Um Ambiente para Simulação de Emissão de Aplicações Multimídia Complexas para TV Digital

Thiago N. Rodrigues
GSORT - IFBA
Rua Emídio Santos, s/n,
Salvador - BA
Email: thiagorodrigues@ifba.edu.br

Manoel C.M. Neto
GSORT - IFBA
Rua Emídio Santos, s/n,
Salvador - BA
Email: manoelnetom@ifba.edu.br

***Abstract*—This paper aims to present a simulation environment that can be used to run interactive applications developed in JavaDTV and NCL, focused on digital television. The proposed environment differs from others because it allows to simulate the broadcaster which is responsible for transmission of both main and extra content.**

***Index Terms*—TV Digital; Virtualização; Ginga-J;**

I. Introdução

A atual grande oferta por tecnologia, como *Internet* de alta velocidade, dispositivos móveis, *tablets*, entre outros, vem tornando a experiência do usuário cada vez mais rica e dinâmica no que diz respeito as suas atividades cotidianas. No âmbito da TV não é diferente. Antigamente, os televisores eram apenas receptores de conteúdo de áudio e vídeo gerados pela emissora de TV, hoje porém, pode-se observar que os aparelhos televisores oferecem uma gama muito grande de opções para seus usuários que vão além do consumo de mídias convencionais (áudio e vídeo) [1].

Todas as tecnologias tendem a evoluir, e com a TV não foi diferente. Atualmente a realidade de transmissões que é proporcionada é a TV Digital (TVD), que ao contrário da TV convencional (analógica) é capaz de oferecer transmissões de áudio e vídeo com definições bem superiores se comparados com as transmissões analógicas.

A TV Digital além de proporcionar qualidade superior nas transmissões ainda oferece a possibilidade de interatividade dos usuários através de aplicativos desenvolvidos para o ambiente de TV, caracterizando o que é chamado de TV Digital Interativa (TVDI) [2].

O desenvolvimento de aplicativos para TV Digital agrega mais interatividade à programação das emissoras, entretanto torna o processo de produção mais complexo. Antes a produção se limitava a vídeo e áudio, hoje porém o processo engloba *software* dando uma conotação multidisciplinar ao novo processo de produção [3][2].

Devido a essa nova faceta na produção de conteúdo para TV Digital, que agora engloba *softwares*, surgiram novos desafios a serem contornados no processo de produção. O desafio mais característico é o problema da sintonização tardia, que ocorre quando um telespectador sintoniza o canal após o envio de uma aplicação [2].

O crescimento da TV Digital em todo mundo vem atrelado ao crescimento em investimentos e em pesquisas na área [4]. Como em qualquer plataforma de desenvolvimento, é necessário ferramentas, técnicas e ambientes que permitam a implementação, testes e simulações dos aplicativos produzidos para esse novo contexto.

Segundo Laureano (2006) [5] a simulação, seja através de emuladores ou máquinas virtuais, é uma alternativa viável devido a redução dos custos, não sendo necessário manter um ambiente real de produção que no ambiente de TV é muito caro, e portabilidade, sendo possível realizar testes e experimentos em outro ambiente que não o de produção.

O objetivo do trabalho proposto é a construção de um ambiente que permita a simulação de um *broadcaster* em rede TCP/IP de conteúdo principal e de conteúdo extra, principalmente de uma categoria de aplicações interativas cujo conteúdo extra consumido tem relação com a semântica do conteúdo principal de áudio e vídeo apresentado (especialmente em transmissões de conteúdo ao vivo) chamadas de Aplicações Multimídia Complexas (AMC) [6].

Neste trabalho será apresentado inicialmente um capítulo sobre TV Digital abordando sua arquitetura, os tipos de aplicações para TVDI, o problema da sintonização tardia e os aspectos ubíquos relacionados à TV Digital e ao simulador proposto. No terceiro capítulo serão tratados os trabalhos relacionados, na sequência os capítulos sobre o protocolo desenvolvido, a arquitetura, a implementação e um capítulo sobre a ferramenta e seu funcionamento. Por fim, os capítulos sobre os testes e resultados além da conclusão e propostas de trabalhos futuros.

II. TV Digital

A TV Digital trouxe grandes vantagens para os telespectadores se comparada com à TV analógica. Além da qualidade superior de áudio e vídeo, a interatividade, através de aplicações, é uma das grandes inovações da TV Digital [3].

A implantação e crescimento da TV Digital está diretamente ligada a mudanças tanto na forma de produção de conteúdo, que passa a englobar *softwares*, quanto na arquitetura de transmissão das emissoras.

Neste capítulo será apresentado inicialmente a nova arquitetura de transmissão para TV Digital, seguido dos tipos de

aplicativos. Será abordado também o problema da sintonização tardia e como resolvê-lo e os aspectos ubíquos relacionados ao propósito do trabalho aqui apresentado.

A. Arquitetura de TV Digital

Devido ao novo desenho televisivo que a TV Digital passa a proporcionar depois da sua evolução, oferecendo aos telespectadores mais que apenas vídeo e áudio de alta qualidade, surgiu uma nova arquitetura que constitui a TV Digital e sua etapas de produção, transmissão e recepção [1].

Fig. 1. Arquitetura da TV Digital

Segundo Zuffo (2003) [4] a situação da TV aberta analógica era de apenas produção digital, enquanto que a transmissão e recepção eram completamente analógicas. Hoje porém, considerando a nova arquitetura da TVD, tanto a produção quanto a transmissão e recepção são digitais.

De acordo com a Figura 1, a nova arquitetura da TV Digital é agora composta por seis elementos básicos, cada um com suas características e responsabilidades bem definidas, e são [1]:

1) *Ilha de Edição e Aplicativos*: É o responsável pela produção dos conteúdos principal e extra .
2) *Multiplexador*: Responsável por integrar todos os elementos produzidos em um único conteúdo audiovisual.
3) *Modulador*: É o elemento responsável pela transmissão do conteúdo multiplexado pela emissora.
4) *Set-Top-Box e DTV*: Responsável pela sintonização e recepção do conteúdo gerado pelas emissoras.
5) *Dispositivos*: Elementos (celulares, *tablets* e etc.) que agregam na interação entre os telespectadores e os conteúdos interativos das emissoras.
6) *Canal de Retorno e Internet*: Aplicações interativas podem utilizar canais de interatividade para acessar conteúdos provenientes de outros provedores e também retornar informações à emissora.

Exceto pelo fato de que o simulador proposto não realiza a multiplexação entre conteúdo principal e extra, transmitidos-os por canais diferentes, a intenção é simular a parte que representa a cadeia de produção e transmissão da arquitetura apresentada que consiste da Ilha de Edição e Aplicativos, Multiplexador e Modulador.

B. Programas para TV Digital Interativa

No contexto da computação, a palavra "programa" é sinônimo de *software*, entretanto no contexto da TV o significado mais apropriado é "atração". Em [1], um programa para TV Digital Interativa é definido como uma aplicação multimídia pela qual o telespectador pode interagir via controle remoto.

Para Neto (2011) [1] e (2008) [7], os programas ou aplicativos para TVDI podem ser divididos em três categorias distintas:

1) Aplicativos que não possuem vínculo semântico com o conteúdo principal. Por exemplo, acessar e-mails durante a exibição de um filme.
2) Aplicativos com relação semântica com o conteúdo de áudio e vídeo apresentados, porém sem restrições fortes de sincronização. Por exemplo, visualizar a tabela de classificação de um campeonato de futebol durante a exibição de uma partida.
3) Aplicativos que além de possuir relação semântica também possuem forte sincronização com o conteúdo exibido. Por exemplo, anúncios publicitários que são exibidos quando um produto é enquadrado durante a exibição de um filme ou novela. Esta última categoria ainda pode ser subdividida em mais duas:
   a) Aplicativos cujo conteúdo principal é conhecido a priori (filme, novela e etc.)
   b) Aplicativos cujo conteúdo principal é gerado ao vivo (futebol, corridas e etc.)

O desenvolvimento de aplicativos para TVDI da primeira categoria é similar ao processo tradicional de desenvolvimento de *software*, o que diferencia são os requisitos não-funcionais como "plataforma de execução" e "dispositivos de entrada e saída". Na segunda e terceira categorias as particularidades do ambiente de TV e o contexto aplicado devem ser respeitados no processo de produção [1].

Em [6], os aplicativos da terceira categoria, especialmente os que o conteúdo principal é gerado ao vivo, são denominados de Aplicativos Multimídia Complexos (AMC), e com o objetivo principal de simular a transmissão desses tipo de aplicativos que o simulador deste trabalho é proposto.

C. Sintonização tardia

Ao utilizar uma televisão o telespectador pode sintonizar um canal a qualquer momento de sua transmissão. Levando em consideração o conteúdo principal convencional (áudio e vídeo) este fato não acarreta nenhum problema. Porém no caso da transmissão de conteúdo extra (aplicações multimídia) a sintonização após o início de envio desta aplicação (sintonização tardia) pode ocasionar que o telespectador não receba o conteúdo ou o receba parcialmente. Outro fator que agrava esta situação é que o sentido da comunicação é unidirecional, esta característica impossibilita que o telespectador

faça a requisição do conteúdo extra ao sintonizar um canal [8].

Diante desta situação, é necessário uma solução que possibilite ao telespectador sempre receber o conteúdo, ainda que ele tenha sintonizado o canal tardiamente. Para tal fim, o Sistema Brasileiro de TV Digital, adotou o protocolo padrão DSM-CC (*Digital Storage Media Command and Control*) para a transmissão de conteúdo extra em um ambiente de difusão [8].

O protocolo DSM-CC possui características cíclicas para transmissão de dados, o que torna os recebimentos independentes do instante de sintonização, e proporcionando assim, para os usuários, a possibilidade de receber o conteúdo esperado ainda que o mesmo tenha sintonizando após o início da transmissão, bastando apenas esperar o próximo ciclo de envio [9].

Devido a suas características cíclicas de envio, o protocolo DSM-CC é conhecido como "Carrossel", podendo ser dividido em dois tipos principais: (i) carrossel de dados e (ii) carrossel de objetos [8][10].

1) *Carrossel de dados*: Utilizado para envio de informações não estruturadas, como por exemplo uma imagem ou um arquivo de texto, que não possuem relação semântica com o conteúdo principal [8].
2) *Carrossel de objetos*: Utilizado para difusão de informações estruturadas como diretórios e aplicativos. O carrossel de objetos é construído sobre o carrossel de dados e possui uma série de mensagens e controles que possibilitam a estruturação da informação, tornando possível a transmissão de sistemas de arquivos, aplicações multimídia e objetos do tipo *stream* e *stream event* que são utilizados, no caso do DSM-CC, para fazer referência ao fluxo principal, possibilitando a relação semântica dos objetos transmitidos com o conteúdo principal [9][8].
   Para a estruturação das informações, o carrossel de objetos utiliza cinco tipos de objetos para a transmissão de dados, são eles: (i) File, (ii) Directory, (iii) Service Gateway, (iv) Stream e (v) Stream Event [10]. Os itens abaixo descrevem o objetivo de cada um destes objetos.
   a) *File*: Possui informações acerca de um arquivo a ser transmitido [10].
   b) *Directory*: Representa a construção de um diretório [10].
   c) *Service Gateway*: Semelhante ao objeto *Directory*, este objeto representa o diretório raiz do conjunto de dados a ser transmitido [10].
   d) *Stream*: Representa um fluxo de áudio e vídeo [10].
   e) *Stream Event*: Representa pontos de sincronização contidos entre objetos do tipo *Stream* [10].

