# RV-MTV: Framework para Interação Multimodal com Aplicações de Realidade Virtual em TV Digital e Dispositivos Móveis

Caio Cesar Viel
Universidade Federal de São Carlos
caio_viel@dc.ufscar.br

Erick Lazaro Melo
Universidade Federal de São Carlos
erick_melo@dc.ufscar.br

Luis Carlos Trevelin
Universidade Federal de São Carlos
trevelin@dc.ufscar.br

Cesar A. C. Teixeira
Universidade Federal de São Carlos
cesar@dc.ufscar.br

## ABSTRACT
The Interactive Digital TV enables the creation and dissemination of interesting multimedia applications. It is natural to ask whether Interactive Animation and Virtual Reality applications are contemplated. Initially one can say no. Due to the high computational power required by these applications, it is not plausible run them at a set-top box or embedded processor on a TV set. This same question can be asked when mobile devices are taken into account, since they have similar computational constraints to Digital TV. This paper presents a strategy that enables the use of Interactive Digital TV and Mobile Devices to provide multimodal interactions with some classes of Virtual Reality applications.

## RESUMO
A TV Digital Interativa propicia a criação e disseminação de aplicações multimídia interessantes. É natural perguntar se aplicações de Animação Interativa e de Realidade Virtual são contempladas. Inicialmente pode-se dizer que não, já que devido ao alto poder computacional exigido por essas aplicações, não é plausível executá-las em um *set-top box*ou processador embarcado em aparelhos de TV. Essa mesma pergunta pode ser feita quando dispositivos móveis são levados em consideração, já que possuem restrições computacionais similares à TV Digital. Este trabalho apresenta uma estratégia que possibilita utilizar ambientes de TV Digital Interativa e Dispositivos Móveis para proporcionar interação multimodal com algumas classes de aplicações de Realidade Virtual.

## Categories and Subject Descriptors
H.5.1 [**Multimedia Information Systems**]: Communications Applications – *Artificial, augmented, and virtual realities, Video.*

## General Terms
Measurement, Performance, Experimentation, Human Factors.

## Keywords
Digital Interactive TV, IPTV, Video Encoding, Virtual Reality, Multimodal Interaction, Framework.

## 1. INTRODUÇÃO
Aplicações de realidade virtual (RV) permitem criar uma interface entre usuários e sistemas computacionais que busca promover sensação de realidade para o usuário. Isso é possível graças ao uso de técnicas de renderização de imagem e suporte computacional com grande capacidade de processamento. Plataformas como TV Digital Interativa (TVDi) e dispositivos móveis (*smartphones, tablets*) têm-se popularizado em ambientes domésticos. Observa-se também o interesse crescente dos usuários em interagir com conteúdos multimídia ou de múltiplas mídias utilizando diferentes classes de dispositivos. Esse cenário de convergência de mídias abre oportunidades para a criação e disseminação de aplicações bastante interessantes. A viabilidade de aplicações de RV nesse cenário requer o desenvolvimento de tecnologias adequadas e uma mudança no paradigma das aplicações (*Figura 1*). Novos requisitos precisam ser considerados no projeto de uma aplicação de RV. Na Figura 1 – *esquerda* são listadas as etapas de processamento em uma aplicação de RV típica. As aplicações são baseadas em um único computador e tela, com dispositivos de interação locais. Já aplicações de realidade virtual baseada em *cloud-computing* e com suporte a múltiplos dispositivos (Figura 1 – *direita*) podem ser distribuídas em um conjunto de computadores trabalhando de forma conjunta para proporcionar ao usuário uma experiência satisfatória em quaisquer plataformas (*hardware* e *software*). Nesse cenário pode ser considerado o uso de mais de um tipo de dispositivo de interação simultaneamente, além de mais de um dispositivo de exibição, como múltiplas telas em um ambiente de multiprojeção e dispositivos móveis. O conteúdo, neste caso, é adaptado às características do dispositivo.

Uma possível estratégia para possibilitar a interação dos usuários de sistema de TVDi com aplicações de RV sem custos com processamento extra local – a adotada no projeto XPTA.LAB[1] e descrita neste trabalho – é executar as aplicações remotamente, em máquinas com maior poder computacional, e enviar aos *Set-Top Boxes* (STB) ou processadores integrados aos aparelhos de TV (que neste trabalho serão referenciados apenas como STB) as imagens e sons gerados pela aplicação de RV, através de fluxos de vídeo e áudio codificados. Os STBs podem

WebMedia'11: Proceedings of the 17th Brazilian Symposium on Multimedia and the Web. Full Papers.
October 3 -6, 2011, Florianópolis, SC, Brazil.
ISSN 2175-9642.
SBC - Brazilian Computer Society

[1] O Projeto "*Virtualidade Imersiva e Interatividade Baseada em CloudComputing*", está inserido no Programa Laboratórios de Experimentação e Pesquisa em Tecnologias Audiovisuais – XPTA.LAB, promovido pelo Ministério da Cultura, através da Secretaria do Audiovisual, com apoio da Sociedade Amigos da Cinemateca.

receber os fluxos via radiodifusão terrestre, para aplicações colaborativas massivas, e enviar, pelo canal de retorno ou de interatividade, as interações realizadas pelos respectivos usuários. Para aplicações mais individualizadas, o STB pode usar seu canal de interatividade para receber os fluxos. Essa solução pode ser adequada também a sistemas de *Internet Protocolo TV* (IPTV)

para proporcionar a interação com aplicações de RV, em serviços semelhantes à *VideoonDemand* (VoD).

Dispositivos móveis têm características similares às de um STB típico: baixa capacidade de processamento, de memória reduzida e de comunicação de dados. Respeitando-se diferenças tecnológicas, é plausível aplicar a dispositivos móveis técnicas semelhantes às aplicadas para o ambiente de TVDi para permitir a interação com aplicações de RV.

