg.org/>
- Moreno, M. F., Pernisa Júnior, C., Barrere, E., Teixeira, S., Montezano, C. T., Shuen, L.-C., Soares Neto, C. d. S., Muchaluat-Saade, D. C., Josué, M. I., dos Santos, J. A. F., Colcher, S., Moares, D. d. S., Omaia, D., de Araújo, T. M. U., de Souza Filho, G. L., 2023. R&D Progress on TV 3.0 Application Coding Layer. *SET International Journal of Broadcast Engineering* ISSN Online: 2446-9432. Disponível em: <https://dx.doi.org/10.18580/setijbe.2023.1>
- Pallets Project, 2026. Flask. <https://flask.palletsprojects.com/>, acessado em: 19 jun. 2026.
- pandas development team, T., fev. 2020. pandas-dev/pandas: Pandas. Acessado em: 19 jun. 2026. Disponível em: <https://doi.org/10.5281/zenodo.3509134>

Perkis, A., 2021. Design and Evaluation of a Second Screen Interactive Digital Media Solution using MPEG-DASH. *Advances in multimedia*.

Qt 6, 2026. Qt for Python. Qt Group, acessado em: 19 jun. 2026. Disponível em: <https://doc.qt.io/qtforpython-6/index.html>

Ramírez, S., 2026. FastAPI. Acessado em: 19 jun. 2026. Disponível em: <https://fastapi.tiangolo.com/>

SCTE 35-2, 2026. Digital Program Insertion Cueing Message Part 2: Event-Based Signaling. Standard SCTE 35-2, ABNT - Associação Brasileira de Normas Técnicas.

Seabold, S., Perktold, J., statsmodels: Econometric and statistical modeling with python. In: 9th Python in Science Conference, 2010.

Shuen, L.-C., Neto, C. S., Moreno, M., Bentivi, Z., Gonçalves, J. V., Costa, I. V., Avaliação da Experiência do Telespectador na Transição para a TV Orientada a Aplicativos. In: Anais Estendidos do XXX Simpósio Brasileiro de Sistemas Multimídia e Web, 2024. SBC, Porto Alegre, RS, Brasil, pp. 261–266. Disponível em: [https://sol.sbc.org.br/index.php/webmedia\\_estendido/article/view/30504](https://sol.sbc.org.br/index.php/webmedia_estendido/article/view/30504)