Um Ambiente de Simulação para Aplicações Multimídia Complexas

Felipe A.L. Machado
GSORT - IFBA
Rua Emídio Santos, s/n,
Salvador - BA
Email: felipealmeida@ifba.edu.br

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

***Resumo*—This paper aims to present a simulation environment that can be used to run interactive applications developed in JavaDTV, focused on digital television. The main objective of this simulation environment is provide a complete experience executing all categories of aplications with focus on aplications of live generated content. The proposed environment differs from others because it allows to simulate reception of main content , extra content and execute JavaDTV applications to allow extra content presentation.**

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

I. Introdução

O crescimento do poder de processamento dos diferentes tipos de dispositivos (celulares, PDAs, computadores e etc.) e o aumento da largura da banda das redes são fatores essenciais para a convergência digital da comunicação e informações. Essa convergência resulta na produção de um novo tipo de permutabilidade e interconectividade entre tipos diferentes de mídias [24].

Como consequência, mudanças relacionadas à mídia e aos veículos de comunicação vêm ocorrendo. A Internet e as tecnologias de comunicação móvel têm fornecido diversos serviços digitais aos usuários. A televisão, por exemplo, está seguindo esta tendência por meio do processo de digitalização que ocorre em toda a cadeia, desde a produção, edição e transmissão, até a recepção do conteúdo [25].

A televisão é um importante meio de comunicação, presente no cotidiano da maioria das pessoas em todo o mundo. Com o desenvolvimento e aprimoramento das tecnologias, principalmente, no que tange as tecnologias digitais, a sociedade vem presenciando o surgimento de um novo paradigma para o sistema de televisão. [4].

Com o fenômeno da Televisão Digital Interativa (TVDI), um leque de novas possibilidades surgiram, a televisão passa a ter uma qualidade de imagem e som bastante superior àquela oferecida pela televisão tradicional e além disso são incorporadas propriedades interativas que permitem novas experiências para o usuário. [1].

Por meio de programas de TVDI surgiu a possibilidade de transmissão de softwares interativos e personalizados junto com qualquer outro programa da grade de uma emissora. [2] [3]. Um dos pontos fundamentais para desenvolvimento desses softwares é a existência de um ambiente que permita simular o seu funcionamento antes da sua implantação no ambiente real [13].

Entre os principais motivos para utilização de ambientes para simulação de softwares estão a redução de custo de compra dos ambientes reais, aumento da velocidade de desenvolvimento e a segurança, pois a aplicação em teste só pode causar danos a um ambiente virtual. [13].

Os simuladores [13] de programas de TV digital interativos comumente utilizados em ambientes de desenvolvimento são o XletView [16], Ginga-NCL Virtual Box [14] e o Emulador Ginga-J [9]. Estes ambientes de simulação permitem executar aplicações interativas de Televisão digital a partir do uso de computadores simulando o receptor de TV. Por outro lado, eles não contemplam na recepção de conteúdo extra utilizando redes TCP/IP [23].

O fato dos simuladores encontrados não oferecerem a possibilidade de recepção de conteúdo extra utilizando redes de comunicação é um impeditivo para a execução de uma categoria de aplicações interativas onde seu conteúdo tem três características importantes: (1) possuem relação com a semântica do conteúdo principal ; (2) são transmitidos via fluxos alternativos; e (3) são exibidos de forma sincronizada [4]. Esta categoria de aplicações multimídia são denominadas de aplicações multimídia complexas (AMC) [23].

O objetivo deste trabalho é apresentar uma infraestrutura de simulação de um receptor de TVDI onde é possível receber conteúdo extra e principal através de uma rede TCP/IP e executar aplicações JDTV, que também serão recebidas através da rede de comunicação na forma de conteúdo extra. Com isso este ambiente deve permitir a execução de AMC para TV Digital nas suas duas modalidades de interatividade (aberta e conectada). É importante ressaltar que este ambiente não contempla a simulação da fase de produção de conteúdos.

Para validar o funcionamento e demonstrar as principais funcionalidades do simulador foram desenvolvidas duas aplicações JDTV (Xlets). Elas permitem apresentar durante a transmissão de uma corrida de F1 replays enviados por um Broadcaster e a previsão do tempo obtida em um provedor de conteúdos na Internet. Os vídeos com estas aplicações em execução podem ser vistos no YouTube [1].

[1]http://youtu.be/lJ6ZPxgf-CI http://youtu.be/VRg6149XesU

Este artigo foi dividido da seguinte forma: na primeira seção são apresentados os conceitos da Televisão Digital Interativa, na segunda seção são apresentadas as classificações das aplicações para TVDI e o problema da sintonização tardia, na terceira seção é feito um paralelo entre TVDI e computação ubíqua, na quarta seção são apresentados os trabalho correlatos, na quinta seção é apresentado o funcionamento do ambiente de simulação construído e em seguida sua arquitetura (sexta seção), na sétima, oitava e nona seções são apresentados estudos de caso utilizando o ambiente de simulação, na décima seção é feita a conclusão do artigo e por fim são apresentados os trabalhos futuros.

II. Televisão Digital Interativa

Diante do surgimento da Televisão Digital Interativa (TVDI), nos vemos rodeados de possibilidades que ultrapassam àquelas proporcionadas pela TV Convencional. Além da alta definição de imagem e som, são incorporadas propriedades interativas que permitem novas experiências para o usuário. Outros importantes recursos fornecidos pela TVD estão relacionados à mobilidade, que possibilita a recepção do sinal digital por aparelhos em movimento, e a portabilidade, pois a televisão passa a ser pervasiva, onde a mesma pode estar em qualquer lugar que haja a presença de uma tela para exibição [1].

Os "programas de TV Digital Interativos" possibilitam a veiculação de softwares e dados aos fluxos de áudio e vídeo transmitidos pelas emissoras. Este fato apresenta-se como uma de suas maiores inovações, visto que proporciona uma vasta gama de oportunidades para que novos serviços e aplicações sejam desenvolvidas e agregados a esta mídia interativa [2] [3] [4]. Essas características são semelhantes com as encontradas em sistemas ubíquos onde softwares diferentes podem ser executados em plataformas diferentes e muitas vezes não convencionais (ex.: TV) [5].

Com um novo modelo televisivo (TVDI) também surgiu um novo modelo de produção de conteúdo que é caracterizado por um novo processo, não linear, interativo, ágil, multidisciplinar e convergente [4]. Este modelo de produção tem em sua arquitetura básica a presença de seis elementos: Ilha de Edição e Aplicativos, Multiplexador, Modulador, Set-Top-Box e TVD, Dispositivos e Canal de Retorno e Internet [4]. Nos tópicos abaixo cada um destes elementos serão detalhados.

1) Ilha de Edição e Aplicativos: Responsável pela produção de conteúdo, tanto principal (áudio e vídeo) como extra (aplicações) [4].

2) Multiplexador: Responsável por fundir um ou mais fluxos de dados aos fluxos de áudio e vídeo, compondo os eventos e programas, que fazem parte dos serviços consumidos pela audiência [11] e integram diversos elementos em um conteúdo audiovisual;

3) Modulador: Responsável pela transmissão de todo conteúdo sobre um canal controlado por uma emissora [4].

4) Set-Top-Box e DTV: Responsável por sintonizar um canal da emissora, decodificar o sinal e apresentar todo conteúdo recebido de forma sincronizada [4].

5) Dispositivos: Responsável por proporcionar interatividade entre o conteúdo e o usuário-telespectador. Tais dispositivos são conectados ao receptor tais como: controle remoto, celulares e etc [4].

6) Canal de Retorno e Internet: Canais de interatividade que possibilitam acessar conteúdos oriundos de outros geradores como portais de notícias, servidores de vídeo, entre outros [4].

Para que todos os elementos da cadeia de produção, distribuição e consumo de conteúdos audiovisuais interativos para televisão digital operem perfeitamente, é necessário que exista uma padronização comum entre os elementos. Diante desta situação o Sistema Brasileiro de TV Digital (SBTVD ou ISDB-Tb) especifica vários padrões de referência, dentre os quais a camada de middleware Ginga é, sem dúvida, a mais importante diferença do SBTVD com relação aos outros padrões internacionais [4].

O objetivo principal de um middleware de TV Digital é agir como uma camada de software responsável por intermediar a comunicação entre o sistema operacional do receptor e as aplicações que operam sobre o mesmo, abstraindo as características específicas de cada plataforma e provendo uma série de serviços específicos para camadas superiores, permitindo a execução de aplicações portáteis para dispositivos com capacidades distintas de processamento e arquiteturas de hardware [9].

O middleware Ginga foi desenvolvido pela PUC-RJ (Pontifícia Universidade Católica do Rio de Janeiro) e pela UFPB (Universidade Federal da Paraíba) por meio da junção de propostas FlexTV e MAESTRO [9].

O FlexTV foi a proposta inicial de middleware procedural, esta proposta incluía um conjunto de APIs compatíveis com padrões adotados em outros países além de novas funcionalidades, como a comunicação com múltiplos dispositivos, permitindo a interação de diversos usuários com uma aplicação interativa a partir de dispositivos remotos [9].

O MAESTRO foi a proposta de middleware declarativo focada em prover facilidade de sincronismo espaço temporal entre objetos multimídia, utiliza a linguagem declarativa NCL e agregando as funcionalidades da linguagem de script Lua [9].