Sistemas de interação multimodal são capazes de combinar dois ou mais tipos de interações do usuário de maneira coordenada [1], o que pode tornar mais naturais as interações de usuários com aplicações, contribuindo com a sensação de realidade que a RV busca atingir. Com a recente popularização de dispositivos de consumo que oferecem múltiplas formas de interação, torna-se possível o desenvolvimento de aplicações com interações multimodais, que podem tornar as aplicações mais ubíquas e fáceis de serem utilizadas. Uma *ApplicationProgram Interface* (API) de alto nível, que forneça facilidades para construção de aplicações que utilizem interações multimodais provenientes de dispositivos diversos, pode ser útil.

Neste trabalho é apresentada proposta de um *framework* que permite a usuários de sistemas de TVDi e dispositivos móveis interagir com aplicações de RV. Esse *framework* é capaz de comunicar-se com diversos tipos de dispositivos via tecnologias como *Universal Plugand Play* (UPnP) e *Zero Configuration* (Zeroconf) ou via *drivers* nativos. Através dele é possível construir aplicações que recebem eventos multimodais de dispositivos heterogêneos de forma simples e transparente.

A validação da proposta foi realizada com uma implementação do *framework* em linguagem C++, com a construção de uma aplicação prova de conceito e a realização de experimentos de avaliação.

A *seção 2* do trabalho apresenta um levantamento bibliográfico de trabalhos relacionados. Na *seção 3* são discutidas possíveis estratégias que foram consideradas na proposta do framework. A arquitetura do *framework* é apresentada na *seção 4*. A prova de conceito que valida o *framework* é apresentada na *seção 5*. A *seção 6* finaliza o texto com as conclusões do trabalho.

## 2. TRABALHOS RELACIONADOS

Jurgelionis*et al* [2] propõem um cenário de *pervasivegamming*baseadoem um servidor de processamento local, responsável pela execução dos jogos e pela geração do vídeo em vários formatos e tamanhos, capaz de suportar vários dispositivos com características e capacidades diferentes, como STB, *smartphones* e computadores de baixo desempenho. Além dos *streamings* de áudio e vídeo, também é possível gerar *streams* de imagens pré-renderizadas para dispositivos com maior poder computacional que são responsáveis pela renderização final das imagens. A diferença desse trabalho para o aqui apresentado consiste no fato de que em [2] é necessário uma máquina local para realizar o processamento. Com o framework aqui proposto, esse processamento pode ser realizado tanto em uma máquina na residência do usuário como também de forma distribuída em máquinas remotas, localizadas em uma nuvem computacional. Essa segunda abordagem poderia diminuir o custo para o usuário final, já que ele não precisaria adquirir uma máquina com a capacidade de processamento necessária para executar aplicações de RV, e não restringiria a sua mobilidade.

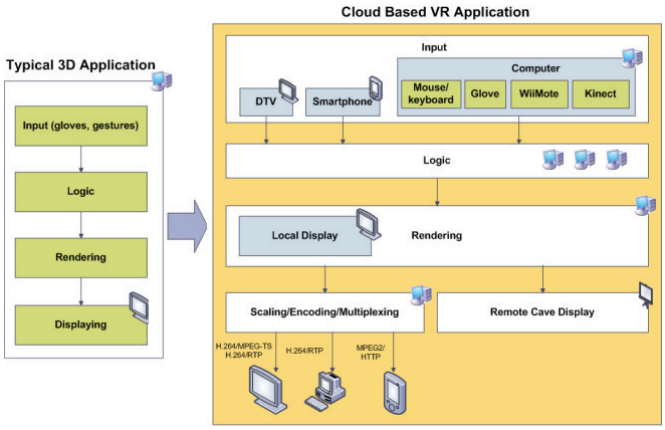


**Figura 1. Paradigma de aplicações de Realidade Virtual**

Existem vários serviços que utilizam uma abordagem semelhante, em que aplicações que demandam muito processamento ou *hardware* dedicado são executadas em nuvens computacionais e promovem a usuários remotos a sensação de estarem interagindo com uma aplicação local. O serviço de *CloudGamming*OnLive [3] permite que usuários remotos participem de jogos eletrônicos em seus servidores, interagindo com eles a partir de seus computadores pessoais. Para reduzir a latência da aplicação são enviadas às máquinas dos usuários imagens pré-renderizadas. Entretanto esse processo ainda é inviável para STB típicos ou outros dispositivos de baixo poder computacional.

Aplicações de RV têm como restrição baixo tempo de resposta para as interações dos usuários. A estratégia aqui apresentada tende a aumentar o tempo de resposta das aplicações. Parte desse aumento é devido à latência causada pela codificação de vídeo. Vários estudos já foram feitos para tentar minimizar a latência dos processos de codificação de vídeos. O estudo de Holub*et al* [4] leva em consideração vídeos em alta definição, que são ainda mais custosos para serem codificados. Tem-se também o aumento de tempo de resposta induzido pela execução de aplicações na nuvem, que pode ser crucial para alguns tipos de aplicações. Alguns estudos abordam este problema, como o de Barker e Shenoy[5], que realiza uma análise empírica da performance de aplicações colaborativas nas nuvens.

Guimarães *et al* [6] esboçam uma arquitetura para a difusão de aplicações de RV imersiva em dispositivos com baixa capacidade de processamento, utilizando servidores em nuvens computacionais para realizar o processamento. No entanto, os autores não se aprofundaram nas técnicas utilizadas para permitir que esses dispositivos possam interagir com as aplicações de RV.

Considerando a televisão como um mero dispositivo de saída visual, Robertson *et al* [7] apresentam um sistema em que uma aplicação executando em um PDA pode fazer uso da televisão como uma tela secundária, e assim mostrar mais informações visuais para o usuário. Cesar *et al* [8] exploram a idéia da tela

secundária na direção contrária. Em seu trabalho, os autores propõem três categorias de uso de um PDA que atua como uma tela secundária com capacidades de apresentação e interação: (1) permite ao usuário navegar pelos conteúdos e selecionar um determinado conteúdo para ser apresentado na televisão; (2) permite que o usuário continue assistindo, utilizando a tela do PDA, o conteúdo sendo exibido na TV de sua sala de estar, enquanto deixa o ambiente; (3) permite que o usuário faça recomendações de conteúdo para amigos e familiares, enviando trechos do conteúdo acompanhados de anotações ou conteúdos extras.