Além das características sobre estruturação da informação, o carrossel também pode ser espeficificado de acordo com ambiente de transmissão utilizado na comunicação. Carrosséis do tipo U-U (*User to User*) são especificados pelo protocolo para comunicação ponto-a-ponto. Nesse caso, esse tipo de carrossel é utilizando apenas para o canal de retorno, que é quando há necessidade do receptor enviar alguma informação para o provedor de conteúdo (a resposta de uma enquete, por exemplo). Já os carrorréis do tipo U-N (*User to Network*) são especificados para utilização em ambientes de *broadcast*, sendo adotados, dessa forma, para transmissões por difusão, que é o caso do envio de conteúdo principal e extra. [8].

Apesar do DSM-CC já ser um padrão adotado pelo Sistema Brasileiro de Televisão Digital, foi desenvolvido para o simulador proposto um protocolo baseado no DSM-CC, porém menor e com menos recursos devido ao extenso tamanho e complexidade do protocolo padrão, utilizando assim, um conjunto reduzido e simplificado de seus objetos.

O protocolo desenvolvido neste trabalho implementa um carrossel de objetos do tipo U-N e busca a maior similaridade conceitual possível com o protocolo DSM-CC a fim de manter as características cíclicas do mesmo, além de garantir entrega de forma confiável dos conteúdos transmitidos.

D. Aspectos Ubíquos

Com o avanço das tecnologias e a proliferação dos dispositivos móveis com inúmeros recursos, o desenvolvimento de aplicações multimídias inteligentes vem sendo cada vez mais estimulado. Com isso, a forma com que as pessoas trabalhavam e interagiam com conteúdos multimídia vem sofrendo mudanças, acrescentando facilidades no cotidiano das pessoas e promovendo uma revolução multimídia que impacta em todos os setores da sociedade [11].

Para TV Digital, o contexto é o mesmo já que hoje é possível a veiculação de *softwares* interativos juntamente com qualquer outro programa da grade do emissor [12][13]. Essa nova abordagem adotada agrega novas características ao seu processo de produção como a multidisciplinaridade, interatividade, heterogeneidade, convergência, etc. Essas características são semelhantes com aquelas encontradas em sistemas ubíquos [11] onde *softwares* diferentes podem rodar em plataformas diferentes e muitas vezes não convencionais (ex.: TV).

O projeto apresentado propõe a criação de um emissor *bordcaster* para esses aplicativos multimídia, permitindo a difusão de conteúdo extra e principal no âmbito da TV Digital. A seguir serão explicados os aspectos que tornam o projeto com características ubíquas.

1) *Invisibilidade*: O emissor de conteúdo é independente dos receptores, enviando os dados de conteúdo principal e extra em *broadcast*. O receptor para receber os dados deve sintonizar no canal de transmissão sendo totalmente transparente para o telespectador de onde vem os dados recebidos. Para o emissor também é indiferente quem consome os dados.
2) *Interoperabilidade*: Para garantir a troca de informações, ainda que os dispositivos sintonizados ao emissor de conteúdo utilizem outras tecnologias, foi implementado um protocolo baseado no DSM-CC, isso permite que receptores que implementem o mesmo protocolo possam consumir o conteúdo transmitido independente da

plataforma sobre a qual o emissor foi desenvolvido e está sendo executado.

3) *Ambiente Volátil*: Segundo Colouris (2005) [14], um sistema volátil é aquele que alterações são comuns e não excepcionais e o o conjunto de usuário, *hardware* e *software* é dinâmico, mudando de forma imprevisível. O simulador *broadcaster* proposto transmite os dados na rede e qualquer receptor pode sintonizar para recebê-los, entretanto o *broadcaster* é completamente independente dos receptores. Tanto a sintonização quanto a dessintonização de um receptor não afeta em nada o emissor, dessa forma não há necessidade de um tratamento específico quando algum receptor deixa de receber os dados, uma vez que a comunicação no ambiente de simulação é semelhante ao da TV convencional sendo unidirecional e ocorrendo sempre do emissor para os receptores.
4) *Sensibilidade ao contexto*: Para Weiser (1991) [15], a percepção do contexto é uma característica intrínseca de ambientes inteligentes. Considerando o *broadcaster* como parte de um ambiente completo de simulação pode-se dizer que a sensibilidade ao contexto é favorecida pela possibilidade das aplicações transmitidas de consumir conteúdos de outros contextos (provedores), como a *web*, dependendo da situação e do propósito da aplicação
5) *Mobilidade Lógica*: O grande objetivo do simulador é a transmissão de AMC's para execução em uma plataforma cliente, dessa forma, o código é movido do *broadcaster* para um simulador de receptor para interação dos usuários.

É importante lembrar que o simulador de *broadcaster* faz parte de um cenário maior que consiste no conjunto de um emissor, além do receptor e eventualmente um *provider* (provedor de conteúdo que não o *broadcaster*) e muitas vezes as características apresentadas só fazem sentido se analisadas tendo em vista o ambiente completo de simulação uma vez que as partes se completam.

III. Trabalhos Relacionados

Com o surgimento da TV Digital e sua oferta de interatividade ao telespectador, é natural que investimentos em pesquisas nessa área sejam realizados. Devido a possibilidade de interação, que agrega agora ao processo da TV desenvolvimento de software (aplicativos para TV), é possível encontrar diversos trabalhos voltados para ferramentas capazes de oferecer suporte à implementação, transmissão, execução, teste e simulação desses aplicativos.

Neste capítulo será descrito dois trabalhos relacionados. O primeiro é composto de um carrossel de dados para transmissão de sistemas de arquivos e o segundo uma implementação do gerador de mensagens do protocolo DSM-CC sobre IP. Um comparativo entre os dois trabalhos e o simulador aqui desenvolvido será apresentado no final.

A. Um Carrossel de Dados para Transmissão de Sistemas de Arquivos

O DSM-CC, protocolo adotado pela maioria dos sistemas de televisão digital no mundo, é baseado no conceito de carrossel para o envio cíclico de conteúdo multimídia, permitindo dessa forma que os telespectadores tenham acesso ao conteúdo ainda que tenham sintonizado tardiamente o canal. O carrossel do DSM-CC pode ser dividido em dois, o carrossel de dados que tem como objetivo enviar dados não estruturados como uma arquivo de imagem, áudio ou vídeo, por exemplo, e o carrossel de objetos que é implementado sobre o carrossel de dados e permite a transmissão de dados estruturados como um sistema de arquivos inteiro entre outros tipos de mensagens típicos do carrossel de objetos.

Em [16], o desenvolvimento do carrossel de objetos é tratado como complexo e de difícil implementação. Devido a essa dificuldade, o objetivo do trabalho é a implementação do carrossel de dados do DSM-CC porém modificado de forma que seja possível a transmissão de diretórios inteiros (e seus conteúdos) sem a necessidade de implementar o carrossel de objetos sobre o de dados.

Para alcançar o objetivo foi proposto alterações na implementação do carrossel de dados, que com a ajuda de estrutura e descritores adicionais seria possível a transmissão de sistemas de arquivos inteiros sem a necessidade de implementação do carrossel de objetos. Com as mudanças nos descritores, o protocolo agora pode carregar informações sobre o caminho absoluto e relativo dos diretórios ou arquivos entre outras informações que permitem a execução da transmissão de forma que se crie um espelho da árvore de diretório transmitido no receptor.

Os experimentos realizados no estudo apresentado em [16] apontam que a solução é viável e eficaz. Para se chegar a essa conclusão foram realizados três testes denominados Test1 onde é transmitido uma árvore de diretório de três níveis e cinco arquivos, Test2 para um aplicativo java com dez arquivos, e Test3 um arquivo Java de extensão ".jar".

Tamanho Original (Byte)	**Protocolo**	**Tamanho Total (Byte)**	**Eficiência da largura de banda (%)**
Test1 - 20674 bytes	CO	25568 bytes	80.9
Test1 - 20674 bytes	CD	23668 bytes	87.9
Test2 - 30057 bytes	CO	34216 bytes	87.8
Test2 - 30057 bytes	CD	32712 bytes	91.9
Test3 - 22220 bytes	CO	24628 bytes	90.2
Test3 - 22220 bytes	CD	24064 bytes	92.3

TABLE I
TESTE DE TRANSMISSÃO

Na Tabela I, pode-se observar que a eficiência da transmissão quando realizada com o carrossel de dados proposto (CD) e com o carrossel de objetos (CO) são bem próximas nos três casos. Apesar dos bons resultados é importante lembrar que o carrossel de dados apresentado em [16] propõe apenas a transmissão de diretório utilizado o carrossel de dados, não contemplando outras funções características dos carrosséis de objetos.

B. Gerador de mensagens do DSM-CC sobre IP

Com o surgimento da TV Digital e a convergência das tecnologias para dispositivos móveis, já é possível encontrar no mercado aparelhos capazes de recepcionar o sinal de TVD. Apesar dessa funcionalidade que agrega exibição de TV Digital em dispositivos móveis, que antes só era possível nas próprias televisões, tais aparelhos não dão suporte a execução de conteúdo multimídia interativa devido a ausência do *middleware* Ginga (adotado pelo Sistema Brasileiro de Televisão Digital).

Com o objetivo de permitir a execução de conteúdo multimídia em dispositivos móveis é proposto em [8] a implementação do Ginga-NCL para estes dispositivos baseados no sistema operacional Android, dessa forma sendo possível a execução de aplicações escritas em NCL nos aparelhos com a tecnologia embarcada.

Para testar e validar a estrutura construída foi necessário a implementação de uma parte do protocolo DSM-CC (DSM-CC sobre IP) permitindo, dessa forma, a avaliação da transmissão e recepção dos conteúdos. Apesar do objetivo central do trabalho não ser o desenvolvimento do protocolo DSM-CC essa parte é de grande importância para o objetivo final uma vez que o servidor integrado com o gerador de mensagens DSM-CC compõe o ambiente para a implementação do Ginga-NCL em dispositivos baseados em Android.

A Figura 2 mostra como é a composição e o funcionamento do servidor que implementa o carrossel de dados com o gerador de mensagens do DSM-CC e os aparelhos móveis clientes com a implementação do *middleware* Ginga-NCL através de uma rede.

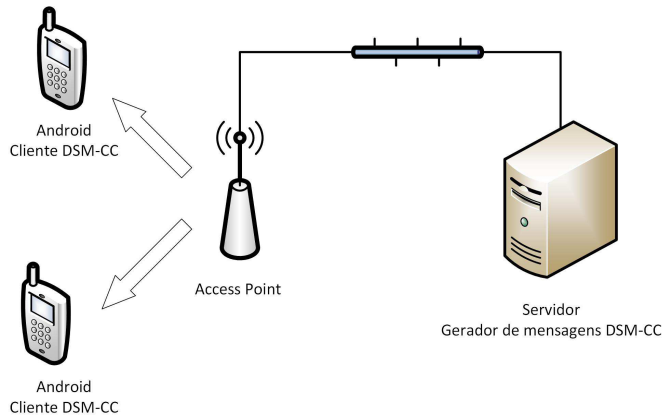

Fig. 2. Arquitetura de transmissão

Basicamente, o servidor envia em *broadcast* pela rede as mensagens geradas pelo protocolo DSM-CC sobre IP dos aplicativos NCL adicionados ao carrossel. Os aparelhos com o sistema operacional Android recepcionam o conteúdo a partir do cliente DSM-CC também implementado e incluída da solução juntamente com o Ginga-NCL desenvolvido para a plataforma móvel no trabalho apresentado.

O servidor implementado não oferece a opção de transmissão de conteúdo principal, sendo assim, a sua única função é a de transmissão de conteúdo multimídia a fim de corroborar com o funcionamento da implementação do Ginga-NCL. Os testes realizados para validar o *middleware* consistiam em uma simples transmissão de aplicativos NCL e sua visualização no aparelho móvel. Os resultados mostraram que a implementação do protocolo funcionou de forma satisfatória, uma vez que todos as transmissão foram concluídas.