A maioria das propostas de middlewares para TV Digital oferecem como suporte a execução de aplicações interativas em dois ambientes: declarativo e imperativo [4] [9]. No middleware Ginga não é diferente, ele conta com estrutura para execução de aplicações declarativas escritas em Nested Context Language (NCL) e imperativas escritas na linguagem de programação Java [9]. Estas duas diferentes soluções foram denominadas de Ginga-NCL (declarativa) e Ginga-J (imperativa) respectivamente [4]. O Ginga integrou as duas soluções (Ginga-J e Ginga-NCL), tomando por base as recomendações internacionais da ITU [12]. Desta forma, o Ginga é subdividido em dois subsistemas interligados, também chamados de Máquina de Execução (Ginga-J) e Máquina de Apresentação (Ginga-NCL) [9]. A arquitetura do Ginga é exibida na figura 1.

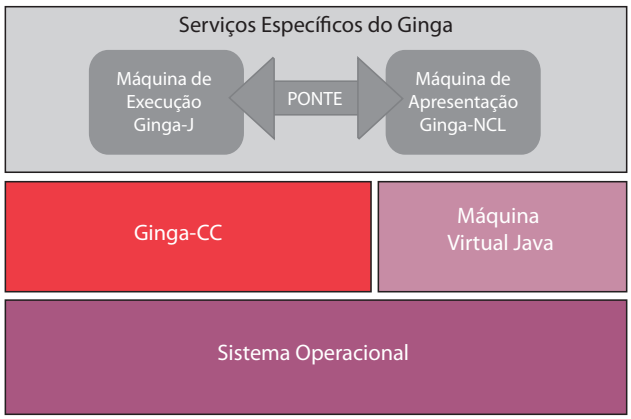

Figura 1. Arquitetura do middleware Ginga

Como apresentado na arquitetura demonstrada na figura 1, os dois ambientes (Ginga-J e Ginga-NCL) podem trabalhar de forma independente ou em cooperação, visto a existência de uma "ponte" que disponibiliza mecanismos para intercomunicação entre os mesmos, de modo que as aplicações imperativas utilizem serviços disponíveis nas aplicações declarativas, e vice-versa. Portanto, é possível a execução de aplicações híbridas, permitindo agregar as facilidades de apresentação e sincronização de elementos multimídias da linguagem NCL com o poder da linguagem orientada a objetos Java [9] [4] [15].

As características, requisitos e tipos de funcionalidade contidas em uma aplicação para TVDI influenciam diretamente o tipo de linguagem (paradigma) a ser utilizado na implementação. Assim como em outros domínios, (como na Web) as aplicações para TV digital são, normalmente, construídas usando abordagens declarativas, imperativas ou híbridas, integrando os dois tipos de linguagens [4].

O Ginga Common Core (Ginga-CC) é responsável por disponibilizar funcionalidades específicas de TV Digital comuns para os ambientes imperativo e declarativo, abstraindo as características específicas de plataforma e hardware. Entre suas principais funções, temos: exibição e controle de mídias, o controle de recursos do sistema, canal de retorno, dispositivos de armazenamento, acesso a informações de serviço, sintonização de canais, entre outros [9] [4].

III. Aplicações para TVDI

Segundo [8] [7] [4], as aplicações para TVDI, independente de sua linguagem de programação, são categorizados em três categorias, são elas:

1) Aplicações que consomem ou produzem dados e informações que não têm nenhum vinculo semântico com o conteúdo principal (áudio e vídeo) apresentado pela emissora. Temos como exemplo desta categoria de aplicações: serviços de acesso a e-mail via TV (TV-Mail) e acesso a dados bancários (TV-banking) ambos executados durante a exibição de uma partida de futebol.
2) Aplicações que consomem ou produzem dados e informações que têm relação com a semântica com o conteúdo principal (áudio e vídeo) apresentado pela emissora porém sem restrições fortes de sincronização. Temos como exemplo desta categoria de aplicações: serviços de acesso a informações sobre a tabela do campeonato brasileiro durante a exibição de uma partida de futebol.
3) Aplicações que consomem ou produzem dados e informações que têm relação com a semântica com o conteúdo principal (áudio e vídeo) apresentado pela emissora e são exibidos de forma sincronizada com este conteúdo. Temos como exemplo desta categoria de aplicações: exibição de anúncios interativos (propaganda interativa) pode ser feita durante a transmissão de um filme no exato instante em que a câmera enquadra o produto anunciado (ponto de sincronização temporal). Esta categoria pode ainda ser subdividida em:
   a) Programas de televisão que conteúdo principal (áudio e vídeo) é conhecido a priori (programas gravados como: filmes, desenhos animados, novelas, programas de auditório e etc);
   b) Programas de televisão que conteúdo principal (áudio e vídeo) é gerado ao vivo (partidas de futebol, corridas de Formula 1, lutas de MMA e etc);

As aplicações pertencentes a terceira categoria definida nos tópicos anteriores serão denominadas Aplicações Multimídia Complexas (AMC) neste artigo.

Uma das principais situação problema no âmbito das aplicações multimídia é o controle de sincronização temporal, (ou seja, a estrutura temporal) envolvendo diferentes tipos de mídia, interação com o usuário e as limitações da plataforma de apresentação [6]. A sincronização temporal é uma situação problema ainda mais grave se vista do angulo das AMCs.

O problema da sintonização tardia é um dos principais problemas para execução de AMCs, ele ocorre quando um telespectador sintoniza a um determinado canal de transmissão depois que parte do conteúdo extra para este programa já foi transmitido.

Na figura 2 o diagrama mostra um exemplo de sintonização tardia. Neste exemplo o Broadcaster envia Replays com os melhores momentos de um determinado programa de televisão. O telespectador A sintoniza o canal de transmissão antes da emissão de qualquer replay, sendo assim ele recebe todo o conteúdo extra enviado pela emissora, no entanto, o telespectador B só sintoniza ao canal de transmissão depois do envio do Replay 1, sendo assim este telespectador tem acesso apenas a parte do conteúdo enviado durante a transmissão do programa.

Um agravante para o problema da sintonização tardia é o fato que a comunicação no ambiente televisivo ocorre em sentido unidirecional partindo sempre do Broadcaster (emissora) para o receptor, tal característica impossibilita que o receptor faça requisições solicitando conteúdo extra não recebido a um Broadcaster [6].

Para resolver o problema da sintonização tardia foi necessário adotar um protocolo de comunicação que tenha como

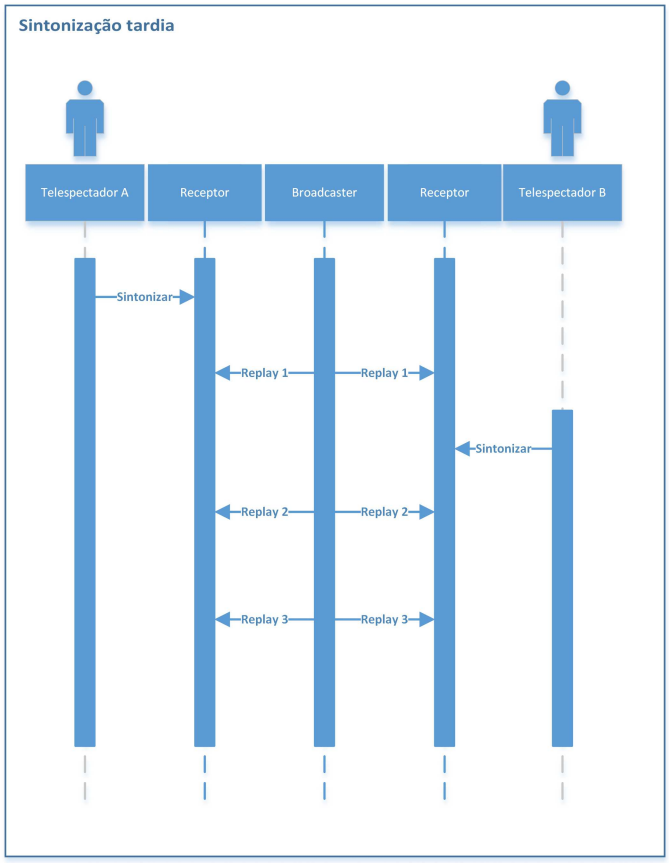

Figura 2. Sintonização Tardia

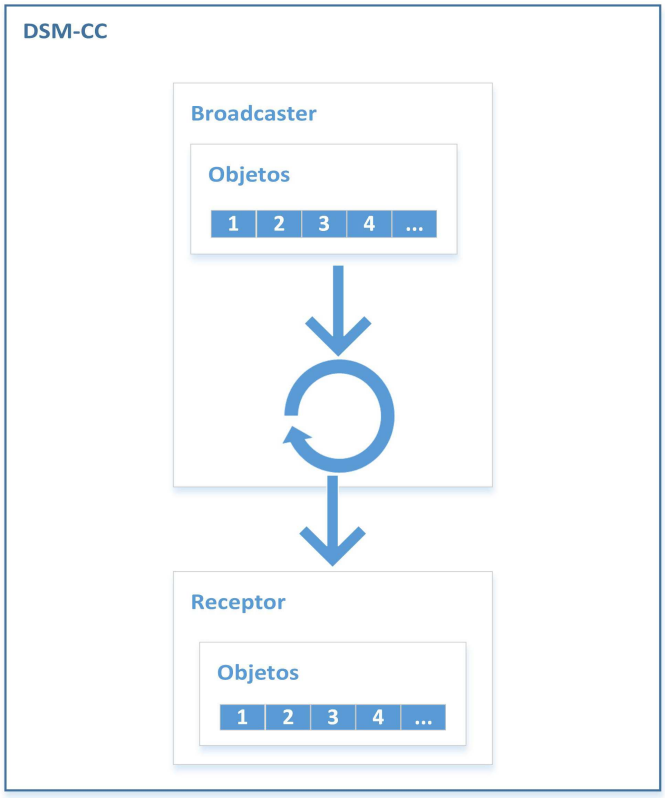

Figura 3. DSM-CC

características de transmissão o envio cíclico de dados, tornando o recebimento de dados independente do momento de sintonização ao transmissor já que caso a sintonização ocorra após o envio de um dado o receptor só precisa esperar para que o dado seja enviado novamente no próximo ciclo de envio [17].