Seguindo na mesma linha, mas voltando-se para o contexto da televisão digital brasileira, Soares *et al* [9] apresentam o modelo de controle hierárquico usado pelo Ginga-NCL para suporte a múltiplos dispositivos de exibição.

Os cenários apresentados por esses trabalhos, apesar de fazerem uso de dispositivos secundários, exploram principalmente a tela adicional que os dispositivos oferecem. Ao contrario da proposta aqui apresentada, o objetivo principal deles não é oferecer às aplicações de TVDi a possibilidade de receberem eventos multimodais gerados pelo dispositivo adicionail. Os cenários apresentados por Cesar *et al* [8] e Soares *et al* [9] permitem que o dispositivo secundário ofereça entrada para a aplicação da televisão, mas essa não é nada mais do que um simples evento de pressionamento de tecla como o gerado pelo controle remoto tradicional.

# 3. ESTRATÉGIAS

Aplicações de RV utilizam-se de interfaces entre usuários e sistemas computacionais que procuram recriar ao máximo a sensação de realidade para o usuário. Isso é possível graças ao uso de técnicas de renderização de imagem e de suporte computacional com grande capacidade de processamento. Devido aos custos associados a esse suporte computacional, aplicações de RV estão geralmente limitadas a ambientes de pesquisa ou processadores pessoais de boa configuração de memória, processamento e recursos.

Uma estratégia para propiciar aos usuários de TVDi a possibilidade de interação com aplicações de RV seria utilizar um STB com recursos equivalentes a de computadores pessoais, como descrito no parágrafo anterior. Esse modelo não é plausível, pois STB geralmente são máquinas projetadas para possuir custo baixo e muitas vezes apresentam-se embarcados em aparelhos de TV, não sendo compatíveis com o processamento necessário para executar aplicações de RV.

Uma segunda estratégia consiste em utilizar o processamento de uma máquina externa que, conectada ao STB, possibilite a distribuição do processamento. Essa máquina poderia estar localizada na casa do usuário, ou estar em algum ponto da *internet*, como um servidor ou aglomerado de servidores, comunicando-se ao STB através de seu canal de interatividade. Com a máquina localizada na internet seria possível uma redução de custos para o usuário final, pois ele não precisaria adquirir equipamentos adicionais. Além disso, cria-se um ambiente propício para o desenvolvimento de aplicações de RV colaborativas, em que dois ou mais usuários poderiam interagir com o mesmo ambiente virtual simultaneamente.

Com a aplicação executado remotamente, o STB e a TV poderiam atuar apenas como interface física de saída, de forma semelhante a monitores em sistemas *desktop*. O monitor acessa a memória de vídeo e a partir dela forma a imagem que deve ser exibida. Nesse caso é necessário enviar periodicamente ao STB cópias do *buffer* onde a imagem gerada pela aplicação de RV está armazenada. Para imagens no formato *Full HD* (1920 x 1080 pixels) com 30 quadros por segundo, seria necessária uma largura de banda entre o servidor de processamento e o STB de aproximadamente 1.9 Gbit/s, impraticável com a infraestrutura de rede existente atualmente na *Internet* ou com a banda disponível em canais de sistemas de radiodifusão terrestre. Logo, tentar utilizar diretamente a imagem gerada no servidor de processamento remoto é inviável com a tecnologia atual.

Uma terceira estratégia poderia se basear em técnicas de visualização remota, utilizadas em sistemas computacionais para permitir o acesso remoto de interfaces gráficas ou *cloudgamming*. Essas técnicas consistem em gerar imagens pré-rendenderizadas na máquina remota e enviá-las para a máquina local. As imagens pré-renderizadas contém informações que permitem à máquina local finalizar a renderização e obter a imagem original exibida pela máquina remota. É possível enviar as imagens pré-renderizadas geradas pela aplicação de RV nos servidores de processamento remoto até os STB, pois a largura de banda necessária para enviar imagens pré-renderizadas é significativamente menor do que a necessária para as imagens prontas. Apesar da capacidade de processamento necessária para reproduzir as imagens originais a partir das imagens pré-renderizadas ser inferior à necessária para executar aplicações de RV, ela ainda é maior do que a disponível na maioria dos STB, inviabilizando também esta estratégia.

A quarta estratégia consiste em tratar a sequência de imagens produzidas no servidor de processamento pela aplicação de RV como um vídeo digital. Podem-se utilizar algoritmos de compressão de vídeo, como o MPEG-2 *Video*[10] ou o H.264 [11], para codificar esse vídeo e enviá-lo ao STB ou *smartphone* por radiodifusão terrestre, satélite, cabo, ou pelo canal de interatividade. Torna-se viável assim propiciar aos usuários de TVDi e de dispositivos móveis a interação com aplicações de RV sem a necessidade de taxas de comunicação de dados exorbitantes, pois os algoritmos de compressão de vídeo conseguem gerar fluxos a taxas aceitáveis. Os STB comerciais possuem decodificação de vídeo em *hardware*, o que contribui com a estratégia. É com base nessa estratégia que este trabalho foi desenvolvido. A *Figura 2* ilustra a estratégia sendo utilizada para permitir a usuários de sistema de TVDi a possibilidade de interação com aplicações de RV individualizadas. Observa-se uma semelhança com serviços de IPTV, porém ao invés de distribuir programas de TV ou VoD por uma rede IP, envia-se vídeo e áudio gerados a partir de uma aplicação de RV. A *Figura 3* apresenta outra possibilidade para distribuição de vídeo e áudio utilizando radiodifusão terrestre, mais adequada para aplicações de RV colaborativas massivas de pouca individualidade, em que todos os participantes enxergam uma mesma imagem, como pode acontecer, por exemplo, em uma gincana virtual.