C. Comparativo

Os trabalhos apresentados evidenciam uma característica importante para a simulação da TVD no que diz respeito a sua faceta multimídia: a transmissão de conteúdo extra. Enquanto que no primeiro trabalho o objetivo é a implementação de uma forma mais simples para transmissão de conteúdos complexos sem a utilização do carrossel de objetos, que é de difícil implementação, o segundo já possui o objetivo de implementá-lo a fim de gerar mensagens DSM-CC no intuito de validar o *middleware* Ginga-NCL para dispositivos móveis.

O simulador apresentado neste trabalho se aproxima dos dois trabalhos relacionados na questão de transmissão de conteúdo extra, possibilitando a transmissão desde conteúdos simples como um único arquivo a um sistema de arquivo inteiro (características do carrossel de dados e carrossel de objetos do DSM-CC). Por outro lado, o objetivo do simulador é fornecer um ambiente completo para simulação e testes de aplicativos para TVD o que torna necessário não só a transmissão de conteúdo extra em todas suas formas, como também o conteúdo principal, o que é de grande importância para a execução de aplicações interativas nas quais o conteúdo extra possui relação semântica com conteúdo audiovisual principal exibido (AMC).

IV. Protocolo

Com o objetivo de manter uma entrega confiável sem a utilização do TCP e uma maior similaridade possível com DSM-CC foi definido um protocolo de verificação de integridade e de tipos de objetos.

A Tabela II representa o protocolo desenvolvido para o simulador que é capaz de encapsular todas as informações necessárias acerca dos objetos a serem transmitidos, atendendo também às características do carrossel de objetos, uma vez que o protocolo é baseado no DSM-CC. Para que o protocolo possua tais características cíclicas é necessário que algumas informações estejam presentes em seu escopo a fim de que os receptores possam identificar quais pacotes já foram recebidos e aqueles que ainda não foram.

A transmissão dos pacotes é feita pelo simulador através do serviço UDP, não havendo dessa forma garantia de entregas dos mesmos, logo é de extrema importância a identificação da quantidade total, assim como a sequência dos pacotes recebidos já que o controle de recebimento é feito pelos receptores com base no protocolo.

De acordo com a Tabela II, os campos do protocolo são:

1) *ID*: Cada objeto do conteúdo adicionado ao carrossel recebe um identificador único. Por exemplo, se uma aplicação é composta por um *.class* e mais dois arquivos

Campo	Tamanho	Descrição
ID	8 bytes	Identificador único do objeto no carrossel
Type	2 bytes	Tipo do objeto (Single File, Binded File, Binder Directory e Binder Application)
Version	2 bytes	versão do objeto
Number of Packets	8 bytes	Quantidade total de pacotes que compõe o objeto
Packet Sequence	8 bytes	Número de sequência do pacote
Data Length	4 bytes	Tamanho da área de dados utilizado no pacote
Data	30720 bytes	Área reservada para dados

TABLE II
CABEÇALHO DO PROTOCOLO

XML, todos os três arquivos serão mapeados como objetos do carrossel e receberão identificadores únicos. Dessa forma o receptor poderá tratar cada elemento individualmente e verificar sua consistência.

2) *Type*: Tipo do objeto no carrossel de dados. Os tipos possíveis de objetos são:
   a) *Single File*: Arquivo avulso, não possui dependência com nenhum outro objeto e pode ser consumido ao término de sua transmissão.
   b) *Binded File*: Arquivo que possui dependência com outros objetos e só pode ser consumido quando a transmissão de todos os outros também estiver concluída. Esses objetos são gerados a partir de aplicativos ou diretórios. Ao selecionar uma pasta para envio, todos os arquivos da pasta e das subpastas são mapeados como *Binded File*, o mesmo acontece para Aplicativos, ao selecionar um arquivo Java com extensão *class*, todos os arquivos que estão na mesma pasta e nas subpastas do seu diretório base também serão mapeados como *Binded File*.
   c) *Binder Directory*: Pacote que contém informações acerca dos objetos (*Binded File*) que compõe o diretório transmitido.
   d) *Binder Application*: Pacote que contém informações acerca dos objetos (*Binded File*) que compõe a aplicação transmitida, além de dados como nome da aplicação, tipo de execução e classe principal.
3) *Version*: Versão do objeto no carrossel. Em alguns casos pode ser necessário a substituição do conteúdo transmitido. Os novos objetos substituem os antigos no carrossel mantendo o mesmo ID e atualizando sua versão, dessa forma o carrossel receptor não descartar o objeto e substitui o antigo pelo novo recebido.
4) *Number of Packets*: Número total de pacotes que compõe o objeto do conteúdo transmitido. O objeto só pode ser considerado completo no receptor após receber todos os pacotes de acordo com campo em questão.
5) *Packet Sequence*: Número de sequência do pacote recebido. Uma vez que o UDP não garante a entrega dos pacotes, é necessário um controle dos pacotes recebidos. Através dos campos *ID* e *Packet Sequence* é possível identificar os pacotes que já estão concluídos e os que ainda faltam ser recepcionados, evitando duplicidade e garantindo a consistência dos dados.
6) *Data Length*: Tamanho em bytes utilizado na área de dados do pacote. Como não necessariamente um pacote ocupa toda área reservada para dados, o campo é importante para evitar a leitura de lixo.
7) *Data*: Área reservada para dados. Para os pacotes do tipo *Binder Directory* e *Binder Application* é utilizado para transmissão de informações do conjunto de objetos que compõe o aplicativo ou diretório, além de outras características. Para tal, foi estabelecido um subprotocolo para área de dados do pacote que será descrito posteriormente.

A área de dados do protocolo é utilizada para transportar as informações pertinentes ao conteúdo. Para os objetos do tipo *Single File* e *Binded File* a área de dados transporta os dados do arquivo o qual ele representa. Por exemplo, ao adicionar um vídeo no carrossel, *luta.avi*, ele será representado como um *Single File*. Supondo que o vídeo tenha 60Kb (61440 *bytes*) de tamanho, precisaremos de dois pacotes para conseguir transportar todo conteúdo já que o tamanho máximo da área de dados do protocolo é de 30720 *bytes*. Além dos pacotes onde são transmitido os dados, também é gerado um pacote exclusivo para metadados do conteúdo, que nesses dois casos é apenas o nome (e caminho) do arquivo. Logo, a quantidade total de pacotes por objeto é igual ao tamanho total do arquivo dividido pelo tamanho máximo da área de dados do protocolo, acrescido de mais um pacote de metadados.

Objetos do tipo *Binder Directory* não transportam dados de nenhum arquivo, seu objetivo é relacionar todos os objetos do tipo *Binded File* que foram gerados a partir dos arquivos que a pasta selecionada para envio contém (incluindo arquivos das subpastas). Todos os identificadores dos objetos referentes aos arquivos da pasta são adicionados a área de dados do pacote de um objeto *Binder Directory*. O pacote pode transmitir 3840 identificadores, já que, de acordo com o protocolo, cada identificador de objeto possui 8 *bytes* e o tamanho máximo da área de dados é de 30720 *bytes*. O receptor deve utilizar as informações contidas nesse pacote para verificar se todos os arquivos do diretório enviado já foram recepcionados e liberar o conteúdo para consumo.

Objetos do tipo *Binder Application* também não transportam dados de nenhum arquivo, este tipo de objeto relaciona todos os arquivos que compõem o aplicativo de forma semelhante ao *Binder Directory*. Quando um arquivo Java de extensão *class* é selecionado para envio, o Broadcaster o identifica como um conteúdo do tipo “Application”, então todos os arquivos contido na mesma pasta do arquivo Java são mapeados como *Binded File*, e o mesmo acontece com os arquivos das subpastas do diretório do conteúdo selecionado. O Objeto *Binder Application* deve relacionar todos esses objetos para que o receptor possa verificar se todos os arquivos da aplicação foram recebidos e liberar o conteúdo para consumo.

Bytes	Descrição
1º	Tipo de execução da aplicação (*Automatic*, *Notify*, *List*)
2º	Tamanho em *bytes* relativo ao caminho para classe de arranque
3º ao 252º	Caminho para a classe de arranque
253º	Tamanho em *bytes* relativo ao nome da aplicação
254º ao 285º	Nome da aplicação
286º ao 289º	Quantidade de ids de objetos contidos no conteúdo extra
290º ao 30720º	Ids dos objetos do conteúdo extra

TABLE III
ÁREA DE DADOS PARA PACOTES DE OBJETOS DO TIPO BINDER APPLICATION

A Tabela III representa as informações que são transmitidas na área de dados dos pacotes dos objetos do tipo *Binded Application*. O primeiro *byte* indica o tipo de execução da aplicação. O segundo *byte* é referente a quantidade de *bytes* utilizados com o nome da classe principal (*.class*) que pode utilizar até 250 *bytes* (do 3º ao 252º). O 253º *byte* representa a quantidade de *bytes* utilizados com o nome da aplicação que tem reservado até 32 *bytes* (do 254º ao 285º) para esse fim. Na sequência, são 4 *bytes* para representar a quantidade de identificadores transmitidos. A área reservada para transmissão de identificadores dos objetos é de 30430 bytes (do 290º ao 30720º) o que pode representar até aproximadamente 3803 identificadores de objetos do tipo *Binded File*.

V. ARQUITETURA

A arquitetura do Broadcaster, apresentada na Figura 3, pode ser representada por dois módulos chamados de "Gerador de conteúdo principal" e "Gerador de conteúdo Extra".

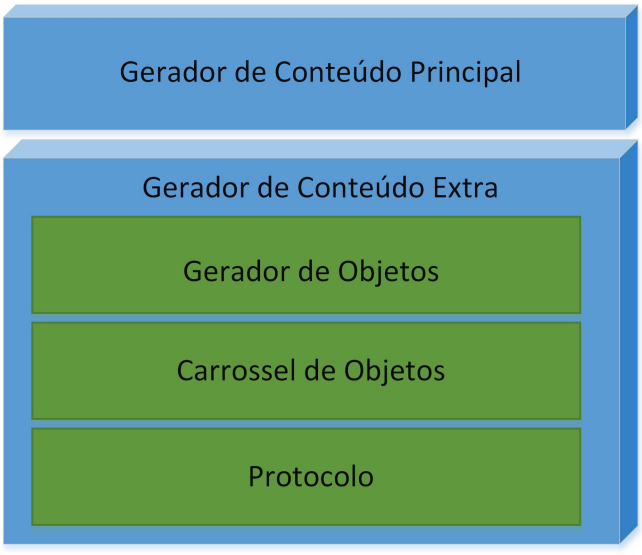

Fig. 3. Arquitetura

A. Gerador de conteúdo Principal

Para transmissão de conteúdo principal, o módulo *Gerador de Conteúdo Principal* utiliza a biblioteca *libvlc* incluída no projeto Vídeo LAN media player (VLC) e uma API denominada VLCj que permite manipular as funcionalidades de transmissão, recepção e codificação de mídias a partir de código escrito em Java [17].

A API VLCj é carregada em tempo de execução pelo Broadcaster já que a mesma é distribuída juntamente com *player* VLC e não precisa ser instalada juntamente com o simulador. Após carregada a API, o módulo *Gerador de conteúdo Principal* está apto para transmitir o conteúdo principal a partir de um arquivo de vídeo do computador.

Para transmissão do conteúdo principal o módulo utiliza o protocolo UDP, uma vez que não há necessidade de garantia de entrega de pacotes e a eventual perda de alguns é aceitável pois não compromete o objetivo real do simulador que é a transmissão de aplicações multimídia complexas.