O protocolo para transmissão cíclica de dados adotado pelo sistema brasileiro de televisão digital terrestre (SBTVD) foi o Digital storage media command and control (DSM-CC), devido a sua característica cíclica este protocolo é comumente denominado de Carrossel [17].

A figura 3 representa o funcionamento do DCM-CC, o Broadcaster armazena uma lista de objetos (dados) que deseja enviar aos receptores, para o envio deste dados o Broadcaster irá ler cada objeto da lista de forma cíclica e enviará os dados para o receptor. O receptor ao sintonizar um canal começa a receber os dados que estão sendo enviados armazenando o conteúdo novo e desprezando o que já foi recebido [6].

Voltando ao problema de sintonização tardia apresentado na figura 2 é visível que podemos resolvê-lo por meio do uso do DSM-CC uma vez que o Telespectador B (ou qualquer outro telespectador) receberá todo o conteúdo extra independente do momento de sintonização ao canal de transmissão devida a característica cíclica do protocolo.

IV. Televisão Digital e Computação Ubíqua

As tecnologias mais profundas são aquelas que desaparecem, aquelas que se misturam ao ambiente e ao cotidiano das pessoas e passam a ser imperceptíveis. Esta foi a definição para computação Ubíqua sugerida por Mark Weiser também conhecido como pai desta área da computação em 1991 [19].

Um ponto primordial que diferenciam a computação Ubíqua da computação tradicional é a necessidade de informação perceptual do ambiente visando atingir o objetivo da “invisibilidade”. Com tais informações perceptuais o sistema pode interagir com o usuário de forma mais natural mascarando a computação no ambiente real [19].

Antes da era da digitalização, os serviços de comunicação formavam uma cadeia discreta de componentes que restringiam tipos distintos de conteúdo para redes e dispositivos específicos. Este modelo de produção de conteúdo é denominado modelo vertical ou modelo tradicional. Na figura 4 pode-se observar que nesta estrutura a empresa difusoras de conteúdo estava verticalmente integrada ao longo de toda a cadeia produtiva. Este fato impedia que informações fossem facilmente transferidas de um serviço para outro [20] [4].

A chegada da era da digitalização de conteúdos possibilitou o armazenamento, a modularização e a distribuição de conteúdo por diferentes meios de comunicação. Este fato, aliado ao crescimento da inovação tecnológica e ao acesso facilitado a redes de comunicação (Internet), ocasionou o crescimento do fenômeno denominado convergência. No âmbito

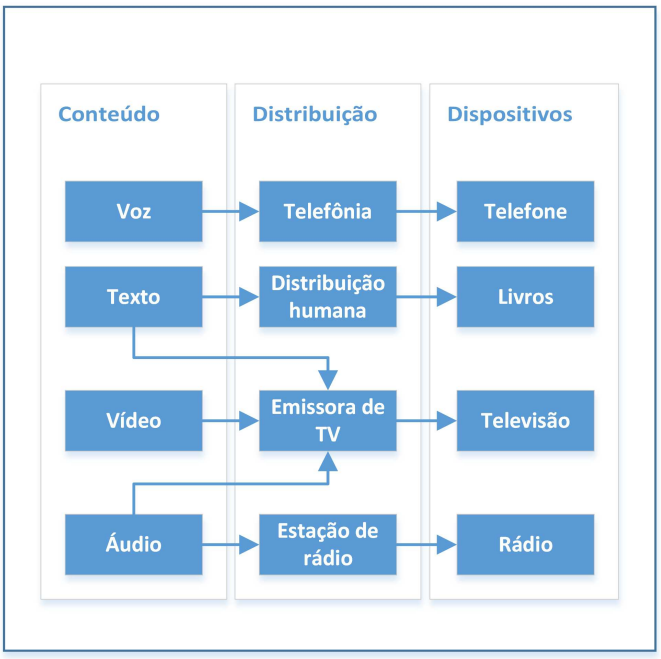

Figura 4. Organização vertical do modelo de produção de conteúdo antes da era da digitalização

da televisão digital, o fenômeno da convergência é definido como a união entre as telecomunicações (redes) e conteúdos digitais resultando em interconexão e interoperabilidade para diversos tipos de mídia [21] [4].

Com o avanço da convergência digital, surgiu a necessidade de aprimorar o modelo de produção de conteúdo tradicional (modelo vertical). Diante desta necessidade, foi definido um novo modelo de produção, o modelo horizontal de produção de conteúdo multimídia. O modelo horizontal é definido em seis fases, são elas: criação, produção empacotamento, serviços, distribuição e interface [21].

1) Criação de conteúdo, fase na qual o conteúdo digital é gerado e tem como objetos, os vídeos e músicas e, como agentes, os atores, jogadores e músicos.
2) Produção de conteúdo, fase na qual o conteúdo criado é produzido, transformando o mesmo em programas de televisão ou rádio. Os objetos são definidos como filmes, programas de esporte e os agentes são os diretores e designers, por exemplo.
3) Empacotamento, fase responsável pela montagem dos diferentes conteúdos, criando uma programação para o usuário final, tendo como objetos os canais de TV e, como agentes, os editores de TV.
4) Desenvolvimento e gerência de serviços, fase que tem como objetivo fornecer serviços que agregam valor ao conteúdo criado. Esta fase tem como objetos serviços personalizados e interativos. Como agentes temos os desenvolvedores de software.
5) Distribuição, fase na qual existe a transmissão do conteúdo gerado. Os objetos são os satélites, agentes e provedores de satélite.
6) Interface, fase que provê o contato com o usuário. Nesta, está inserida a produção de dispositivos para interação entre usuário e conteúdo. Esta fase tem como objetos os set top boxes e, como agentes, os fabricantes de hardware.

A divisão em fases permite a flexibilização das atividades e a geração de um produto específico ao final de cada etapa [21]. As relações entre os objetos e os agentes de cada fase podem ser observadas na figura 5.

Figura 5. Organização horizontal do modelo de produção de conteúdo pós era da digitalização

Diante do novo processo de produção de conteúdo multimédia (que agora engloba o desenvolvimento de softwares), voltado para televisão digital interativa, temos como desafio atender as características requiridas por este novo processo. Este novo processo é caracterizado como não linear, iterativo, ágil multidisciplinar, heterogêneo e convergente. Essas características são semelhantes com aquelas encontradas em sistemas ubíquos onde softwares diferentes podem rodar em plataformas diferentes e muitas vezes não convencionais (ex.: TV) [5].

A identificação de um sistema Ubíquo ocorre através da incorporação de algumas características chave desta categoria de sistema. Dentro das principais características dos sistemas

Ubíquos temos a invisibilidade, heterogeneidade, mobilidade física, mobilidade lógica, adaptabilidade (sensibilidade ao contexto) , tolerância a falhas, intemporabilidade, avaliação de contexto, segurança, privacidade das informações, mapeamento de dispositivos, busca de serviços e localização. É valido ressaltar que para um sistema ser caracterizado como Ubíquo o mesmo não precisa implementar todas as características citadas e sim um sub conjunto delas.

Nos tópicos a seguir as características de sistemas Ubíquos que se encaixam no ambiente de simulação desenvolvido serão descritas. As características presentes no ambiente de simulação são: heterogeneidade, mobilidade lógica, intemporabilidade, invisibilidade, tolerância a falhas e sensibilidade ao contexto.

A. Heterogeneidade

Um ambiente heterogênico é um ambiente onde dispositivos de diferentes plataformas, arquiteturas e sistemas operacionais estão inseridos. Nos dias atuais estes ambientes são cada vez mais comuns, visto que o acesso a redes é uma caraterística presente em diversos dispositivos que necessitam se comunicar.

A principal barreira a respeito da heterogeneidade esta no desenvolvimento de aplicações. Normalmente aplicativos são desenvolvidos para uma classe de dispositivos, plataforma e sistema operacional especifico. Esta prática leva ao desenvolvimento de versões diferentes da mesma aplicação para diversos dispositivos resultando em custo de desenvolvimento [19].