Nesta estratégia, quando o usuário deseja interagir com uma aplicação de RV, são alocados um ou mais servidores de execução de aplicações de RV responsáveis pelo processamento necessário para a execução da aplicação. Parte da aplicação fica localizada no STB ou dispositivo móvel do cliente e é responsável pela captura das interações do usuário e realizar a comunicação com o restante da aplicação, localizada nos servidores de execução. O

codificador de áudio e vídeo utiliza algoritmos de compressão para codificar vídeo e áudio gerados pela aplicação de RV, produzindo fluxos de vídeo e áudio comprimidos, que são enviados ao STB ou dispositivo móvel através de um rede de alta velocidade ou via radiodifusão terrestre, pelo ISDB-TB [12, 13, 14] por exemplo. O codificador de áudio e vídeo pode estar localizado na mesma máquina responsável pela execução da aplicação de RV, desde que ela possua capacidade de processamento suficiente para executar a aplicação de RV e realizar a codificação.

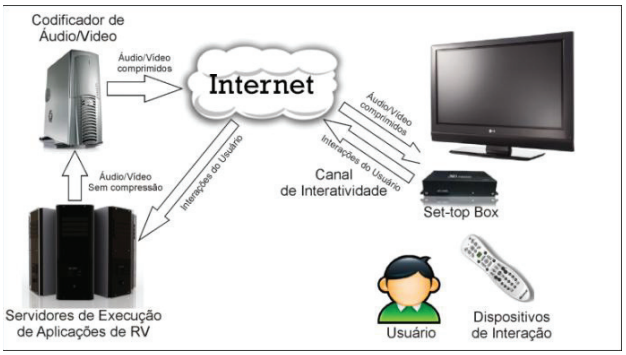


**Figura 2. Estratégia de Streaming de Vídeo: Pela Internet**

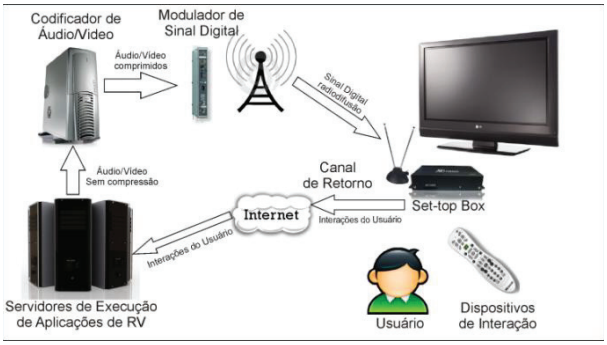


**Figura 3. Estratégia de Streaming de Vídeo: Via Radiodifusão**

O STB ou dispositivo móvel recebe os fluxos de áudio e vídeo, decodifica-os e os apresenta ao usuário. O usuário pode interagir com a aplicação de RV utilizando o controle remoto da TV, mas também com outros dispositivos como controle remoto de jogos eletrônicos, *tablets*, telefones celulares, etc. As interações do usuário são enviadas através do canal de interatividade do STB ou via rede de dados dos dispositivos móveis até a aplicação de RV, localizada nos servidores de execução de aplicações. A aplicação processa as interações do usuário, gerando imagens e sons digitais como resposta, que são retornadas ao usuário via *streaming*.

# 4. ARQUITETURA DO FRAMEWORK

A estratégia adotada para a difusão de aplicações de RV em TVDi e dispositivos móveis apresenta dois grandes desafios:

- Permitir que uma mesma aplicação projetada para um ambiente imersivo (como uma caverna) possa ser utilizada por usuários fora desses ambientes; e

- Possibilitar que os usuários interajam com as aplicações utilizando dispositivos comuns de uso pessoal ou doméstico. Essa interação pode ser feita utilizando um ou mais dispositivos simultaneamente.

Este trabalho busca solucionar esses desafios através da proposta de um *framework* (*Figura 4*) que atenda tanto aos interesses do usuário quanto do desenvolvedor de *software*.

Com relação aos desenvolvedores, o *framework* apresenta um ferramental que permite a construção de aplicações utilizando primitivas de mais alto nível, abstraindo a complexidade dos processos de codificação e difusão de vídeo (API de *Streaming*). Ao usuário o *framework* oferece um conjunto de recursos que permite a integração da aplicação com diversos dispositivos locais ou remotos (API de Interação Multimodal).

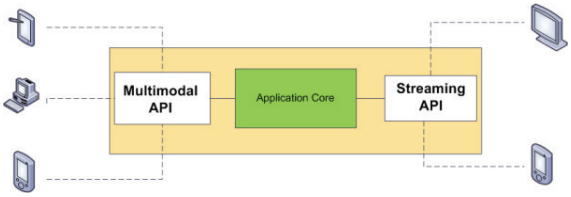


**Figura 4. Framework para Aplicações de RV**

## 4.1 API de *Streaming*

A API de *Streaming* permite aos desenvolvedores criar aplicações capazes de codificar áudio e vídeo e enviar fluxos via rede de dados ou gerar arquivos com as mídias codificadas.

A *Figura 5* apresenta a arquitetura do Componente de *Streaming*.

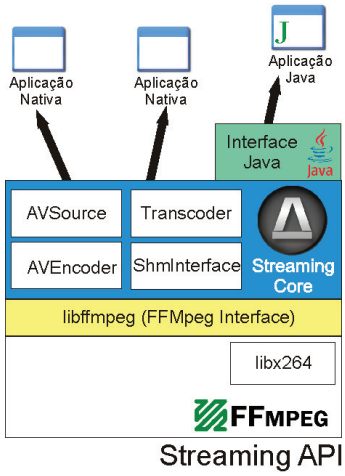


**Figura 5. Arquitetura do componente de *Streaming***

Ela permite à aplicação escolher a fonte do áudio ou vídeo que será utilizada. Essas fontes podem ser arquivos de áudio e vídeo, dispositivos de captura como uma *webcam* ou a própria tela do computador. A API também permite que a aplicação defina parâmetros para essas fontes, como por exemplo o número de quadros por segundo gerados por um dispositivo.

É possível especializar o processo de codificação através de uma série de parâmetros, como selecionar o codificador que será utilizado, a taxa de bits, número de quadros por segundo e etc. Além disso, a API suporta a utilização de arquivos de configurações, que permitem definir o processo de codificação de forma mais simples e reutilizável.