B. Gerador de conteúdo Extra

O módulo *Gerador de conteúdo extra* tem a finalidade de transmitir em *broadcast* todo conteúdo extra gerado pelo canal. Para transmissão do conteúdo, alguns passos devem ser seguidos:

1) *Identificação do tipo de conteúdo*: O conteúdo escolhido deverá ser verificado com a finalidade de indentificar-se o seu tipo, que pode ser "Application", "File" ou "Directory". Caso o conteúdo selecionado seja um arquivo Java de extensão *class*, o módulo o identificará como *Application*, caso seja qualquer outro tipo de arquivo, o conteúdo será identificado como *File*. Se o conteúdo for uma pasta, ele será identificado como sendo do tipo *Directory*.
2) *Criação dos objetos*: Os arquivos do conteúdo selecionado são mapeados em objetos para o carrossel do simulador. Todos os arquivos do diretório selecionado ou da pasta onde se encontra o arquivo *.class* da aplicação (incluindo os arquivos das subpastas) são mapeados em objetos do carrossel. O único arquivo do conteúdo do tipo *File* também é mapeado em um objeto. Todos os tipos de objetos serão explicados posteriormente.
3) *Inserção no carrossel de dados*: Inclusão dos objetos gerados a partir dos arquivos relacionados ao conteúdo transmitido em uma fila circular para transmissão.
4) *Transmissão*: Envio, dentro de uma janela de tempo, dos objetos em *broadcast*.

O módulo "Gerador de Conteúdo Extra" possui o objetivo principal de promover a difusão de AMC's e qualquer outro tipo de recurso, atuando desde a coleta dos itens que compõe o conteúdo selecionado e sua transformação em objetos, até a sua transmissão de fato. Devido as suas várias funções, o módulo em questão foi sub-divido em outros três sub-módulos, são eles:

1) *Gerador de Objetos*: Este sub-módulo é responsável por identificar o tipo do conteúdo transmitido e percorrer de forma recursiva o diretório do conteúdo enumerando os seus arquivos e mapeando-os em objetos.
2) *Carrossel de Objetos*: O carrossel foi desenvolvido de forma que não mantenha a responsabilidade na criação

dos objetos, dessa forma o carrossel apenas recebe uma estrutura que representa o conteúdo que deseja ser transmitido com os objetos já listados e prontos para entrar na fila circular de transmissão. O carrossel apenas desencapsula a estrutura recebida e adiciona todos os objetos no final de sua fila.

3) *Protocolo*: Este sub-módulo possui todas as estruturas necessárias de representação do protocolo, desde classes de encapsulamento dos dados para o pacote, até estruturas de conversão dos dados em *bytes* para transmissão.

Após a seleção do conteúdo extra que deve ser transmitido, o sub-módulo *Gerador de objetos* deve identificar qual é o tipo do conteúdo, isso porquê os objetos criados para o carrossel variam de acordo com o que se quer transmitir. Como pode ser visto na Figura 4, conteúdo do tipo *Directory* pode gerar objetos do tipo "Binded File" e "Binder Directory", já os conteúdos do tipo *Application* podem gerar objetos do tipo "Binded File" e "Binder Application". Arquivos simples (conteúdo do tipo *File*) geram apenas objetos do tipo "Single File".

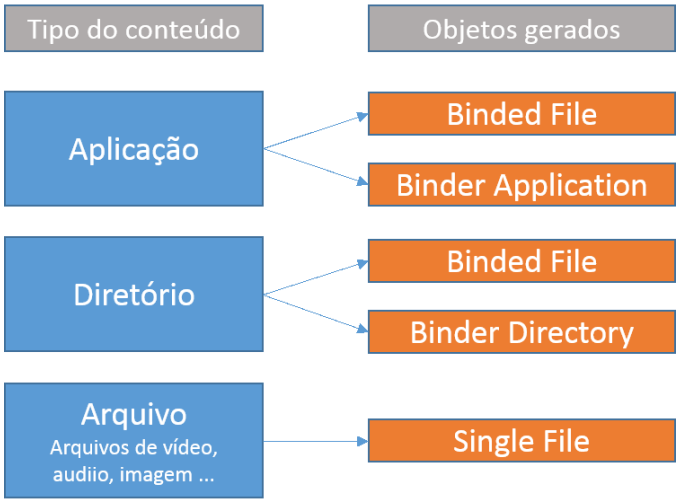

Fig. 4. Objetos gerados

Para conteúdos do tipo diretório e aplicação, os arquivos do conteúdo são todos mapeados como *Binded file*, que diferem do *Single file* por serem objetos que possuem dependência de outros arquivos, ou seja, esses objetos não podem ser consumidos até que todos os objetos do mesmo conteúdo sejam recepcionados. A Figura 5 representa a estrutura de uma aplicação, "MMA App", que possui um arquivo Java de extensão *class* e mais duas imagens. Nesse caso, os arquivos *fighter1.png*, *fighter2.png* e *MMA.class* serão mapeados como *Binded File* já que eles compõem a aplicação e são dependentes entre si.

Ainda no exemplo do MMA App, todo conteúdo do tipo aplicação gera um objeto *Binder Application* (*Binder Directory* para diretório). Esse tipo de objeto tem a finalidade de carregar informações sobre o conteúdo e os identificadores dos objetos que compõe a aplicação. Dessa forma, o receptor pode saber ao receber os arquivos do tipo *Binder* quais objetos possuem dependência entre si e liberar o consumo do conteúdo quando todos forem recebidos. A seguir um resumo sobre os tipos de objeto:

1) *Single File*: Representa um arquivo sem relação com nenhum outro objeto.
2) *Binded File*: Representa um arquivo dos conteúdos do tipo *Application* ou *Directory* que possuem relação com outros objetos.
3) *Binder Application*: Relaciona os objetos que formam a aplicação. Estrutura a informação para utilização do receptor.
4) *Binder Directory*: Relaciona os objetos que formam a pasta transmitida estruturando a informação do diretório para o receptor montá-lo no recebimento dos dados.

Os objetos do tipo *Binder Directory* carregam os identificadores de todos os arquivos que compõem o diretório transmitido, dessa forma o receptor pode estruturar os dados recebidos. Já os objetos do tipo *Binder Application* além dos identificadores de todos os objetos que compõem a aplicação, carregam também o nome do conteúdo, o caminho para o arquivo principal e o tipo de execução da aplicação. Os tipos de execução dos conteúdos do tipo *Application* são:

1) *Automatic*: A aplicação transmitida deve ser iniciada automaticamente assim que for recebida pelo receptor.
2) *Notify*: A aplicação não deve ser iniciada no receptor, porém o usuário deve ser notificado do recebimento de um novo conteúdo. Para acessá-lo o usuário deverá iniciar o aplicativo a partir de uma lista de aplicativos que o receptor pode manter.
3) *List*: A aplicação não é iniciada e nem o usuário é notificado do recebimento do conteúdo, entretanto, a aplicação deve ser adicionada à lista de aplicativos do receptor.

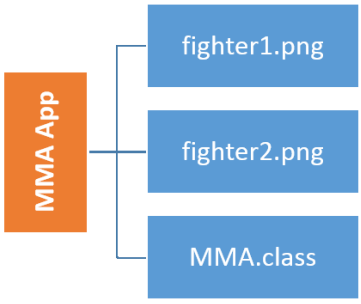

Fig. 5. MMA App

Após mapear todos os arquivos e criar os *Binders*, o sub-módulo "Carrossel de objetos" deve receber o conjunto de objetos que representa o conteúdo e adicioná-los ao carrossel de objetos do simulador. Cada objeto do conteúdo a ser

transmito é adicionado individualmente no carrossel e cada um possuirá sua própria janela de tempo para ser transmitido.

O carrossel desenvolvido funciona de forma semelhante ao DSM-CC, cada objeto é transmitido em uma fatia de tempo fixa. Assim, cada vez que o tempo de uma janela se esgota o próximo objeto é transmitido e isso é repetido sucessivamente até o último objeto do carrossel. Uma vez que a última janela tenha se esgotado, o carrossel seleciona novamente o primeiro objeto e repete o ciclo de envio.

Afim de manter a similaridade com o padrão DSM-CC e permitir uma entrega confiável sem uso do TCP, foi definido e implementado o protocolo de verificação de integridade e de tipo de objetos.

A implementação da transmissão do carrossel de objetos do simulador utilizou bibliotecas Java para difusão de dados via rede (*DatagramSocket*) com base no protocolo UDP.

O sub-módulo "Protocolo" contém as estruturas necessárias de representação (cabeçalhos do protocolo) e conversão para a realização da transmissão. Suas classes recebem o objeto selecionado do carrossel e o transforma em um vetor bruto de *bytes*, só então o sub-módulo "Carrossel de dados" realiza a difusão dos dados na rede. O receptor deve realizar o processo contrário, transformando os *bytes* recebidos em objetos novamente.

VI. Desenvolvimento

Após a elaboração da arquitetura e protocolo do simulador foi dada continuidade no desenvolvimento da ferramenta definindo-se os requisitos funcionais básicos do sistema e, consequentemente, os casos de uso baseados nesses requisitos. Por fim, o simulador foi programado utilizando os requisitos e diagramas como insumos.

A. Requisitos Funcionais

De acordo com as necessidades de um simulador para envio de conteúdo extra e principal, os requisitos identificados para a ferramenta, considerando a arquitetura e conceitos adotados pela proposta de trabalho, foram:

1) *Configurar dados de rede*: Antes de qualquer ação o operador do sistema deve fornecer dados referentes ao meio de envio do conteúdo extra e principal, como o *IP* de *broadcast*, nome do canal e portas de envio (uma porta para envio do conteúdo principal e outra para o conteúdo extra).
2) *Enviar conteúdo principal*: Transmitir *stream* de um vídeo escolhido pelo operador da ferramenta pelo *IP* e portas definidos previamente na configuração dos dados de rede.
3) *Iniciar carrossel de objetos*: Transmitir os conteúdos extras selecionados pelo operador do simulador, de forma cíclica, utilizando *IP* e portas definidos previamente na configuração dos dados de rede.
4) *Pausar carrossel de objetos*: Interromper a transmissão do conteúdo extra até que o operador decida iniciá-lo novamente, voltando nesse caso a transmitir de onde parou.

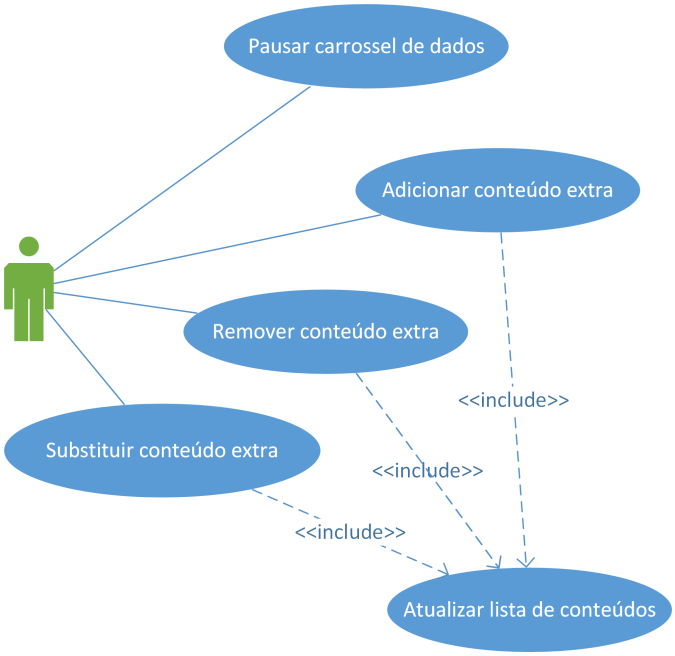

Fig. 6. Casos de uso do simulador de Broadcaster para TV Digital

5) *Reiniciar carrossel de objetos*: Interromper a transmissão e remover todo conteúdo contido no carrossel, iniciando-o novamente em seguida sem nenhum objeto em sua fila de transmissão.
6) *Adicionar conteúdo ao carrossel de objetos*: Adicionar um conteúdo ao carrossel para transmisão, exibindo-o em uma lista para o operador com os seguintes dados: (i) nome do conteúdo, (ii) versão e (iii) tipo do conteúdo. Está função pode ser acionada inclusive com o carrossel de objetos iniciado.
7) *Remover conteúdo do carrossel de objetos*: Remover o conteúdo selecionado do carrossel e da lista de exibição. Está função também é independente do estado do carrossel de objetos assim como a requisito "Adicionar conteúdo ao carrossel de objeto".
8) *Substituir conteúdo do carrossel de objetos*: Substituir o conteúdo do carrossel e atualizar a versão na lista de exibição. O número indicador da versão do conteúdo deve ser atualizado na lista de transmissão.