Com a finalidade de transpassar o problema da heterogeneidade, o ambiente de simulação desenvolvido foi escrito utilizando a linguagem de programação Java. O código Java é executado em cima de uma máquina virtual, este fator faz com que o código fonte da aplicação seja independente da plataforma e sistema operacional de execução do sistema, como resultado o ambiente de simulação desenvolvido pode ser executado em dispositivos que tenham a máquina virtual Java instalada.

B. Mobilidade lógica

A mobilidade lógica, ou migração de código, é a ação de migrar um código fonte ou aplicação compilada de forma completa ou parcial para execução em um outro dispositivo. Nos sistemas Ubíquos esta mobilidade ocorre por diversos motivos que vão desde a adaptabilidade a restrições computacionais (onde a migração ocorrer para um dispositivos com maior poder computacional) até a gerência de energia (onde a migração ocorrer para economizar a energia de um determinado dispositivo).

Em sua maioria a migração lógica ocorre utilizando código fonte interpretado e não compilado, esta prática é vastamente utilizada em sistemas tradicionais como por exemplo em web site através do uso de linguagem de script como Javascript e VBScript. Um dos problemas da migração lógica utilizando códigos interpretados é que não é possível manter o sigilo do mesmo.

A grande diferença entre a mobilidade lógica em sistemas tradicionais e sistemas Ubíquos é o motivo pelo qual a migração ocorre. O motivo da migração lógica geralmente ocorrer com objetivo de prover uma melhor experiência do usuário por meio do uso de interfaces mais amigáveis, economiza de energia e aumento da velocidade de funcionamento do sistema.

O simulador implementado permite o recebimento de e execução de aplicações (Xlets) enviadas através de um Broadcaster. Dentre a estrutura de recebimento de dados implementada no ambiente de simulação, existe um módulo específico para recebimento de aplicações e sua respectiva execução. Esta mobilidade lógica ocorre pois as aplicações interativas para televisão digital são executadas no receptor devido a característica unidirecional da comunicação que é sempre partindo do Broadcaster para o Receptor tornando inviável que as aplicações sejam executadas no Broadcaster sendo impossível que o Receptor realize requisições devido a comunicação unidirecional.

C. Intemporabilidade

A comunicação entre dispositivos diferentes é uma das características fundamentais dos sistemas Ubíquos. Esta necessidade de comunicação é resultado da facilidade do acesso a redes e a natureza heterogenia dos dispositivos. Assumindo que as implementações internas são fatores particulares de cada dispositivo, a computação Ubíqua deve encontrar maneiras de mascarar a heterogeneidade garantindo a interoperabilidade entre diversos dispositivos [19].

Em relação a interoperabilidade nas comunicações, os desenvolvedores têm enfrentado constantes problemas de incompatibilidade entre protocolos. A solução mais comumente adotada para superar estes problemas é a utilização de protocolos bem definidos ou middlewares de comunicação [19].

A maneira mais comum para garantir a interoperabilidade na comunicação entre dispositivos é a utilização de middlewares. Os middlewares tem um papel fundamental na implementação de sistemas Ubíquos, eles tem como responsabilidade estabelecer comunicação entre diferentes dispositivos trabalhando como um tradutor [19].

Com a finalidade de garantir a interoperabilidade toda a comunicação feita para o recebimento de dados de um Broadcaster no ambiente de simulação do simulador é feita através de um protocolo baseado no DCM-CC. A utilização deste protocolo garante a comunicação entre entre Broadcaster e Receptor independente da arquitetura em que cada um é executado.

D. Invisibilidade

Um sistema Ubíquo exige que uma intervenção mínima por parte do usuário para oferecer uma aproximação razoável da invisibilidade. Os usuários podem intervir nos sistemas com o objetivo de aprimorar a sua experiência com a utilização do sistema. É desejável que o sistema aprenda com as intervenções dos usuários, evitando que ele precise intervir no sistema para configurá-lo para uma mesma situação [19].

A implementação de um sistema completamente invisível ao usuário é um grande desafio para a computação Ubíqua [19]. No entanto já é comum a implementação de funcionalidades que independem de ação dos usuário dentro de um sistema. Temos como exemplo de funcionalidades invisíveis as atualizações automáticas de sistemas operacionais, adaptação automática de brilho de tela a depender da luminosidade do ambiente entre outras.

No ambiente de simulação desenvolvido uma funcionalidade específica tem como característica a invisibilidade. Ao sintonizar a um Broadcaster o receptor começa a escutar em uma porta pré-determinada para o recebimento de conteúdo extra. Todo conteúdo extra enviado pelo Broadcaster é recebido e armazenado no receptor, caso este conteúdo seja uma aplicação JDTV (xlet) e no metadado desta aplicação ela esteja setada como de execução automatizada a mesma é executada sem qualquer intervenção do usuário.

E. Tolerância a falhas

Não é possível evitar que um sistema falhe, mas suas consequências, ou seja o colapso do sistema, a interrupção no fornecimento de um serviço, podem ser evitadas pelo uso adequado de técnicas de tolerância a falhas. Entretanto, tolerar falhas é uma operação de alto custo pois as técnicas mais comuns de tolerância a falhas consistem em redundância de hardware ou manutenção de backups [22].

O conceito de tolerância a falha do ponto de vista da computação Ubíqua esta altamente atrelado ao conceito de invisibilidade. Se é desejado que o sistema seja imperceptível ao usuário e que dispense qualquer intervenção do mesmo tolerar falhas passa a ser fundamental.

O protocolo desenvolvido e adotado na implementação do ambiente de simulação tolera falhas relativas a perda de pacotes. Utilizando um protocolo baseado no DSM-CC, cada pacote recebido pelo receptor é checado e mapeado como recebido, caso um pacote seja perdido o receptor sabe que o mesmo não foi recebido e aguarda o próximo cíclico de envio do Broadcaster para receber este pacote.

F. Sensibilidade ao contexto

Em sistemas tradicionais não é comum que o mesmo avalie o estado do seu ambiente, como resultado disso tais sistemas não tem a capacidade de realizar decisões baseados no estado do contexto no qual ele pertence. Porém uma das necessidades dos sistemas Ubíquos é a percepção do contexto para tomada de decisão baseada em seu estado [19].

O desafio da sensibilidade ao contexto esta relacionada à obtenção e interpretação da leitura de dados extraídos do ambiente. Entre tais dificuldades pode-se ressaltar o uso de geolocalização não preciso, dificuldade em implementar sistemas com tomada de decisão adequada, obtenção de dados de múltiplos sensores que podem estar em desacordo [19].

A sensibilidade ao contexto não foi aplicada ao ambiente de simulação em si como uma caraterista própria, porém este ambiente provê um arcabouço quera que as aplicações que executem sobre ele possam ser sensíveis ao contexto. Esta possibilidade ocorrer pois as aplicações multimídias (xlet) através do utilização do ambiente de simulação podem consumir dados de diversas fontes de dados (Broadcaster e Providers na Internet). No caso do Xlet Clima, a aplicação simulada pode obter dados referentes a previsão de tempo tanto do Broadcaster como da Internet, a tomada de decisão de onde obter tal dado se dá através da disponibilidade do serviço fornecedor.

V. Trabalhos correlatos: ambientes de simulação virtualização

Um emulador é um sistema que simula toda uma arquitetura computacional, de forma que todo o conjunto de instruções do sistema operacional (SO) emulado são traduzidas e executadas no SO original. A diferença entre emuladores e máquinas virtuais, está no nível de abstração. Enquanto em um emulador toda as instruções são traduzidas e interpretadas, em máquinas virtuais as instruções são executadas no modo mais nativo possível. Comumente, emuladores e máquinas virtuais são usados para testar o funcionamento de programas escritos para arquiteturas diferentes daquelas em que o programa é escrito [13].

Os principais motivos para utilização de emuladores e máquinas virtuais para o teste de softwares são: redução de custo de compra dos ambientes reais, o aumento da velocidade de desenvolvimento já que o ambiente emulado é executado na mesma máquina onde o software é desenvolvido evitando que a cada interação de teste a aplicação seja transportada e instalada no ambiente real e a segurança pois a aplicação em teste só pode causar danos a um ambiente virtual [13].

Atualmente encontram-se disponíveis alguns emuladores e máquinas virtuais para testes de aplicações desenvolvidas para televisão digital interativa (TVDI), nós tópicos a seguir alguns destes ambientes serão apresentados.

A. Set-top Box Virtual Ginga-NCL

O Set-top Box Virtual Ginga-NCL é uma máquina virtual que contém a implementação de referência Ginga-NCL C++, executada sobre uma plataforma Linux. Esta versão foi desenvolvida coma a finalidade de prover uma implementação de maior desempenho, de forma que possa ser facilmente portada para plataformas com características semelhantes aos set-top boxes comerciais [14].

Todo tipo de ação e comandos de controle que são enviados para o ambiente de virtualização em questão são executados por meio de uma aplicação de console remoto SSH (Secure Shell), ou seja a instalação de aplicações são feitas utilizando esta interface. Somente as ações de interatividade com as aplicações já instaladas no ambiente de virtualização serão feitas diretamente na máquina virtual. Isso reproduz fielmente um ambiente real de desenvolvimento de aplicações Ginga-NCL, no qual o set-top box deve ser controlado remotamente [14].

Após virtualizada esta máquina virtual oferece um ambiente para teste de aplicações NCL/NCLua com portabilidade para diversos sistemas operacionais e com o desempenho fiel ao

ambiente real de execução, a figura 6 apresenta a execução de uma aplicação utilizando o Set-top Box Virtual Ginga-NCL [14].