A API realiza o *streaming* dos vídeos e fluxos codificados através do protocolo RTP/RTSP [15, 16]. Também é possível encapsular os fluxos em um Mpeg-TS [17] e enviá-lo para o destino via protocolo UDP.

Para ilustrar a utilização da API do componente, apresenta-se no *Quadro 1* uma aplicação desenvolvida em C++ utilizando a API de *Streaming*. Essa aplicação utiliza o terminal X11 para gerar umfluxo de vídeo utilizando o codificador de vídeo H.264 e realiza o streaming desse vídeo através do protocolo RTP.

**Quadro 1. Utilização da API de *Streaming***

```
int main() {
    AVSource* source = new X11Terminal(25);

    AVEncoder* encoder = new AVEncoder(source);
    encoder->setVideoCodec(H264);
    encoder->setVideoPreset("ultrafast");
    encoder->setPropertyValue("crf", "22");

    Streaming* streammer =
            new RTPStream("127.0.0.1", 5004);
    streammer->addStream(encoder);
    streamer->start();
    streamer->waitFinishing();
    return 0;
}
```

## 4.2 API de Interação Multimodal

Esta API possui uma série de funcionalidades que permite aos desenvolvedores utilizar interações multimodais em suas aplicações de uma forma simples e transparente. Escondendo toda a complexidade em lidar com os vários dispositivos heterogêneos que podem existir no ambiente.

Para os dispositivos, a API oferece diferentes protocolos pelos quais eles podem ser encontrados e conectar-se à API. Propõe-se um serviço UPnP [18] e um serviço Zeroconf [19] para descoberta de dispositivos capazes de realizar interações multimodais. Através desses serviços a API pode procurar por dispositivos capazes de realizar interações multimodais. Os dispositivos, uma vez conectados, podem enviar à API as interações multimodais realizadas na forma de eventos. Também propõe-se um servidor TCP, capaz de se conectar aos dispositivos e receber as interações multimodais. Esse servidor não é configuração-zero, portanto é necessário que o usuário configure explicitamente seus dispositivos para que se conectem através do TCP. Isso aumenta o número de dispositivos com os quais a API pode-se comunicar, pois mesmo que um dispositivo não implemente os protocolos Zeroconf ou a arquitetura UPnP, ele poderá se comunicar diretamente via TCP – o que exige o desenvolvimento de um cliente TCP para o dispositivo.

A API também oferece *drivers* específicos para alguns tipos de dispositivos mais simples, como *joypads* de consoles de videogame. Esses dispositivos não são capazes de se conectar à rede, por isso a API se comunica com eles de outras formas, como interface *Bluetooth* ou *Universal Serial Bus* (USB).

A API não relaciona diretamente eventos e dispositivos, podendo um mesmo tipo de evento ser enviado por dispositivos diferentes, e um mesmo dispositivo enviar vários tipos de eventos. O teclado pode ser considerado um dispositivo, bem como um *smartphone* ou o *joypad*. Cada dispositivo é capaz de realizar um ou mais tipos de interações, gerando eventos correspondentes. Por exemplo, um teclado só pode gerar eventos do tipo tecla, porém um *smartphone* pode gerar eventos de aceleração, caneta, reconhecimento de voz e até mesmo tecla – através de um teclado virtual do dispositivo, por exemplo.

Para as aplicações, a API permite ao desenvolvedor determinar quais tipos de eventos ele deseja que a aplicação receba. Isso é feito registrando a aplicação como uma observadora de eventos de determinados tipos. Sempre que um dispositivo enviar um evento daquele tipo, a API irá notificar as aplicações observadoras. Os eventos são enviados para as aplicações através de funções de *callback*.

A *Figura 6* representa a arquitetura do componente. O componente pode receber as interações dos dispositivos através da rede ou via *drivers* nativos. As interações recebidas pela rede são envidas em documento formato XML, por isso existe a necessidade de um *parser* que converte o documento XML em eventos representados por objetos na memória. O *Multimodal Event Manager* recebe todos os eventos e os encaminha para as aplicações que os desejam receber.

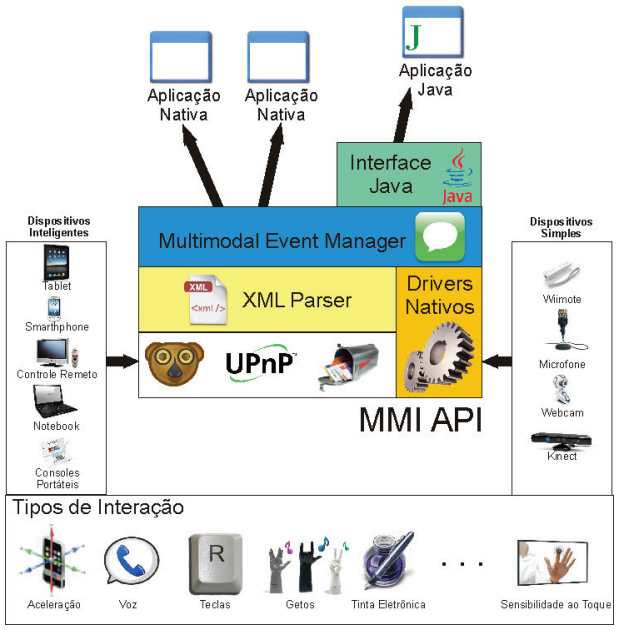


**Figura 6. Arquiteturado componente de Interação Multimodal**

Os dispositivos inteligentes são aqueles capazes de se comunicar através de redes de dados, utilizando a arquitetura UPnP, os protocolos Zeroconf ou mesmo *sockets* TCP.

Os dispositivos simples não são capazes de se comunicar via rede de dados. Em geral são periféricos ou acessórios de algum sistema computacional, como *joypads* de consoles de videogame, *webcams*, microfone, etc. Para que a API seja capaz de se comunicar com esses dispositivos, é necessária a implementação de um *driver* nativo. Esse *driver* captura as interações realizadas com os dispositivos e gera eventos que representam a interação e os envia para o *Multimodal Event Manager*.