Os requisitos acima listados são as funções básicas do sistema e foram o insumo utilizado para a produção do diagrama de caso de uso. É importante salientar que os requisitos elaborados estão totalmente alinhados com a arquitetura proposta e com os conceitos básicos identificados como necessários para o desenvolvimento de um simulador para *broadcast* de AMC's.

B. Casos de Uso

Considerando os requisitos funcionais elaborados, os casos de uso foram extraídos e desenvolvidos de acordo com a Figuras 6 e 7. Dessa forma, foram identificados o total de dez casos uso que são explicados a seguir:

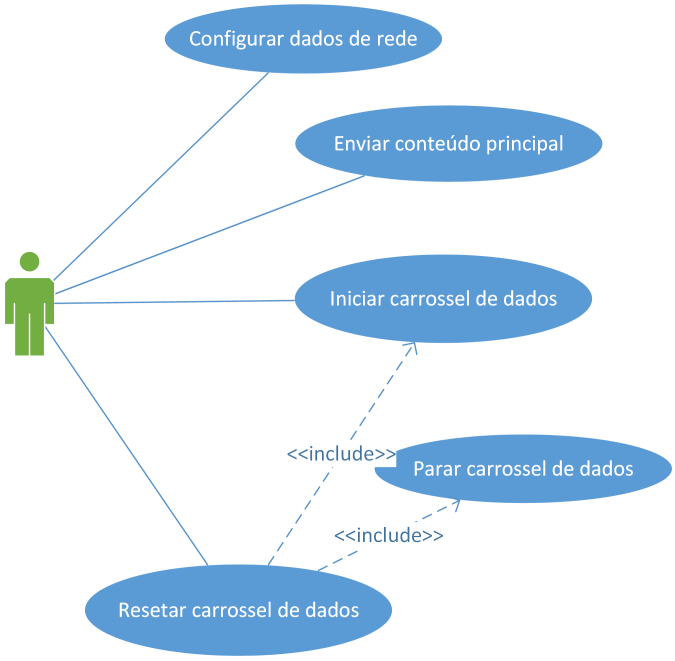

Fig. 7. Casos de uso do simulador de Broadcaster para TV Digital

1) *Configurar dados de rede*: Através de uma interface, que deve ser exibida ao iniciar o simulador, o operador deve entrar com os seguintes dados: (i) IP de broadcast, (ii) porta de envio de conteúdo principal, (iii) porta de envio de conteúdo extra e (iv) nome do canal. As outras funcionalidades do simulador só podem ser liberadas para uso após a configuração.
2) *Enviar conteúdo principal*: O operador deverá selecionar um arquivo de vídeo para ser transmitido como conteúdo principal. Para essa funcionalidade, o simulador deve utilizar a API VLCj que deve ser configurada para transmitir via UDP na porta e IP definidas anteriormente.
3) *Iniciar carrossel de objetos*: O operador poderá a qualquer momento iniciar o carrossel de objetos, mesmo que ainda não tenha adicionado nenhum conteúdo. Está opção deve ser desabilitada quando o carrossel estiver ativo.
4) *Parar carrossel de objetos*: Esta funcionalidade não deve ser ofertada ao operador, e deverá ser usada apenas pelo caso de uso abaixo. Ao parar o carrossel de dados, todos os conteúdos adicionados a ele devem ser removidos.
5) *Resetar carrossel de dados*: O operador poderá resetar o carrossel a qualquer momento, desde que o carrossel já esteja iniciado. Para ser resetado o carrossel deve ser parado, para que todo o conteúdo seja removido, e em seguida iniciado automaticamente.
6) *Pausar carrossel de dados*: Desde que o carrossel esteja iniciado o operador poderá pausa-lo a qualquer momento. Os conteúdos não são removidos, e ao ser reiniciado ele deve voltar a transmitir de onde parou.
7) *Atualizar lista de conteúdo*: Ações de inserção, remoção e substituição de conteúdo do carrossel de objetos devem atualizar de forma automática o componente gráfico do simulador onde é exibido os conteúdos extras que estão sendo transmitidos.
8) *Adicionar conteúdo extra*: O operador poderá adicionar conteúdo extra ao carrossel de dados a qualquer momento, ainda que o carrossel não tenha sido iniciado. Esta ação deve atualizar o componente gráfico onde são listados todos os conteúdos do carrossel. Ao adicionar um novo conteúdo no carrossel, caso seja uma aplicação, algumas informações devem ser informadas pelo operador, são elas: (i) nome da aplicação e (ii) tipo de execução da aplicação (*Automatic*, *Notify* ou *List*).
9) *Remover conteúdo extra*: O operador poderá remover o conteúdo extra do carrossel a qualquer momento. Esta ação deve atualizar o componente gráfico onde são listados todos os conteúdos do carrossel.
10) *Substituir conteúdo extra*: O operador poderá substituir a qualquer momento um conteúdo previamente adicionado ao carrossel. Esta ação deve atualizar a informação de sua versão no componente gráfico onde é exibido os conteúdos do carrossel.

No diagrama de caso de uso das Figuras 6 e 7 existe apenas um ator, o "Operador do sistema", que é responsável por interagir diretamente com a maior parte dos casos de uso. Observa-se ainda no diagrama que para algumas funcionalidades não há interação direta com o ator, como nos casos de uso "Atualizar lista de conteúdos" e "Parar carrossel de dados" . Essas funções sem interação direta com o operador possui o objetivo de completar um ou mais processos.

O Broadcaster não oferece a opção de parar o carrossel de dados para quem o opera, e uma vez que ele esteja rodando não pode ser mais parado de forma definitiva, entretanto caso seja necessário evoluir o sistema fornecendo esta função será apenas necessário criar a interação direta com o operador através da interface gráfica.

C. Implementação

A etapa de implementação do simulador teve como base os requisitos e diagramas anteriormente gerados. Foi utilizado para o desenvolvimento a linguagem de programação JAVA e um conjunto de suas bibliotecas (*Swing*, *DatagramSocket* entre outras) e a API VLCj para a difusão de *stream* de vídeo como conteúdo principal.

Para suprir as necessidades desejadas para simular um *broadcaster* para TV Digital, e assim entregar o que é proposto pela solução, foi necessário implementar a arquitetura e o protocolo de comunicação elaborados em sua totalidade e com todos os detalhes descritos nos capítulos anteriores.

Nesta fase do desenvolvimento é necessário destacar algumas características importantes na implementação que tornam o projeto uma solução viável para a simulação de um *broadcaster* para TV Digital, são elas:

1) *Envio cíclico de conteúdo extra*: A transmissão de conteúdo extra foi implementada com base no conceito de fila circular. Foi desenvolvida uma estrutura cíclica

denominada carrossel de objetos onde cada item desta estrutura representa um objeto do carrossel. Cada objeto da fila é transmitido em um determinado intervalo de tempo fixo e ao término de sua fatia de tempo o carrossel seleciona o próximo objeto. Este tipo abordagem permite contornar o problema da sintonização tardia, possibilitando que os telespectadores que sintonizem o canal após o início da transmissão possam receber os conteúdos extras da mesma forma. O carrossel gira em sentido único e cada objeto recebe uma marca indicativa de onde parou a sua transmissão possibilitando assim que no próximo ciclo do carrossel ele seja transmitido exatamente de onde estava na hora da interrupção.

2) *Tolerância a perda de pacotes*: Devido à natureza cíclica do carrossel, onde um objeto é enviado repetidamente (respeitando sua fatia de tempo) até que o operador da ferramenta remova-o do carrossel, o sistema de transmissão e recepção de conteúdo extra se torna tolerante a perda de pacotes já que mesmo perdendo um pacote basta o receptor esperar o próximo ciclo de envio para receber novamente o mesmo. Dessa forma, com o intuito de garantir maior velocidade, a transmissão é feita sobre o protocolo UDP, uma vez que o protocolo TCP possui um maior *overhead* [18] e não há a necessidade de garantir a entrega do pacote já que esse controle é feito via *software* (nos receptores).
3) *Envio de conteúdo principal em canal separado*: A transmissão do conteúdo principal foi desenvolvida com a utilização da API VLCj e para tal utiliza uma porta para conexão diferente da utilizada no envio de conteúdo extra, dessa forma não há multiplexação do conteúdo principal e extra da transmissão.

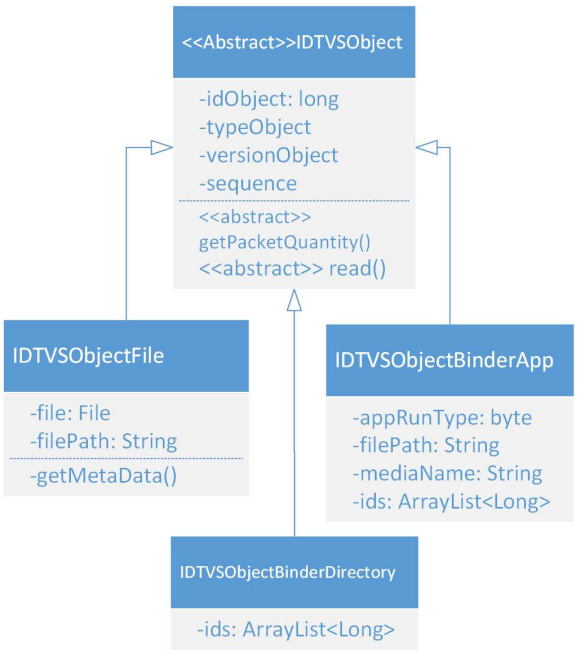

Fig. 8. Diagrama de classes: Objetos do carrossel

Com base na arquitetura elaborada para a solução, primeiramente foi desenvolvido o módulo "Gerador de Conteúdo Extra", seguido do módulo "Gerador de Conteúdo Principal" e por fim a construção da interface gráfica para operar o sistema.

Como pode ser observado na Figura 3 o módulo "'Gerador de Conteúdo extra" ainda pode ser dividio em mais três partes chamadas de "Gerador de objetos", "Carrossel de dados" e "Protocolo", cada um com suas funções e já explanados anteriormente.

Para a geração dos objetos do carrossel foram criadas quatro estruturas básicas (classes) que os representam. De acordo com o diagrama da Figura 8 essas classes são: (i) *IDTVSObject*, (ii)*IDTVSObjectFile*, (iii)*IDTVSObjectBinderApp* e (iv) *IDTVSObjectBinderDirectory*.