Figura 6. Set-top Box Virtual Ginga-NCL

B. *XleTView*

O XleTView é um simulador para TV Digital que contempla a execução de aplicações imperativas desenvolvidas em Java, tais aplicações são denominadas Xlets. A ferramenta implementa APIs do padrão DVB-MHP (Multimedia Home Platform), sendo também independente de plataforma por ser desenvolvido em Java. Além disso, o XleTView implementa a API JavaTV, dessa forma podendo ser considerada uma das ferramentas mais robustas e popular para emular TV Digital em plataforma PC [14].

A API JavaTV é importante para o software em questão por se tratar de um conjunto de classes e interfaces que oferecem funcionalidades e serviços interativos de TV, tais como execução de arquivos de áudio e vídeo e controle de sintonização de canais [14].

Devido a sua simplicidade tanto de instalação quanto de utilização e o fato de ser um software livre de código aberto sob a licença do GNU GPL, o XeleTView se tornou umas das ferramentas mais utilizadas na emulação de TV Digital [14]. Na Figura 7 é apresentada a interface do XleTView.

C. *OpenMHP*

Assim como o XleTView o OpenMHP também é um emulador para TV Digital com a finalidade de executar Xlets em um computador. Também é baseada na especificação DVB-MHP e requer a implementação da API JavaTV e da biblioteca JMF (Java Media Framework) [16].

Como pode ser visto da figura 8, o OpenMHP possui um gerenciador de aplicações onde todas as configurações dos Xlets testados devem ser realizadas através do mesmo. A partir do gerenciador é possível atribuir algumas variáveis

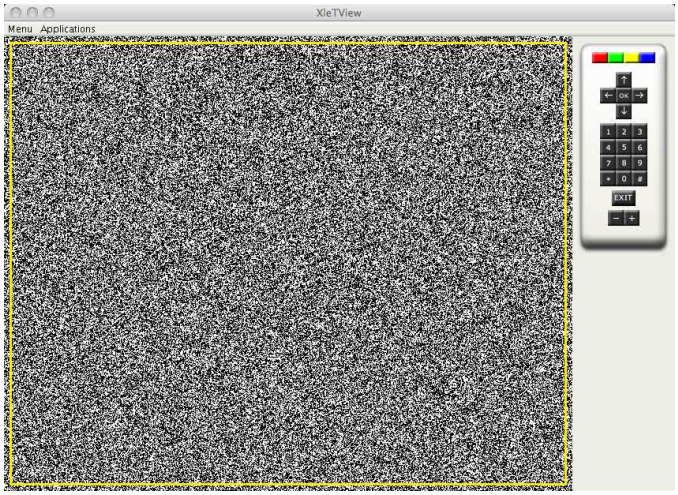

Figura 7. XleTView

as aplicações como o classpath, identificador da organização, identificador da aplicação, entre outras, além da possibilidade de alteração das mensagens de erro e debug que são exibidas durante os testes com o ambiente. As aplicações só estão habilitadas a aparecerem na lista de exibição após as devidas configurações.

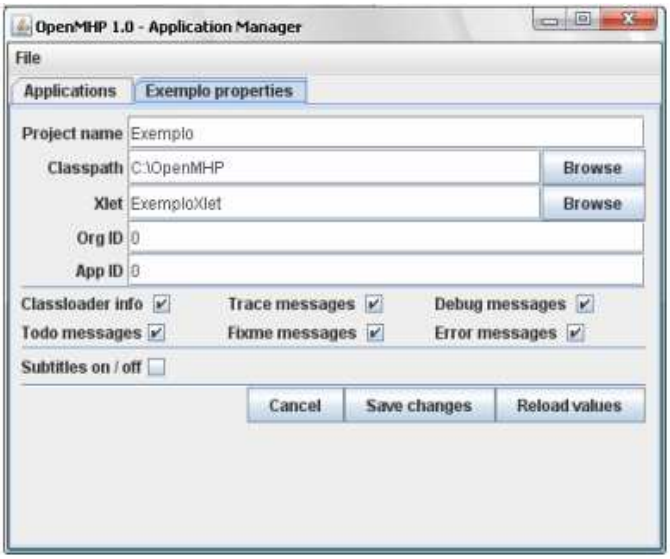

Figura 8. OpenMHP

Após configurada, a Xlet pode ser executada. Durante a execução da aplicação são exibidas três telas: i) Saída textual com informações de depuração; ii) TVScreen (janela de exibição da Xlet); e iii) Janela de controle remoto.

A saída textual com informações de depuração é responsável por exibir todos os eventos e mensagens relativas à aplicação testada no momento da execução. Os eventos que são exibidos textualmente vão desde eventos de cliques na tela de exibição até aqueles sobre as operações realizadas com o controle remoto, passando também pela exibição de erros e exceções

dos Xlets testados [16].

O OpenMHP é uma ferramenta de virtualização para desktop mais complexa em relação XleTView, a sua funcionalidade de depuração oferece uma visão melhor de teste das aplicações desenvolvidas, entretanto a sua instalação e configuração são mais complexas [16].

D. Emulador Ginga-J

Este emulador oferece um ambiente de execução de aplicações mais simples de instalar e usar em um computador pessoal. Porém, devido essa característica a ferramenta apresenta limitações para representar as funcionalidades disponíveis num terminal de acesso real. A principal restrição está relacionada ao uso dos protocolos das redes, como sinalização de aplicações, seleção de fluxos elementares, acesso a informações de serviço e carrossel de dados, já que a execução das aplicações é realizada localmente [10].

O emulador Ginga-J é um projeto de código aberto desenvolvido pelo LAVID (Laboratório de Aplicações de Vídeo Digital) da UFPB (Universidade Federal da Paraíba). Este emulador foi desenvolvido utilizando a linguagem Java e teve sua arquitetura baseada no XleTView [10]. Na figura 9 pode ser visto o funcionamento do emulador Ginga-J executando um Xlet.

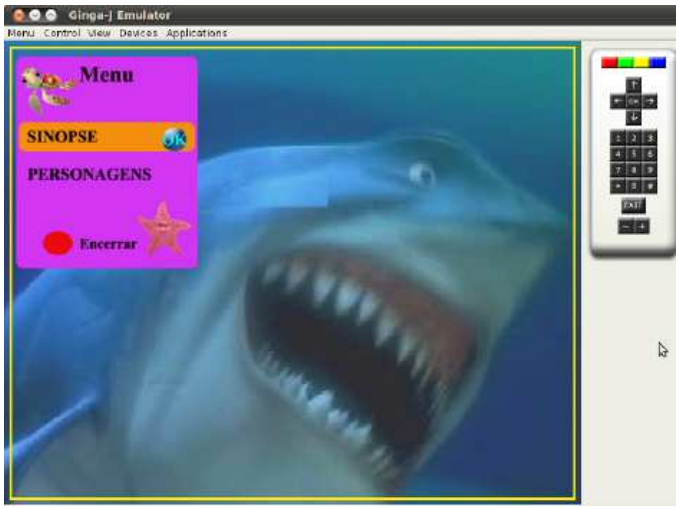

Figura 9. Emulador Ginga-J

E. Análise dos trabalhos correlatos

Os ambientes apresentados permitem executar aplicações interativas de TV digital a partir do uso de computadores simulando o receptor de TV. Por outro lado, eles não contemplam a infraestrutura necessária para recepção de conteúdo (principal e extra). Esta funcionalidade de recepção de conteúdos é ponto fundamental para permitir a execução 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), são transmitidos via fluxos alternativos e são exibidos de forma sincronizada [6].

Na tabela I é feita uma comparação entre os ambientes apresentados. Esta comparação engloba três aspectos que são: i) Plataforma para execução de aplicações NCL; ii) Plataforma para execução de aplicações Java (Xlets); e iii) Infraestrutura para o recebimento de conteúdos extra e principal por meio do uso de uma rede TCP IP.

Ambiente	**NCL**	**Java**	**Recebimento de conteúdos**
Set-top Box Virtual Ginga-NCL	Sim	Não	Não
XleTView	Não	Sim	Não
OpenMHP	Não	Sim	Não
Emulador Ginga-J	Não	Sim	Não
Ambiente proposto	Não	Sim	Sim

Tabela I
COMPARAÇÃO ENTRE AMBIENTES

VI. FUNCIONAMENTO DO SIMULADOR

Os requisitos do simulador incluem a sintonização de um canal de transmissão e como consequência o recebimento e apresentação tando de conteúdo principal como extra. O conteúdo principal trata-se de um fluxo de vídeo enquanto o conteúdo extra pode ser qualquer tipo de arquivo ou diretório, incluindo aplicações JDTV que devem ser executadas pelo simulador.

Durante a fase de concepção do projeto os requisitos funcionais previamente levantados foram mapeados em quatro casos de uso (diagrama de casos de uso exibido na figura 10), são eles:

1) Sintonizar canal.
2) Receber conteúdo principal.
3) Receber conteúdo extra.
4) Executar aplicações JDTV.

A. Sintonizar canal

O usuário pode sintonizar um canal de transmissão por meio do simulador para recebimento de conteúdos principal e extra. Esta ação é a representação da sintonização de um canal em uma televisão convencional. Sintonizar um canal para o simulador significa escutar um par de portas para o recebimento de dados, uma porta para conteúdo principal e uma porta para o conteúdo extra.