Para ilustrar a utilização da API, apresenta-se uma aplicação em C++ que recebe eventos do tipo tecla, enviados por quaisquer dispositivos, e imprime na tela a tecla acionada.

Inicialmente cria-se uma classe que implementa a interface *MMIEventListener* (*Quadro 2*). A classe deve implementar o

método *receiveEvent*. É através desse método que a API irá informar à aplicação o recebimento de eventos multimodais.

Na função *Main* (Quadro 3*)* um objeto da classe *EvListener* é registrado como um observador de eventos do tipo "*key*". O programa então espera até que eventos do tipo "*key*" sejam enviados para a API, que irá chamar a função de *callback* do objeto *EvListener* registrado.

**Quadro2.Classe*EvListener***

```
class EvListener : public MMIEventListener {
public:
    bool receiveEvent(MMIEvent* event);
};

bool EvListener::receiveEvent(MMIEvent* e) {
    cout << "Device " << e->getDeviceId()
         << " eventType: " << e->getEventType()
         << endl;

    if (t == "key") {
        KeyEvent* k = (KeyEvent*) e;
        cout << "Tecla Presscionada: "
             << k->getKeyId() << endl;
    }
}
```

**Quadro3.Função*Main***

```
int main() {
    EvListener* evListener = new EvListener();
    set<string>* evType = new set<string>();
    evType->insert("key");

    MMIManager::getInstance()->addEventListener(
                evListener, evType);

    while (1) {
        usleep(1000000);
    }
    return 0;
}
```

# 5. PROVA DE CONCEITO

No decorrer do desenvolvimento do *framework* foram realizados diversos testes para avaliação e escolha entre diferentes variações de técnicas, algoritmos de codificação de vídeo e métodos para transmissão dos eventos multimodais. Uma prova de conceito, descrita e analisada a seguir, também foi implementada para fins de validação do fremework.

## 5.1 Descrição

Foi desenvolvida uma aplicação utilizando-se a linguagem C++, a API para interação Multimodal e a API d *Streaming*. Com recursos do OpenGL [20], modelou-se um boneco semelhante a um robô e uma sala virtual tridimensional. Através do teclado ou de interações multimodais é possível fazer o robô caminhar pela sala sem ultrapassar os limites definidos por suas paredes.

A *Figura 7* mostra algumas capturas de tela da aplicação em execução.

Pode-se controlar o boneco com três dispositivos diferentes: *Wiimote*, *iPad* e controle-remoto de TV.

Sempre que um novo quadro é gerado pelo OpenGL, a aplicação utiliza a API de *Streaming* para salvar esse novo quadro na área de memória compartilhada existente na API. A aplicação instancia um processo de transcodificação do vídeo, utilizando o dispositivo virtual definido pela área de memória compartilhada como fonte do vídeo. Para codificação é utilizado o codificador H.264. O fluxo de vídeo gerado é encapsulado em um pacote Mpeg-TS e transmitido via protocolo UDP para o servidor de mídia *Wowza Media Server* [21]. A transmissão também pode acontecer diretamente entre os dispositivos, independentemente de um servidor de mídia.

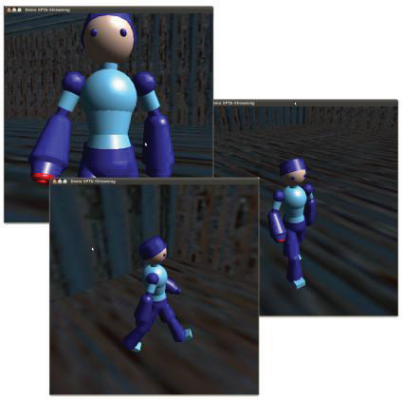

**Figura 7. Robô em uma sala virtual**

Para o controle através de iPad foi necessário desenvolver um código específico para as interações. Além de capturar as interações do usuário e envia-las para a aplicação através da API de Interação Multimodal, o código implementado também permite exibir o vídeo gerado pela aplicação através do servidor de mídia (*Figura 8*). Dessa forma é possível utilizar um iPad para interagir com uma aplicação em execução não só em um Desktop local como também em equipamento remoto.

**Figura 8. Captura de tela da aplicação em um tablet**

## 5.2 Análise

Foram analisados alguns índices de desempenho, da aplicação desenvolvida com o *framework,* em vários cenários diferentes de codificação de vídeo (API de *Streaming*). Os principais índices analisados foram utilização de CPU e quantidade de memória alocada. Também foi medida a taxa de dados gerada e observada a latência entre a geração e exibição do vídeo em cada cenário.

Foi utilizado um computador com processador Intel *Core 2 Duo* E4500, com 2.20GHz com e 2GB de memória DDR2. Nele foi executada a aplicação prova de conceito. Outra máquina, ligada por uma rede *Gigabit ethernet* com a primeira, foi utilizada para receber e apresentar o vídeo.

No primeiro cenário utilizou-se o codificador de vídeo MPEG-2 *Video* com seus parâmetros padrões. No segundo cenário foi utilizado o H.264 em sua configuração de codificação *ultrafast*. O

terceiro cenário também utiliza o H.264, porém na configuração *default*. No quarto e último cenário, o codificador H.264 foi utilizado em sua configuração *veryslow*. Essas configurações dispõem de parâmetros que permitem definir a forma como o codificador H.264 comprime os vídeos. Com a configuração *ultrafast*, o codificador mantém um buffer de quadros pequeno, com o objetivo de minimizar a latência do vídeo gerado. Já com a configuração *veryslow*, o codificador mantém um buffer de quadros grande, buscando minimizar a taxa de dados do vídeo gerado, aumentando, porém, a latência do vídeo.

A *Figura 9* mostra o gráfico que relaciona a memória alocada em cada um dos cenários analisados. Já a *Figura 10*exibe o gráfico que relaciona a taxa de dados gerados pelos cenários analisados, enquanto o gráfico da *Figura 11* exibe a utilização de CPU em cada cenário. O *Quadro 4* mostra o valor da latência observada nos cenários.