A classe *IDTVSObject* é uma classe abstrata que contém os atributos comuns a todos os quatro tipos distintos de objetos (*Single File*, *Binded File*, *Binder Application* e *Binder Directory*) e os seus atributos equivalem a um campo do cabeçalho do protocolo, como pode ser visto na Tabela IV.

Cabeçalho (Protocolo)	**Atributo(classe)**
ID (8 bytes)	long idObject
Type (2 bytes)	short typeObject
Version (2 bytes)	short versionObject
Number of Packets (8 bytes)	calculado pelo método getPacketQuantity()
Packet Sequence (8 bytes)	long sequence
Data Length (4 bytes)	calculado pelo própio conversor do protocolo

TABLE IV
ATRIBUTOS DA CLASSE IDTVSOBJECT

A classe *IDTVSObjectFile* representa tanto os objetos do tipo *Single File* quanto os do tipo *Binded File*, sendo diferenciados apenas pelo atributo "typeObject" da classe. A única diferença entre esses dois tipos de objetos é que os do primeiro tipo são liberados para consumo assim que são recepcionados, enquanto que no segundo tipo o seu consumo só é liberado depois que todos os outro objetos que compõe o conteúdo extra junto com ele são completamente recepcionados.

O sub-módulo gerador de objetos recebe como entrada o conteúdo selecionado pelo operador, então inicia uma busca recursiva no diretório base do conteúdo coletando todos os arquivos (incluído as subpastas) e mapeando-os em *IDTVSObjectFile*. No caso do conteúdo selecionado ser um diretório ainda é gerado mais um objeto do tipo *IDTVSObjectBinderDirectory* e caso seja uma aplicação o objeto gerado é um *IDTVSObjectBinderApp*.

O processo para criação de objetos para conteúdos do tipo "File" é mais simples já que não há que percorrer o diretório base e apenas um objeto (que o representa) do tipo *IDTVSObjectFile* é gerado.

Os objetos dos tipos *Binder Application* e *Binder Directory* são representados respectivamente pelas classes *IDTVSObjectBinderApp* e *IDTVSObjectBinder*. O objetivo dessas classes é realizar a composição da aplicação ou do diretório mantendo uma lista de "ids" dos objetos que compõe o conteúdo extra. Como pode ser visto no diagrama da Figura 8, o campo

"ids" das classes em questão é onde são armazenados os identificadores dos objetos citados.

Apesar do carrossel trabalhar com objetos individualmente, do ponto de vista do simulador um objeto não representa nenhum tipo de mídia consumível. O conteúdo extra de fato é o conjunto dos objetos gerados pelo sub-módulo "Gerador de objetos". Para manter o conjunto de objetos foi criada a classe *IDTVSObjectSet*, que é basicamente uma extensão da classe *ArrayList* do Java e que possui mais uma referência para o objeto principal do conteúdo extra.

No exemplo da Figura 5, se o conteúdo extra selecionado para a transmissão fosse a aplicação "MMA App" o gerador de objetos criaria três instâncias da classe *IDTVSObject-File* para representar os arquivos *MMA.class*, *fighter1.png* e *fighter2.png*, além de criar mais uma instância da classe *IDTVSObjectBinderApp* que manteria uma lista com os ids dos três primeiros objetos. Os quatro objetos criados seriam adicionados a uma instância do *IDTVSObjectSet* e então adicionado no carrossel no sub-módulo "Carrossel de dados".

É importante salientar que ao adicionar um conteúdo extra no carrossel de dados a versão do conteúdo deve inicialmente ser sempre "1", isso porquê a versão do conteúdo não necessariamente é a versão do arquivo (ou aplicação) adicionado. Esse campo serve como controle do conteúdo pelo Broadcaster e, principalmente, pelo receptor. Como o protocolo é cíclico é possível que o receptor receba um conteúdo mais de uma vez, nesses casos, o receptor simplesmente o descarta, porém, antes de descartar, o receptor deve verificar se a versão recebida é superior ao que ele já possui, caso seja, o receptor deve substituir o conteúdo e não descartá-lo.

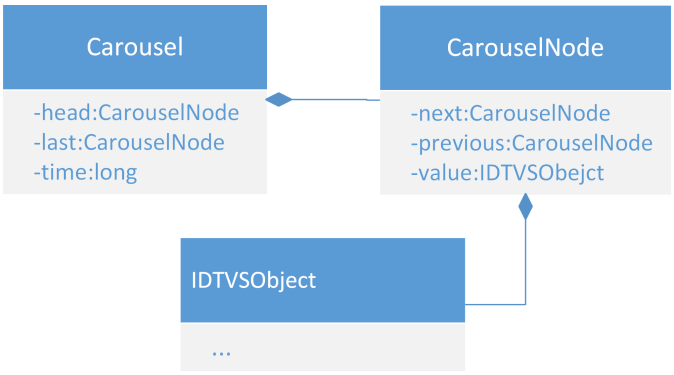

Fig. 9. Diagrama de classes: Carrossel de objetos

O Carrossel de objetos é representado pela classe *Carousel*, como pode ser visto no diagrama da Figura 9. Cada item do carrossel é representado por uma instância da classe *CarouselNode* que por sua vez encapsula um *IDTVSObeject*.

Cada nó do carrossel possui uma referência para o próximo *CarouselNode* e outra para o anterior, além da referência ao objeto (*IDTVSObject*) que ele encapsula.

Um objeto é sempre adicionado no final da fila. Quando o carrossel recebe um objeto para ser inserido, é criado então uma instância da classe *CarouselNode*, que por sua vez recebe o *IDTVSObeject*. Em seguida a nova instância de *CarouselNode* aponta para o item que era o último da fila (campo *previous*) e para o primeiro (campo *next*). Toda vez que um novo objeto é adicionado ao carrossel o campo *last* da classe *Carousel* é atualizado para o novo valor inserido.

Quando iniciado o carrossel começa o processo de seleção dos objetos para transmissão. De acordo com protocolo desenvolvido os dados dos objetos são transformados em pacotes. Para realizar essa tradução foram criadas algumas estruturas no sub-módulo "Protocolo" para apoiar nessa tarefa e descentralizar código.

A classe *Protocol* descreve o protocolo com seus campos e tamanhos em *bytes*. Já a classe *ProtocolPacket* é a representação de um pacote de acordo com o protocolo, enquanto a classe *ProtocolConvert* é um conversor de *ProtocolPacket* em vetor de *bytes*.

Ao selecionar um objeto, o carrossel o traduz para um *ProtocolPacket* fazendo um pareamento dos atributos do objeto com os atributos do pacote (id, tipo, versão, quantidade total de pacotes necessários para transferir o objeto, o número do pacote que está sendo transmito no exato momento, quantidade de *bytes* utilizado na área de dados e os dados para transmissão).

Muitas vezes um objeto precisa de muitos pacotes para ser transmitido (normalmente os que representam arquivos) já que o tamanho do pacote é limitado pelo protocolo e arquivos de mídia costumam ser grandes. No caso de objetos do tipo *Bider Application* e *Binder Directory* apenas um pacote é necessário já que esses objetos apenas carregam informações de estruturação e composição do conteúdo extra.

Após a tradução de um objeto em pacote o carrossel faz uma chamada a classe *ProtocolPacket* passando o objeto e recebendo um vetor de *bytes* pronto para ser transmitido. Depois da difusão o carrossel repete o processo pegando o próximo pacote do objeto selecionado. Caso o tempo esgote outro objeto é eleito e o processo se repete.

Uma vez realizada a implementação do "Gerador de conteúdo extra" foi iniciado o desenvolvimento do módulo "Gerador de conteúdo principal". Antes da transmissão o arquivo de vídeo que deseja-se utilizar como conteúdo principal deve ser selecionado pelo operador do sistema e passado para o sub-módulo. A partir da API *VLCj*, que possui todas as estruturas necessárias para transmissão e codificação de mídias, é realizada a transmissão do vídeo via *stream*.

A interface gráfica do simulador foi desenvolvida a partir da biblioteca de componentes gráficos *Swing*, nativa do Java, seguindo os requisitos funcionais e os casos de uso para garantir a usabilidade desejada para a ferramenta.

VII. A Ferramenta

Com base nos requisitos e casos de uso apresentados foi desenvolvido o simulador como pode ser visto nas Figuras 10 e 11. Para melhor apresentação a ferramenta foi dividia em três partes básicas: (i) Tela de configuração, (ii)Transmissor de conteúdo principal e (iii) Transmissor de conteúdo extra.

A. Tela de configuração

Ao iniciar o simulador é exibido primeiramente uma interface (*popup*) de configuração que bloqueia a tela principal do Broadcaster até que todos os dados sejam preenchidos e confirmados. Antes dessa etapa é impossível iniciar a transmissão de conteúdo, uma vez que os dados exigidos são obrigatórios para a realização da difusão dos dados na rede.

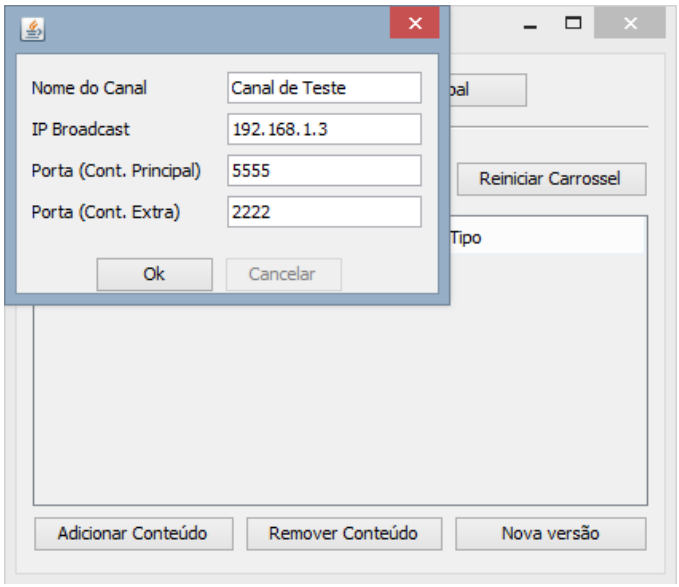

Fig. 10. Interface de configuração

Todos os campos do formulário da interface de configuração são de preenchimento obrigatório e já são exibidos com um valor padrão. De acordo com a Figura 10 os campos para configuração do Broadcaster e seus valores padrão são:

1) *Nome do Canal*: Nome atribuído ao canal de transmissão. Este dado é apenas para questão de informação e não influencia nenhuma função da ferramenta, ainda assim é obrigatório. Seu valor padrão é "Canal de Teste".
2) *IP de Broadcast*: IP para *broadcast* do conteúdo principal e extra na rede. A pesar do objeto ser realizar difusão em *broadcast* o simulador aceita um IP que não seja o IP de *broadcast* da rede. Seu valor padrão é "127.0.0.1".
3) *Porta (conteúdo principal)*: Porta para transmissão do conteúdo principal. Seu valor padrão é "5555".
4) *Porta (conteúdo extra)*: Porta para transmissão do conteúdo extra (deve ser um valor diferente da porta para conteúdo principal). Seu valor padrão é "2222".

Após o preenchimento dos dados pelo operador do sistema basta clicar o botão "ok" na parte inferior da tela para confirmar. Se algum dos dados inseridos for inválido ou esteja faltando será exibida uma mensagem pedindo o preenchimento correto do(s) campo(s) em questão, caso contrário a interface de configuração será fechada e o Broadcaster estará pronto para uso.

B. Transmissor de conteúdo principal

Após a configuração do simulador, a tela principal do sistema é desbloqueada permitindo ao operador inciar o envio de conteúdo principal e extra.