B. Receber conteúdo principal

Este caso de uso é acionado após a sintonização de um canal. Para o simulador receber o conteúdo principal significa escutar uma porta (definida na sintonização do canal) para o recebimento de um fluxo de vídeo e apresentar o mesmo para o usuário final.

O usuário não deve ter nenhum controle sobre o fluxo de vídeo do conteúdo principal. Assim como em uma transmissão de televisão convencional o usuário pode apenas sintonizar um canal e assistir ao conteúdo, não é possível adiantar, pausar ou retroceder o fluxo de vídeo.

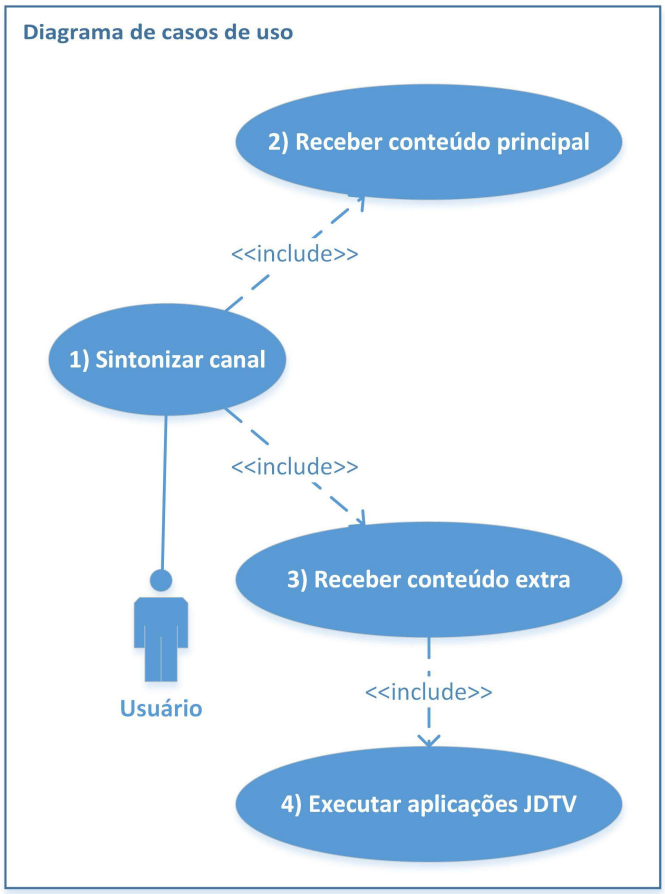

Figura 10. Diagrama de casos de uso

C. Receber conteúdo extra

Este caso de uso é acionado após a sintonização de um canal. Para o simulador receber o conteúdo extra significa escutar uma porta (definida na sintonização do canal) para o recebimento de dados.

O conteúdo extra é classificado em três tipos distintos: 1) Arquivos; 2) Diretórios; 3) Aplicações JDTV. Cada tipo de conteúdo extra recebido tem um tratamento especifico, são eles:

1) *Arquivo*: Para o recebimento de arquivos o simulador deve receber os dados e armazenar em disco.
2) *Diretório*: Para o recebimento de diretórios o receptor deve receber toda a árvore de arquivos (diretório, sub diretórios e arquivos) e armazená-los em disco respeitando a hierarquia do diretório enviado.
3) *Aplicação*: Para o recebimento de aplicações JDTV o simulador deve receber e armazenar toda a estrutura de arquivos da aplicação. Além disso metadados sobre a aplicação devem ser recebidos, são eles: a) Nome da aplicação; b) Identificação da classe de arranque da aplicação. Estes metadados são mantidos na memória para futura execução.

D. Executar aplicações JDTV

Caso o conteúdo extra recebido seja uma aplicação JDTV ela deve ser executa pelo simulador. Este caso de uso é acionado após o recebimento de conteúdo extra para este tipo de conteúdo. Para executar a aplicação os metadados obtidos na fase de recebimento de dados são utilizados.

VII. Arquitetura

A arquitetura do simulador esta dividida em três módulos denominados Receptor de conteúdo extra, Receptor de conteúdo principal e Apresentação conforme pode ser visto na Figura 11.

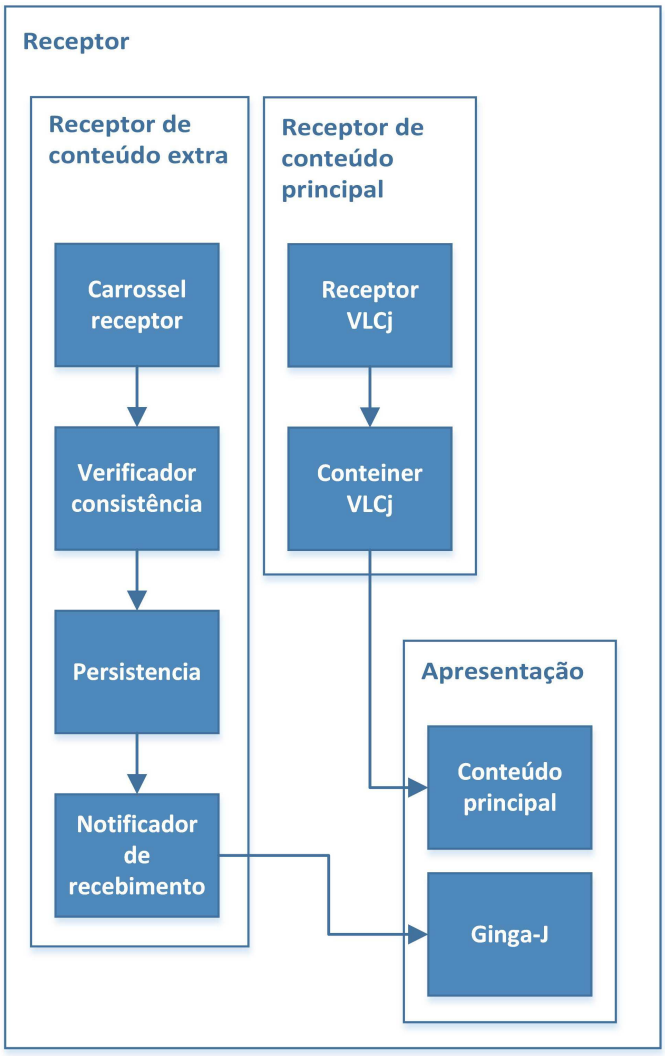

Figura 11. Arquitetura

A. Receptor de conteúdo extra

O módulo Receptor de conteúdo extra tem como finalidade receber o conteúdo extra, verificar sua integridade, persistir o conteúdo recebido e notificar a disponibilidade deste conteúdo. Para isso, tal módulo foi dividido em quatro sub-módulos cada um com sua respectiva responsabilidade.

Devida a grande complexidade para implementação do protocolo padrão de transmissão de conteúdo (DSM-CC) adotado pelo Sistema Brasileiro de Televisão Digital Terrestre, para este ambiente de simulação foi desenvolvido um novo protocolo baseado no DSM-CC, porém com um conjunto de recursos menos extensos.

O protocolo desenvolvido implementa um carrossel de objetos do tipo User - Network e têm grande similaridade conceitual como o protocolo DSM-CC com a finalidade de manter a garantia de entrega de forma confiável através do envio cíclico de dados.

1) Carrossel receptor: O sub-módulo denominado Carrossel receptor tem como objetivo recepcionar os dados provenientes de um Broadcaster. O protocolo de comunicação adotado por este módulo foi baseado no DSM-CC. A implementação do Carrossel receptor utilizou bibliotecas Java para transmissão de dados via rede (ex.: Socket, DatagramSocket, etc.). Cada pacote recebido por meio deste sub-módulo serve como entrada para o próximo sub-módulo.

2) Verificador de consistência: A fim de manter a similaridade com o DSM-CC e permitir uma entrega confiável sem uso do TCP, foi definido e implementado um protocolo de verificação de integridade no sub-módulo Verificador de consistência.

O sub-módulo Verificador de consistência tem como finalidade verificar a integridade dos pacotes recebidos do sub-módulo Carrossel receptor evitando o recebimento de pacotes duplicados e o consumo de objetos incompletos.

Para verificar a consistência dos pacotes recebidos este sub-módulo conta com uma lista de que representa cada objeto que já foi iniciado o recebimento, nesta lista cada objeto é representado pelo seu identificador. Cada objeto desta lista contem um sub lista contendo a representação de cada um dos seus pacotes. Para cada representação de pacote é mantido o identificador do pacote e um boleando que identifica se o pacote foi recebido (true) ou não (false).

Para cada pacote recebido do sub-módulo Carrossel receptor o sub-módulo Verificador de consistência efetua primeiramente a leitura do cabeçalho do pacote recebido. Para esta fase somente os seguintes campos são avaliados: ID (identificador do objeto que contém o pacote), Number of packets (número de pacotes do objeto), Packet sequence (número sequencial que identifica o pacote).