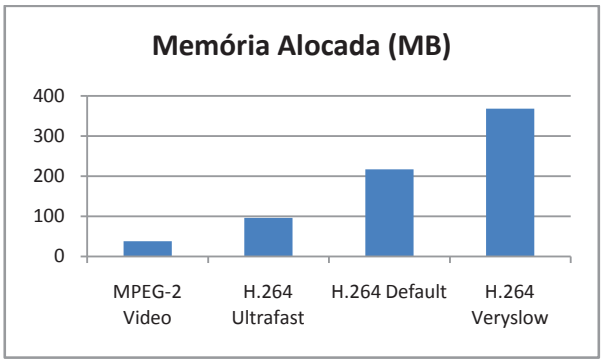


**Figura 9. Memória Alocada pela API de Streaming**

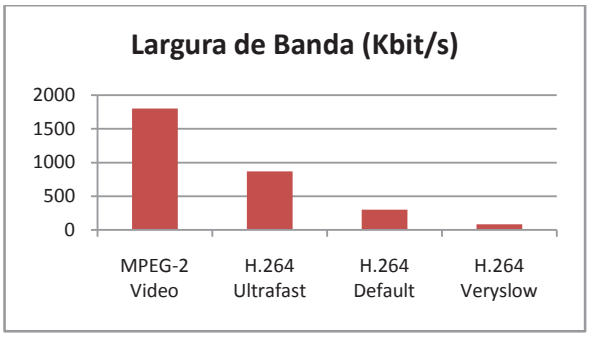


**Figura 10. Taxa de Dados dos Vídeos Gerados**

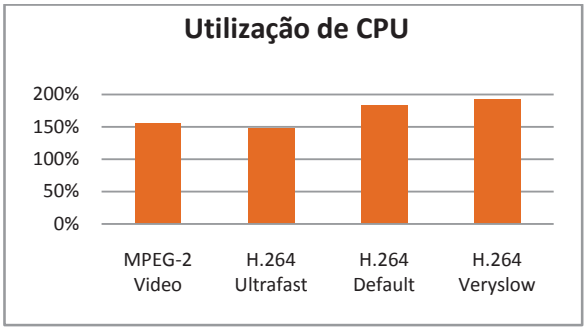


**Figura 11. Utilização de CPU pela API de Streaming**

Observou-se durante o experimento que as configurações do codificador de vídeo influenciam significativamente na latência do vídeo. A configuração H.264 *very-slow*produz um vídeo a taxas muito baixas, mas a um custo elevado de processamento e latência. Já a utilização do codificador MPEG-2 Video possui uma latência muito baixa ao custo de uma elevada taxa de bits. A codificação H.264 *Ultrafast* foi a que melhor se adequou aos resultados desejados. A latência observada foi inferior a 2 segundos e produziu um vídeo com uma taxa de dados relativamente baixa.

**Quadro 4. Latência do observada**

| **Codificador utilizado** | MPEG-2Video | H.264 Ultrafast | H.264 Default | H.264 Veryslow |
|---|---|---|---|---|
| **Latência máxima observada** | 1s | 2s | 12s | 40s |

Um mecanismo de *holdon* foi introduzido para se evitar erros, no resultado apresentado ao usuário, decorrentes da latência do processo. A apresentação é congelada (ou enriquecida com algum efeito dissimulador) enquanto o servidor processa a interação do usuário, que irá promover mudanças no curso da apresentação, gera novo trecho de vídeo com a apresentação correta e envia ao usuário. Nesse interim, o processo cliente descarta o resultado (vídeo/áudio que está recebendo) que produziria erros na apresentação.

## 6. CONCLUSÕES

Neste trabalho foi apresentado um *framework* que permite a interação de aplicações de RV ou animações interativas com usuários de TVDi e de dispositivos móveis através de uma estratégia de *streaming* de vídeo. Essa estratégia consiste em executar a aplicação em servidores de execução com grande poder computacional, capazes de executar as aplicações e de enviar, via streaming de vídeo, as imagens geradas pela aplicação para os STB e dispositivos móveis. Os usuários realizam interações multimodais, que são transferidas para a aplicação via rede de dados. A estratégia foi implementada em um framework que possui duas APIs distintas, uma que fornece facilidades multimodais e outra para realizar a codificação, multiplexação e streaming de vídeo. Com a API para Interação Multimodal é possível construir aplicações interessantes capazes de receber interações multimodais de vários dispositivos heterogêneos, abstraindo particularidades dos dispositivos. Já a API de Streaming facilita o desenvolvimento de aplicações que manipulam áudio e vídeo na forma de *streams*.

Comparando o desempenho dos vários cenários analisados, pode-se concluir que em situações em que não existam restrições quanto à largura de banda, o codificador MPEG-2 *Video* é o mais indicado, pois possui a menor latência de vídeo, ao custo de uma taxa de dados elevada. Já em situações em que existem restrições de banda, utilizar o MPEG-2 pode ser inviável. Devido à necessidade de se utilizar um *buffer* de quadros para codificar vídeos digitais com o H.264, somente configurações com *buffers* pequenos são adequadas para as aplicações de RV que exijem muita interação do usuário, pois se o atraso na geração do vídeo for muito grande, a experiência do usuário poderá ser ruim. Para aplicações contemplativas, em que se tem poucas interações do usuário, diminuindo a importância da latência do vídeo, as configurações com *buffers* maiores do H.264 podem ser vantajosas, já que geram vídeos com taxas de dados muito baixas. Como o processo de codificação de vídeo exige grande esforço computacional, a codificação de vídeo de forma distribuída pode ser um recurso útil.

Em aplicações de RV massivas, em que o resultado a se apresentar, decorrente da integração das inúmeras interações individuais, seja o mesmo para todos, a solução de transmitir os fluxos de áudio e vídeo via radiodifusão terrestre parece ser bastante promissora. Entretanto, para aplicações de RV personalizadas (conteúdo áudio visual exclusivo), o modelo de transmissão através de uma rede IP torna-se necessário.