Para enviar conteúdo principal, o operador deve clicar no botão "Enviar conteúdo principal" situado ao lado esquerdo da área da tela que manipula o conteúdo principal. Como pode ser visto na Figura 11 será aberto em seguida um *JFileChooser* para ser escolhido o arquivo de vídeo que deseja-se transmitir como conteúdo principal do canal.

Após a escolha do conteúdo a transmissão é iniciada automaticamente através do IP e porta informados na tela anterior de configuração. O nome do arquivo que está sendo transmitido pelo simulador é exibido do lado direito do botão "Enviar conteúdo principal". Se o operador da ferramente decidir trocar o conteúdo basta clicar no botão e escolher o novo conteúdo, nesse caso a transmissão do vídeo atual é interrompida e inicia-se a transmissão do novo vídeo escolhido.

C. Transmissor de conteúdo extra

Na tela principal, logo abaixo da área de manipulação do conteúdo principal, encontra-se a área de manipulação do conteúdo extra. Essa área da tela principal ainda pode ser subdivida em três partes: (i) botões de gerenciamento do estado do carrossel, (ii) tabela de conteúdos sendo transmitidos e (iii) botões de gerenciamento dos conteúdos transmitidos.

A transmissão do conteúdo extra é feita utilizando um carrossel de objetos implementado através de uma fila circular. Ao iniciar o simulador o carrossel de objetos está parado (que é seu estado inicial padrão). Ao clicar no botão "Iniciar Carrossel" (Figura 11) o carrossel de objetos sai da inércia e começa a girar transmitindo os conteúdos selecionados pelo IP e portas definidos na tela anterior de configuração.

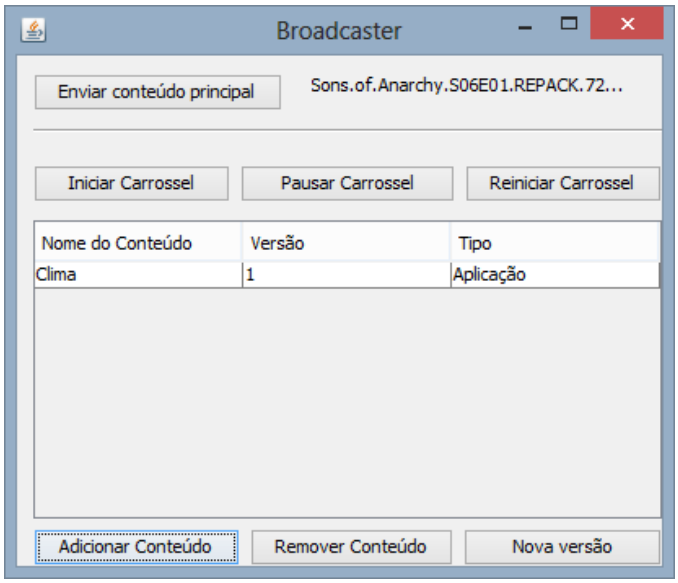

Fig. 11. Interface pricipal do Broadcaster

A seguir será explicado cada elemento da tela do simulador que altera o estado do carrossel de objetos:

1) *Botão Iniciar Carrossel*: Inicia o carrossel removendo-o dos estados de "Parado" ou "Pausado", colocando-o no estado de "Iniciado". Uma vez acionado, o botão é desabilitado. Se o estado atual do carrossel for "Parado", quando o botão "Iniciar" for acionado, o carrossel deve iniciar a transmissão a partir do primeiro item da fila, caso o estado seja "Pausado" o carrossel é reiniciado de onde havia sido interrompido.
2) *Botão Pausar Carrossel*: Pausa o carrossel colocando-o no estado de "Pausado" e só permanece habilitado quando o estado do carrossel é "Iniciado". Quando acionado interrompe a transmissão dos conteúdos porém salva o estado do mesmo, desta forma possibilitando que o carrossel seja reiniciado de onde parou.
3) *Botão Resetar Carrossel*: Reinicia o carrossel de objetos ao ser acionado removendo todos os objetos da fila e mantendo o estado de "Iniciado". Todos os objetos antes adicionados ao carrossel são descartados, porém é possível adicionar novos elementos uma vez que o carrossel ainda está em execução. Enquanto o estado do carrossel for "Iniciado" o botão permanece habilitado.

Na tela principal do simulador, de acordo com a Figura 11, entre os botões de controle está situado a tabela que lista os conteúdos extras que estão inseridos no carrossel de objetos. Os conteúdos do carrossel são listados em ordem de inserção. Na tabela é possível visualizar três colunas: (i) Nome do conteúdo, (ii) Versão e (iii) Tipo.

1) *Tipo*: Nessa coluna são exibidos os valores "Arquivo", "Diretório" e "Aplicação" para os respectivos tipos de conteúdo "File", "Directory" e "Application".
2) *Versão*: Nessa caluna é mostrada a versão do conteúdo no carrossel. Um novo conteúdo sempre é inserido com número de versão "1", com exceção para os conteúdos do tipo "Directory" que não possuem versão e recebem o valor "0". O valor dessa coluna pode ser alterado caso o operador opte por substituir o conteúdo do carrossel por um outro, nesse caso o número da versão é sempre incrementado em mais uma unidade, mais uma vez esta ação não se aplica para o tipo "Directory".
3) *Nome do conteúdo*: Se o conteúdo inserido for do tipo "File" o nome do conteúdo será o nome do próprio arquivo escolhido, caso seja do tipo "Directory" será preenchido com o nome do diretório em questão, sendo o conteúdo do tipo "Application" o nome fica à escolha do operador, somente neste último caso é que o operador pode escolher o nome do conteúdo.

É importante lembrar que quando o operador clica no botão "Resetar carrossel" todos os objetos são excluídos da fila de transmissão e por consequência da tabela de conteúdos extra.

Logo abaixo da tabela de conteúdos extra está localizado os botões que permitem adicionar um novo conteúdo ao carrossel, remover um conteúdo e substituir um conteúdo por um novo.

Clicando no botão "Adicionar Conteúdo", mais a esquerda da tela, como pode ser visto na Figura 11, um *JFileChooser* é aberto para que o operador da ferramenta escolha o conteúdo no seu computador que deseja adicionar ao carrossel de objetos para transmissão. Ao adicionar um novo conteúdo o mesmo deve ser incluído na tabela de conteúdos extra. Caso o conteúdo escolhido seja um aplicativo uma caixa de diálogo requisitando o nome do aplicativo será aberto, sendo o conteúdo um arquivo ou uma pasta os seus respectivos nomes atribuídos serão os mesmos do sistema de arquivo do computador. Quando um novo conteúdo extra é adicionado ao carrossel é atribuído a ele um número de versão que por padrão é sempre "1", com exceção para os diretórios que não possuem versão.

Para remover um conteúdo do carrossel o operador deve selecionar um dos itens listados na tabela de conteúdo extra e em seguida clicar no botão "Remover Conteúdo", dessa forma o mesmo é removido da fila de transmissão e consequentemente da tabela de conteúdo extra.

O operador do Broadcaster ainda tem a possibilidade de substituir o conteúdo extra por uma outra versão. Clicando no botão "Nova versão", tendo previamente selecionado o item desejado na tabela, será aberto um *JFileChooser* de forma similar ao procedimento de adicionar novo conteúdo, é obrigatório que o novo item selecionado possua o mesmo nome e extensão (no caso de arquivo) do antigo para que o sistema aceita a substituição.

Ao realizar a substituição de um conteúdo extra um novo número de versão é atribuído a ele, sempre somando uma unidade ao número da versão antiga. A nova versão do item é atualizada na tabela de conteúdo.

VIII. Testes e Resultados

A ferramenta apresentada neste trabalho consiste em um simulador broadcaster de conteúdo para um ambiente de TV, logo é de grande importância, e faz-se necessário, que para validar a solução seja realizado testes com uma ferramenta que componha o lado receptor completando dessa forma o ambiente de transmissão.

Para testar o funcionamento do Broadcaster foi planejado realizar um teste inicialmente para validação da transmissão do conteúdo principal utilizando o VLC Player e em seguida o teste da transmissão do conteúdo extra com uma ferramenta de recepção que implementa o protocolo aqui apresentado. Assim, esse capítulo está dividido em duas partes, a primeira relatando os testes de conteúdo principal com o VLC e a segunda com os testes utilizando uma outra ferramenta para recepção e execução de conteúdo extra.

A. Testes do Gerador de conteúdo principal

A implementação do módulo de transmissão de conteúdo principal foi realizada através da biblioteca libvlc e API VLCj pertencentes ao projeto Vídeo LAN (VLC). Por esse motivo foi escolhido o próprio tocador VLC para o teste de transmissão do conteúdo principal.

O *player* VLC possui a opção de configurar um receptor de fluxo na rede, dessa forma é possível capturar e exibir o

conteúdo principal transmitido pelo Broadcaster. No VLC, a partir do menu "Mídia >Abrir Fluxo de Rede", é exibida uma nova tela onde o usuário pode entrar com o protocolo, IP e porta para receber a *stream* via rede e exibir no tocador.

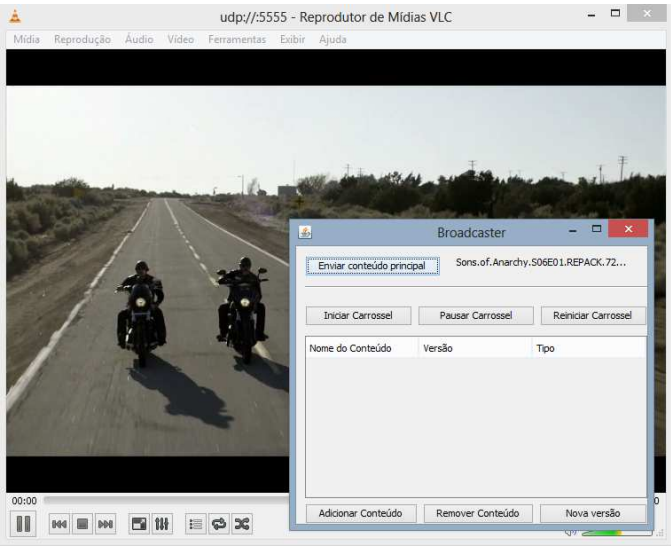

Fig. 12. Teste de conteúdo principal com o tocador VLC

Como pode ser visto na Figura 12, o simulador broadcaster está transmitindo um vídeo como conteúdo principal previamente escolhido e o VLC o está exibindo de forma sincronizada, ou seja, no ponto do vídeo no momento da sintonização e não do começo do mesmo uma vez que, assim como acontece normalmente em um ambiente de TV real, a transmissão de conteúdo principal não é armazenada no receptor. É possível verificar então que a implementação do gerador de conteúdo principal a partir das bibliotecas e APIs do VLC correspondem ao que foi planejado.

B. Testes do Gerador de conteúdo extra

Para realizar os testes, a fim de verificar o funcionamento da transmissão do conteúdo extra, foi utilizado o receptor apresentado em [19] que implementa o protocolo definido em capítulo anterior deste trabalho.

Em [19] é apresentado um simulador de recepção para TV Digital capaz de sintonizar um canal (um endereço IP e porta na rede) e exibir conteúdo extra e principal provenientes de um emissor. Assim como o broadcaster, aqui implementado, o receptor utiliza portas diferentes para receber dados referentes aos conteúdos extra e principal. Outra caraterística semelhante é a utilização da biblioteca libvlc e a API VLCj para implementação da recepção da *stream* de vídeo do conteúdo principal.

O receptor de [19] foi implementado utilizando como base o XletView, uma ferramenta capaz de simular a execução de aplicativos (*xlets*) para TV Digital. Para se adequar a necessidade do ambiente desejado foi desenvolvido, além da recepção do conteúdo principal, recepção do conteúdo extra por rede TCP/IP através de um carrossel de dados implementando o protocolo definido neste trabalho.