Se a lista de objetos não contem nenhum objeto cujo o identificador seja igual ao capo ID do cabeçalho do pacote recebido os seguintes passos são seguidos: 1) É adicionado uma representação do novo objeto na lista de objetos. 2) É adicionado ao objeto recém criado a lista de pacotes que vai de 1 até o valor do campo Number of packets contido no cabeçalho do pacote recebido. Todas as posições da lista são marcadas como pacote não recebido (false). 3) Na lista de pacotes recém criada o pacote cujo o identificador seja igual ao campo Sequence do pacote recebido é marcado como recebido (true). 4) O pacote é enviado para saída.

Se a lista de objetos contem o objeto cujo o identificador seja igual ao capo ID do cabeçalho do pacote recebido e a representação do pacote (na lista de pacotes do objeto) esta marcado como não recebido (false), os seguintes passos são seguidos: 1) Na lista de pacotes o pacote cujo identificador seja igual ao campo Packet sequence do pacote recebido é marcado como recebido (true). 4) O pacote é enviado para saída.

Se a lista de objetos contem o objeto cujo o identificador seja igual ao capo ID do cabeçalho do pacote recebido e a representação do pacote (na lista de pacotes do objeto) esta marcado como recebido (true), este pacote é descartado.

Todo processo executado pelo sub-módulo Verificador de consistência descrito neste tópico pode ser visto na Figura 12.

3) Persistência: O sub-módulo denominado Persistência tem como objetivo persistir os dados recebidos do sub-módulo Verificador de consistência.

Para cada pacote recebido do sub-módulo Verificador de consistência o sub-módulo Persistência efetua primeiramente a leitura do cabeçalho do pacote recebido. Para esta fase somente os seguintes campos são avaliados: ID (identificador do objeto que contém o pacote), Number of packets (quantidade total de pacotes do objeto), Packet sequence (número sequencial que identifica o pacote)e Data Length (quantidade de bytes da parte de dados do pacote).

O processo de persistência inicia-se verificando se já existe um diretório criado para os pacotes daquele objeto por meio do campo ID do pacote recebido. Caso não exista um diretório é criado com o nome do identificador do objeto. Em seguida, a parte de dados do pacote é persistida (dentro do diretório de referência do objeto) em um arquivo cujo nome é o conteúdo do campo Packet sequence do pacote recebido e que tem a extensão pkt. Logo após, sub-módulo Persistência verifica se o objeto está completo, ou seja se a quantidade de arquivos .pkt no diretório de referência do objeto é igual ao valor do campo Number of packets. Caso o objeto esteja completo, os arquivos pkt são lidos de forma sequencial e seu conteúdo é persistido em um único arquivo, ao fim da leitura os arquivos pkt são excluídos e uma instância do objeto é enviada para saída do sub-módulo.

Para implementação deste sub-módulo foram utilizadas bibliotecas Java para manipulação de arquivos como por exemplo: File, FileWriter, BufferedWriter, BufferedReader, FileReader e etc.). Todo processo executado pelo sub-módulo Persistência descrito neste tópico pode ser visto na Figura 13.

4) Notificador de recebimento: Este sub-módulo tem como finalidade informar que um determinado objeto (conteúdo extra) foi recebido. Esta notificação foi implementada de acordo com o padrão de projeto Observer.

B. Receptor de conteúdo principal

O módulo Receptor de conteúdo principal tem como finalidade receber o conteúdo principal, encapsular o mesmo e disponibiliza-lo para o módulo de Apresentação. Para a implementação deste módulo 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

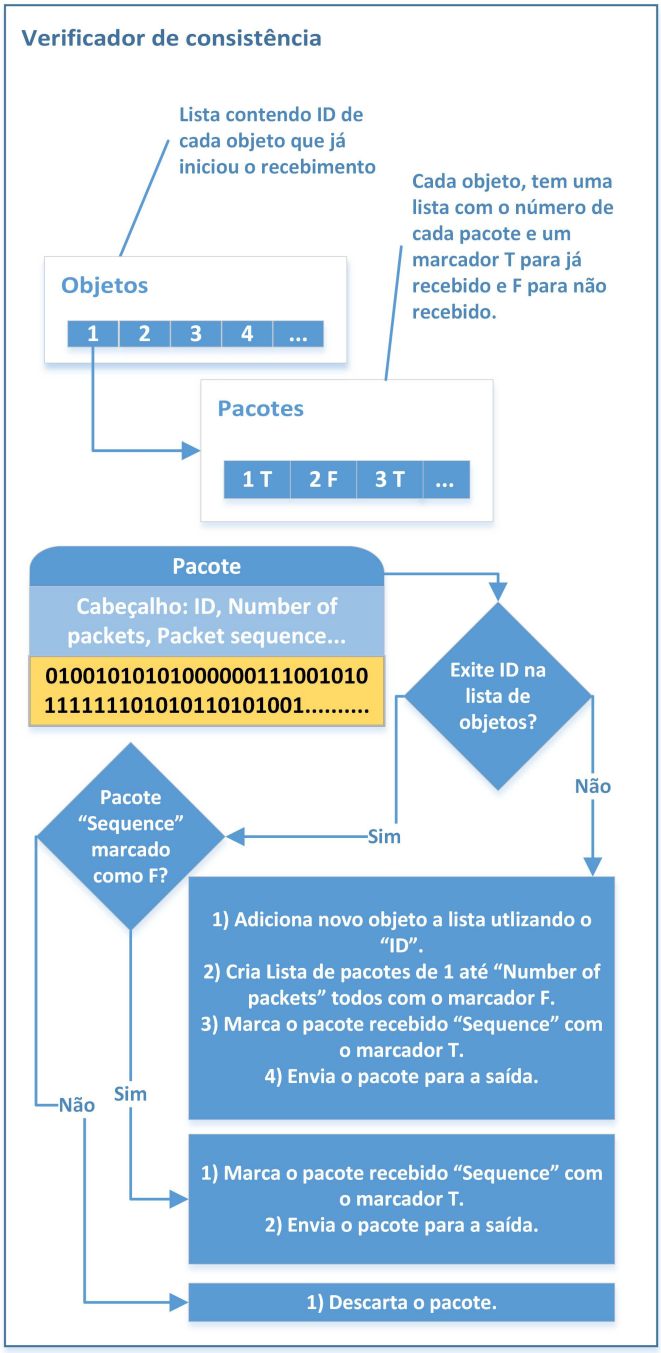

Figura 12. Verificador de consistência

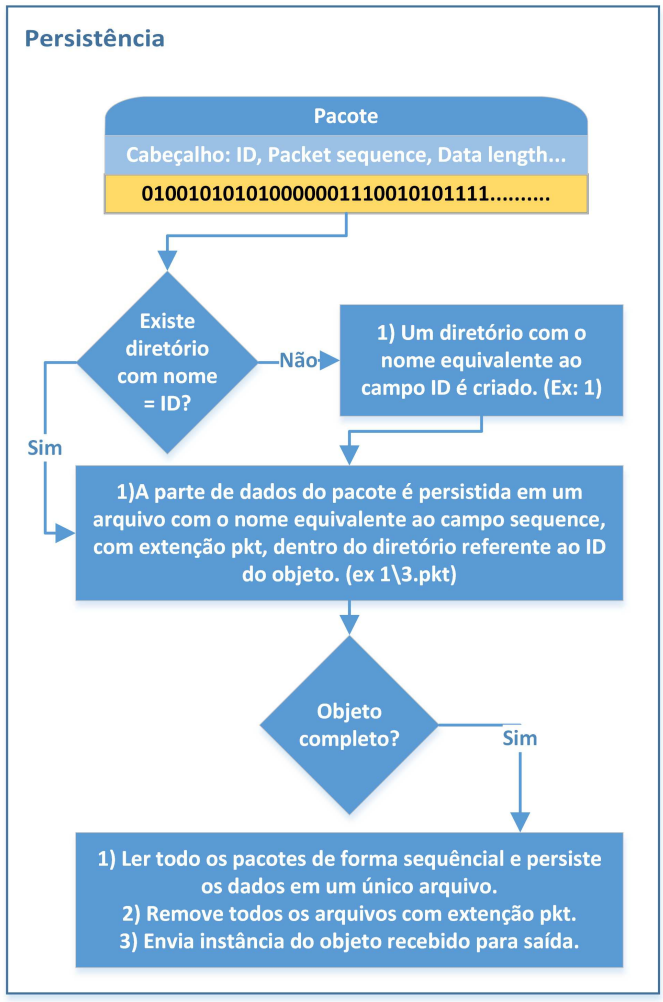

Figura 13. Persistência

código escrito em Java [18]. Para atender aos objetivos deste módulo o mesmo foi implementado em dois sub-módulos.

1) Receptor VLCj: O sub-módulo Receptor VLCj tem como objetivo escutar uma transmissão de conteúdo principal, para isso é necessário sintonizar um fluxo de vídeo definindo o protocolo de comunicação e a porta de escuta. No simulador em questão o protocolo adotado para recebimento de fluxo de vídeo principal é o UDP, a porta de escuta neste caso representa um canal da televisão convencional.

2) Contêiner VLCj: O sub-módulo Contêiner VLCj tem como objetivo integrar o fluxo sintonizado no sub-módulo Receptor VLCj a um contêiner de apresentação e disponibilizar o mesmo para uso do módulo de Aplicação.

Este contêiner funciona como um wrapper que encapsula as especificidades da biblioteca VLCj tornando a sua utilização mais transparente ao desenvolvedor e possibilitando uma futura troca de tectonologia de formar menos agressiva ao ambiente de simulação como um todo.