Apesar de terem sido observadas latências superiores a um segundo nos experimentos, o mecanismo de *holdon*introduzido viabiliza todas as classe de aplicações em que não sejam requeridos *feedbacks* instantâneos (como em certos videogames de ação). Aplicações de passeios a museus virtuais, ou de exploração a objetos visuais, por exemplo, seriam plenamente suportadas.

Os resultados obtidos no trabalho serão incorporados a projetos de desenvolvimento e inovação do laboratório Lince da UFSCar voltados ao apoio do ensino.

## 7. AGRADECIMENTOS

Agradecemos à RNP e ao Ministério da Cultura pelo apoio financeiro dado ao grupo de pesquisa. Este trabalho foi desenvolvido no escopo do projeto XPTA.Lab. Agradecemos à CAPES pelo suporte aos alunos Caio Viel e Erick Melo e ao CNPq pelo suporte ao pesquisador Cesar Teixeira. Agradecemos à Fapesp pelo apoio à participação no evento.

## 8. REFERÊNCIAS

[1] Bruno Dumas, Denis Lalanne, and Sharon Oviatt. 2009. Multimodal Interfaces: A Survey of Principles, Models and Frameworks. In *Human Machine Interaction*, Denis Lalanne and JörgKohlas (Eds.). Lecture Notes In Computer Science, Vol. 5440. Springer-Verlag, Berlin, Heidelberg 3-26. DOI=10.1007/978-3-642-00437-7_1 http://dx.doi.org/10.1007/978-3-642-00437-7_1

[2] Jurgelionis, A., Fechteler, P., Eisert, P., Bellotti, F., David, H., Laulajainen, J. P., Carmichael, R., Poulopoulos, V., Laikari, A., Perälä, P., De Gloria, A., and Bouras, C. 2009. *Platform for distributed 3D gaming. Int. J. Comput*. Games Technol. 2009 (Jan. 2009), 1-15.DOI= http://dx.doi.org/10.1155/2009/231863.

[3] *OnLive*. Disponível em: http://www.onlive.com/. Acessadoem: Março de 2011.

[4] Holub, P., Matyska, L., Liška, M., Hejtmánek, L., Denemark, J., Rebok, T., Hutanu, A., Paruchuri, R., Radil, J., and Hladká, E. 2006. *High-definition multimedia for multiparty low-latency interactive communication*. Future Gener. Comput. Syst. 22, 8 (Oct. 2006), 856-861. DOI= http://dx.doi.org/10.1016/j.future.2006.03.014.

[5] Barker, S. K. and Shenoy, P. 2010. *Empirical evaluation of latency-sensitive application performance in the cloud*. InProceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (Phoenix, Arizona, USA, February 22 - 23, 2010). MMSys '10. ACM, New York, NY, 35-46. DOI= http://doi.acm.org/10.1145/1730836.1730842.

[6] Guimarães M. P., Trevelin L. C., Machado L., Todesco G., Gnecco B., Brega J. 2010. *Virtualidade imersiva e interativa baseada em cloudcomputing*. SVR 2010 - Short Papers. Natal, Brasil. Junho de 2010.

[7] S. Robertson, C. Wharton, C. Ashworth, andM. Franzke. Dual device user interface design: *PDAsand interactive television*. In CHI ’96: Proceedings ofthe SIGCHI conference on Human factors incomputing systems, pages 79–86, New York, NY, USA,

[8] P. Cesar, D. Bulterman, and A. Jansen. *An Architecture for End-User TV Content Enrichment*. Journal of Virtual Reality and Broadcasting, 3(9), Dec.2006.

[9] Luiz Fernando Gomes Soares, Romualdo M.R. Costa, Marcio Ferreira Moreno, and Marcelo Ferreira Moreno. 2009. *Multiple exhibition devices in DTV systems*. In *Proceedings of the seventeen ACM international conference on Multimedia* (MM '09). ACM, New York, NY, USA, 281-290. DOI=10.1145/1631272.1631312 http://doi.acm.org/.

[10] ISO/IEC 13818-2. 2000. Information technology - *Generic coding of moving pictures and associated audio information: Video*.

[11] ITU-T – Telecommunication Standarization Sector of ITU. 2005. *Sereis H: Audiovisual and Multimedia Systems, Infrastructure of audiovisual services – Coding of moving video*.

[12] ABNT Associação Brasileira de Normas Técnicas. 2008. NBR 1560610.1145/1631272.1631312-2:2007, *Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 1: Codificação de dados, Versão corrigida* - 07.04.2008, Brazil.

[13] ABNT Associação Brasileira de Normas Técnicas. 2008. NBR 15606-2:2007, *Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 3: Especificação de transmissão de dados* - 07.04.2008, Brazil.

[14] ABNT Associação Brasileira de Normas Técnicas. 2008. NBR 15606-2:2007, *Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital, Parte 5: Ginga-NCL para receptores portáteis – Linguagem de aplicação XML para codificação de aplicações*- 07.04.2008, Brazil.

[15] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, V. 2003 *Rtp: a Transport Protocol for Real-Time Applications*. RFC. RFC Editor.

[16] Schulzrinne, H., Rao, A., and Lanphier, R. 1998 Real Time Streaming Protocol (Rtsp). RFC. RFC Editor.

[17] ISO/IEC 13818-1. 2000. *Information technology - Generic coding of moving pictures and associated audio information: Systems*.

[18] *UPnP™ Device Architecture 1.0.* UPnP™ Forum. Version 1.0.1, Maio de 2003.

[19] David Stirling and Firas Al-Ali. 2003. *Zero configuration networking*. *Crossroads* 9, 4 (June 2003), 19-23. DOI=10.1145/904080.904084 http://doi.acm.org/10.1145/904080.904084

[20] *OpenGL*. Disponível em: http://www.opengl.org/. Acessado em: Dezembro de 2010.

[21] *Wowza Media Server*. Disponível em: http://www.wowzamedia.com/. Acessado em: Outubro de 2010.