Os aplicativos utilizados para o teste do simulador também foram apresentados em [19], são dois *xlets* desenvolvidos exclusivamente para testar o ambiente de transmissão, um aplicativo para *replay* de momentos de uma corrida de Fórmula 1 e um outro aplicativo para informações sobre previsão do tempo.

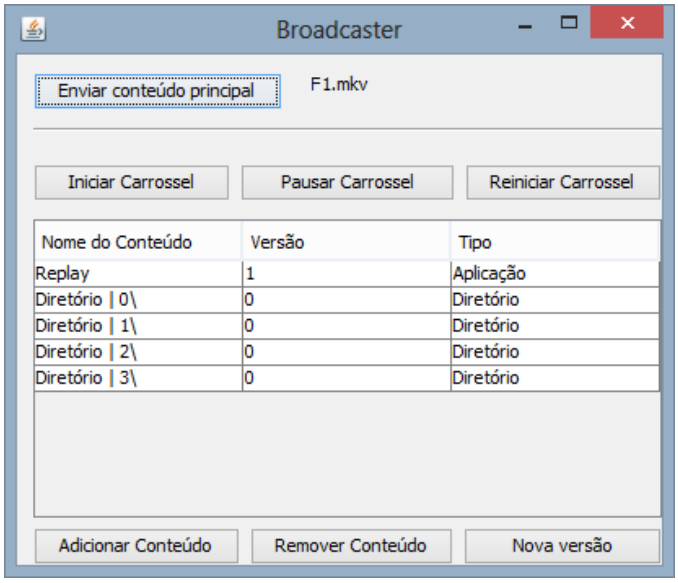

Fig. 13. Broadcaster transmitindo aplicativo de replay e os conteúdos para consumo

O *xlet* para reprise de Formula 1 consome dados enviados pelo próprio broadcaster e para isso é necessário adicionar ao carrossel um diretório que contém um arquivo de vídeo (reprise) e um arquivo XML com metadados referentes ao vídeo em questão. Ao ser recebido no receptor o diretório é então liberado para consumo da aplicação que por sua vez lista o vídeo da reprise para que o telespectador possa assistir através da aplicação.

O roteiro seguido para execução dos testes referente a aplicação de *replay* foi:

1) Adicionar conteúdo principal
2) Adicionar o aplicativo ao carrossel de dados
3) Adicionar os *replays* ao carrossel
4) Avaliar a execução do aplicativo e dos replays no lado receptor

A Figura 13 mostra o broadcaster com o aplicativo de *replay* adicionado ao carrossel de objetos assim como os diretórios com os vídeos de reprise e descritores (XML) para o consumo do aplicativo. Já na Figura 14 é possível observar a execução do *xlet* no receptor. No momento do *screenshot* os vídeos com as reprises já haviam sidos recepcionados e estavam listados no aplicativo. O conteúdo principal exibido também estava sendo transmitido pelo *broadcaster*.

O aplicativo de previsão do tempo tem a característica de consumir dados tanto do *broadcaster* quanto de um *provider* externo. Para consumo de informações provenientes do *broadcaster* a aplicação espera um arquivo XML de nome

Fig. 14. Aplicativo de Replay de momentos de uma corrida

"clima.xml" onde deve conter as informações sobre cidades e temperaturas. Já para as informações em um provedor externo o aplicativo é configurado para se conectar pela *Internet* e obter o arquivo XML da fonte desejada.

O teste da aplicação de previsão do tempo seguiu roteiro semelhante ao de *replay*:

1) Adicionar conteúdo principal
2) Adicionar o aplicativo ao carrossel de dados
3) Adicionar XML com cidades e temperatura ao carrossel
4) Avaliar a execução do aplicativo e e exibição das temperaturas do receptor

Fig. 15. Aplicativo de previsão do tempo

A Figura 15 mostra o aplicativo de previsão em execução no receptor, exibindo as cidades e suas respectivas temperaturas recebidas tanto do próprio broadcaster quanto de um provedor externo. Neste caso de uso foi adicionado para transmissão no carrossel do brodcaster o *xlet* de previsão do tempo e mais o XML com as informações das cidades apresentadas.

É importante salientar que os testes foram realizados sintonizando o receptor antes e após a inclusão dos aplicativos e mídias no carrossel de dados com o objetivo de comprovar que a ferramenta contorna o problema da sintonização tardia. De acordo com os testes realizados, e os resultados obtidos, é possível afirmar que o simulador cumpre o papel desejado, realizando a transmissão de conteúdo principal e extra em um ambiente virtual de TV Digital.

IX. Conclusão e Trabalhos Futuros

Este trabalho apresentou uma ferramenta para transmissão em *broadcast* de conteúdo principal e conteúdo extra, principalmente AMC mas não se limitando a esse tipo de conteúdo. A ferramenta desenvolvida tem o objetivo principal de simular um emissor *brodcaster* de um ambiente de TV real porém utilizando rede TCP/IP para difusão de vídeo, áudio e dados (mídias, aplicativos e etc.). Com essas características as principais contribuições do simulador são:

1) Redução de custos já que não é necessário manter um ambiente real de transmissão para realização de testes e simulações das aplicações desenvolvidas;
2) Segurança, uma vez que os testes são realizados em um ambiente próprio para simulação reduzindo a possibilidade de causar danos ao ambiente de produção ocasionados por erros e problemas de desenvolvimento.
3) Possibilitar a execução de AMC's usando computadores conectados em redes TCP/IP;
4) Aumentar a fidelidade dos testes para esta categoria de aplicativos e;
5) Possibilidade de adaptação do simulador para outros contextos, como a IPTV que também está estruturado em redes TCP/IP.

Para a implementação do simulador foi inicialmente definida a sua arquitetura e desenvolvido um protocolo para verificação de objetos e consistência baseado no DSM-CC. A escolha pelo desenvolvimento de um novo protocolo baseado no DSM-CC, porém menor e com menos recursos, se deu devido a grande extensão e abrangência do protocolo em questão.

É importante lembrar que o *broadcaster* é uma parte de um ambiente maior de simulação, que consiste em sua totalidade de mais um receptor e também *providers*. Dessa forma, a contribuição do *broadcaster* para esse ambiente é somente a transmissão do conteúdo, semelhante as emissoras de TV, não agregando as etapas de produção e recepção.

Devido a extensão e complexidade do DSCM-CC, como dito anteriormente, foi implementado um novo protocolo baseado no DSM-CC, porém menor e com menos recursos. Já que o propósito principal do trabalho é a simulação da transmissão de conteúdos para TV Digital este fato não foi considerado um empecilho.

Para agregar mais fidelidade com o ambiente de transmissão real, uma sugestão de trabalho futuro é a implementação do DSM-CC que é o padrão de carrossel adotado pela maioria dos sistemas de TV Digital do mundo. O trabalho de desenvolvimento do DSM-CC ainda pode ser fragmentado em algumas etapas de maneira que possa atender aos requisitos do simulador mesmo que implementado parcialmente:

1) A primeira etapa seria a implementação do carrossel *User to Network* para difusão em *broadcaster*, que ainda pode ser subdividida em mais duas:
   a) Carrossel de Dados para dados não estruturados e;
   b) Carrossel de Objetos para dados estruturados e sistemas de arquivos.
2) Uma segunda etapa para implementação do carrossel *User to User* para um provável canal de retorno.

Após a implementação do protocolo DSM-CC, o simulador além de oferecer a possibilidade de transmissão em *broadcast* de áudio, vídeo e dados ainda o fará utilizando o protocolo padrão estabelecido, aumentando ainda mais a fidelidade na simulação.

REFERENCES

[1] M. C. M. Neto, "Contribuições para a modelagem de aplicações multimidia em tv digital interativa," 2011.

[2] M. C. Marques Neto and C. A. S. Santos, "Storytocode: a model based on components for specifying interactive digital tv convergent applications," in *Proceedings of the XV Brazilian Symposium on Multimedia and the Web*, ser. WebMedia '09. New York, NY, USA: ACM, 2009, pp. 8:1–8:8. [Online]. Available: http://doi.acm.org/10.1145/1858477.1858485

[3] R. Kulesza, J. Ferreira, S. Filho, A. Livio, R. Brandao, J. Araujo, and G. Filho, "Ginga-j: Implementação de referencia do ambiente imperativo do middleware ginga," *WebMedia*, 2010.

[4] M. K. Zuffo, "Tv digital aberta no brasil: Politicas estruturais para um modelo nacional," 2003.

[5] M. Laureano, *Maquinas Viruais e Emuladores Conceitos, Tecnicas e Aplicacoes*. Novatec, 2006.

[6] M. Marques, T. Rodrigues, F. Machado, R. Kulesza, and C. A. S. Santos, "Um ambiente de simulação para aplicações multimídia complexas," in *Webmedia 2013 XII Workshop de Ferramentas e Aplicações*. SBC, 2013, pp. 61–64.

[7] M. C. Marques Neto and C. A. Santos, "An event-based model for interactive live tv shows," in *Proceedings of the 16th ACM international conference on Multimedia*, ser. MM '08. New York, NY, USA: ACM, 2008, pp. 845–848. [Online]. Available: http://doi.acm.org/10.1145/1459359.1459502

[8] G. D. Ferreira, G. D. Ferreira, G. Nogueira, M. Martinello, G. Comarela, and M. Martinello, "Ginga-ncl em dispositivos portateis: Uma implementacao para a plataforma android," 2010.

[9] M. Moreno, R. Rodrigues, and L. Soares, "Mecanismo de identificação de recursos para aplicações interativas em redes de tv digital por difusão," *SBRC 2007 - VoIP e TV Digital*, 2007.

[10] ISO/IEC, "ISO/IEC 13818-6:1998/Cor 1:1999," Padrão, 1999.

[11] W. I. Grosky, C. Zhang, and S.-C. Chen, "Intelligent and pervasive multimedia systems," *MultiMedia, IEEE*, vol. 16, no. 1, pp. 14 –15, jan.-march 2009.

[12] S. Soursos and N. Doulamis, "Connected tv and beyond," in *Consumer Communications and Networking Conference (CCNC), 2012 IEEE*, jan. 2012, pp. 582 –586.

[13] L. E. C. Leite, G. L. de Souza Filho, S. R. de Lemos Meira, P. C. T. de Arújo, J. F. de A. Lima, and S. M. Filho, "A component model proposal for embedded systems and its use to add reconfiguration capabilities to the flextv middleware," in *WebMedia '06*. New York, NY, USA: ACM, 2006, pp. 203–212.

[14] G. Coulouris, T. Kindberg, and J. Dollimore, *Distributed Systems: Concepts and Design*. Addison Wesley, 2005, vol. 4.

[15] M. Weiser, "The computer for the 21st century," *Scientific American*, September 1991.

[16] H. Zhang, T. Jiang, Z. Gu, and S. Zheng, "Design and implementation of broadcast file system based on dsm-cc data carousel protocol." *IEEE Trans. Consumer Electronics*, vol. 50, no. 3, pp. 929–933, 2004.

[17] D. Terra, N. Kumar, N. Lourenco, L. N. Alves, and R. L. Aguiar, "Design, development and performance analysis of dsss-based transceiver for vlc," in *EUROCON'11*, 2011, pp. 1–4.

[18] A. S. Tanenbaum, *Redes de Computadores*. Campus Elsevier, 2003, vol. 4.

[19] T. Rodrigues, R. Kulesza, M. Marques, F. Machado, and C. Santos, "Um ambiente de simulação para aplicações multimidia complexas," *WebMedia*, 2013.