C. Apresentação

O módulo de Apresentação é responsável por realizar a interface entre o simulador e o usuário. É neste módulo em que o conteúdo principal e as aplicações Ginga-J são exibidas para o usuário. Para isso, este módulo apresenta dois sub-módulos.

1) Conteúdo principal: Este sub-módulo tem como finalidade apresentar o conteúdo principal ao usuário, sendo assim o módulo em questão recebe a instância do contêiner VLCj e a mesma é embutida em um contêiner Java e depois é apresentada ao usuário para simular a tela de uma TV real.

2) Ginga-J: Este sub-módulo tem como finalidade apresentar um grupo especifico conteúdo extra que são as aplicações Ginga-J. As aplicações em questão são exibidas sobre o contêiner Java (Overlay) que é utilizado para exibir o conteúdo principal. Os demais tipos de conteúdo extra (vídeos, áudio, imagens, textos e etc.) serão consumidos por estas aplicações. A implementação deste módulo teve como base a utilização e adaptação do código fonte do simulador XletView.

VIII. Estudo de caso: Xlet Clima

O Xlet Clima foi uma aplicação JavaDTV desenvolvida com o objetivo de homologar o funcionamento do simulador construído. A aplicação em questão consome duas fontes de dados diferentes (Broadcaster e Internet) e exibe ao usuário informações a respeito da temperatura de diferentes localidades.

Para o estudo de caso em questão foram utilizados dados fictícios para apenas duas localidades (Salvador e Rio de Janeiro), tantos os dados provenientes da Internet como do Broadcaster contam com os mesmos valores apenas para fins ilustrativos. A aplicação exibe os dados de duas fontes diferentes mas o conceito prova que na ausência de uma das fontes emissoras a aplicação é capaz de consumir dado de outra fonte garantindo assim uma maior disponibilidade de serviço.

A. Arquitetura

A arquitetura do Xlet Clima foi dividida em cinco módulos, dois módulos de consumo de dados (Consumidor web e Consumidor broadcaster), um módulo para acessar e efetuar a leitura do arquivo XML que será recebido através de ambos consumidores (Leitor XML), uma módulo para gerenciar, solicitar o consumo de dados e prover dados para o módulo de Apresentação, onde as informações serão apresentadas ao usuário. O funcionamento de cada um destes módulos será explicitado nos tópicos abaixo, o diagrama desta arquitetura encontra-se na Figura 14.

1) Thread de consumo de dados: Módulo responsável por periodicamente solicitar atualizações de dados para ambos consumidores. Ao fim de cada solicitação de dados este módulo envia informações para atualizações do módulo de apresentação. A implementação deste módulo utiliza uma thread independente da thread principal da aplicação para evitar o bloqueio da mesma.

2) Consumidor Web: Ao ser solicitado este módulo executa uma requisição http Get para receber o XML da Web. Em seguida, os dados obtidos como resposta da requisição (o corpo do XML) é armazenado na memória e enviado para o módulo de Leitor XML. Para desenvolvimento deste módulo foram utilizadas bibliotecas Java para requisições Http e leitura de fluxo de dados (ex.: HttpURLConnection, InputStream, BufferedReader).

3) Consumidor Broadcaster: Ao ser solicitado este módulo verifica a existência de um arquivo (clima.xml) na pasta de arquivos recebidos do receptor (Este arquivo chega até esta pasta através de todo processo explicado na arquitetura do simulador). Se o arquivo for encontrado ele será enviado para o módulo de Leitor XML.

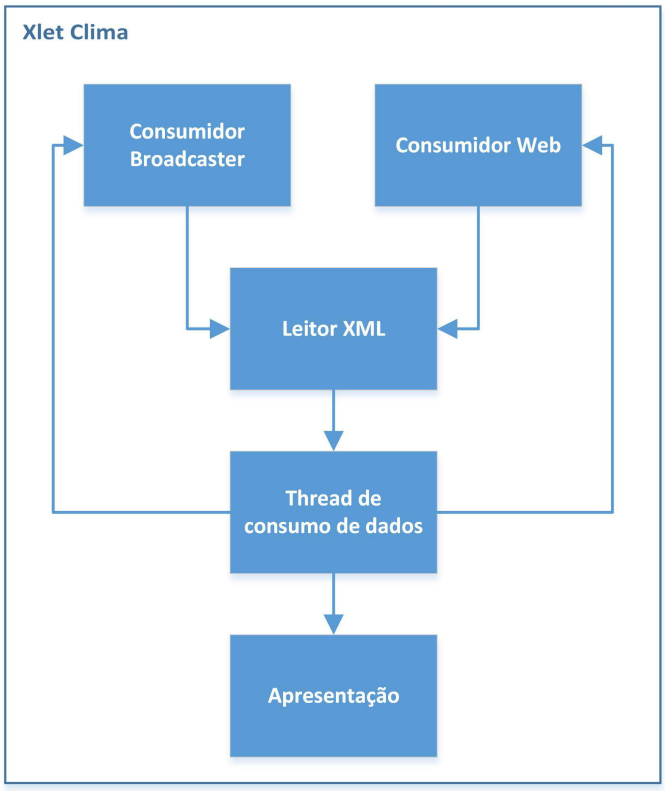

Figura 14. Arquitetura do Xlet Clima

4) Leitor XML: Este módulo lê da memória (dado proveniente do Consumidor web) ou do disco (dado proveniente do Consumidor broadcaster) um arquivo XML e retorna as informações (o nome de cada local e sua respectiva temperatura) de tal arquivo para o módulo Thread de consumo de dados. O código abaixo exemplifica a estrutura do XML consumido.

```
<root>
 <climas>
  <clima>
   <local>Salvador</local>
   <temperatura>29</temperatura>
  </clima>
  <clima>
   <local>Rio de Janeiro</local>
   <temperatura>40</temperatura>
  </clima>
 </climas>
</root>
```

5) Apresentação: Este módulo recebe informações as informações do módulo Thread de consumo de dados e exibe as mesmas para o usuário utilizando a classes da biblioteca Lwuit (Ex: Form, Label e etc).

Na Figura 15, as linhas da tabela apresentam: i) o título “Clima Tempo”; ii) a mensagem “Sua aplicação de clima”;

iii) a previsão, enviada pelo Broadcaster, para Salvador (29 graus) e Rio de Janeiro (40 graus); e iv) a mesma previsão obtidas em um provedor de conteúdos na Internet.

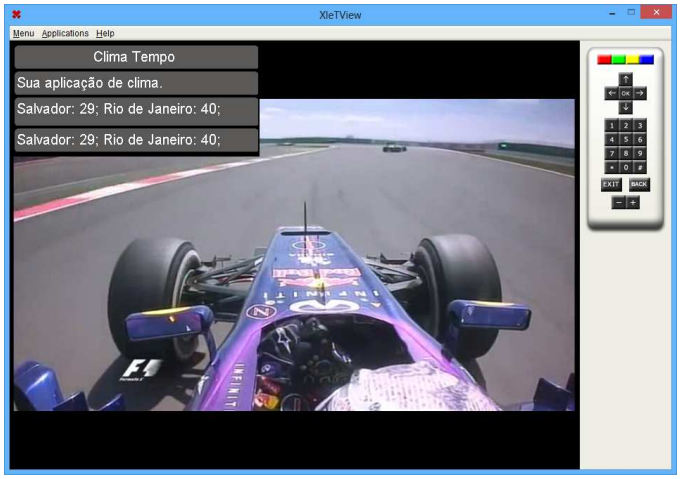

Figura 15. Tela do Receptor: Aplicação de previsão do tempo que consome de dados provenientes do Broadcaster e da Internet (Xlet Clima)

IX. Estudo de caso: Xlet Replay

O Xlet Replay foi mais uma aplicação JavaDTV desenvolvida com o objetivo de homologar o funcionamento do simulador construído. O Xlet Replay é uma aplicação genérica para reprodução de replays de um programa de televisão qualquer (Formula 1, Futebol, Lutas como MMA e etc.), esta aplicação consome dados (texto e vídeo) enviados de um Broadcaster e apresenta para o usuáiro dando a opção do mesmo selecionar a qualquer momento um Replay recebido para execução.

Para este estudo de caso foi simulado como programa de televisão a transmissão de uma corrida de Formula 1 (conteúdo principal), este âmbito foi escolhido pois nele Replays são constantes e de alto interesse dos espectadores. O que caracteriza esta aplicação como uma aplicação multimídia complexa é a possibilidade de consumir conteúdo extra que não é previamente conhecido pois não se sabe exatamente quando irá ocorrer uma batida ou um pit-stop pois se trata uma transmissão ao-vivo.

A. Arquitetura

A Arquitetura do Xlet Replay foi dividida em quatro módulos, um módulo de consumo de dados (Consumidor broadcaster), um módulo para ler o arquivo XML que será recebido do Consumidor broadcaster (Leitor XML), uma módulo para gerenciar a aplicação solicitando o consumo de dados e prover dados para o módulo de Apresentação, onde as informações serão apresentadas ao usuário. O funcionamento de cada um destes módulos será explicitado nos tópicos abaixo, o diagrama desta arquitetura encontra-se na Figura 16.

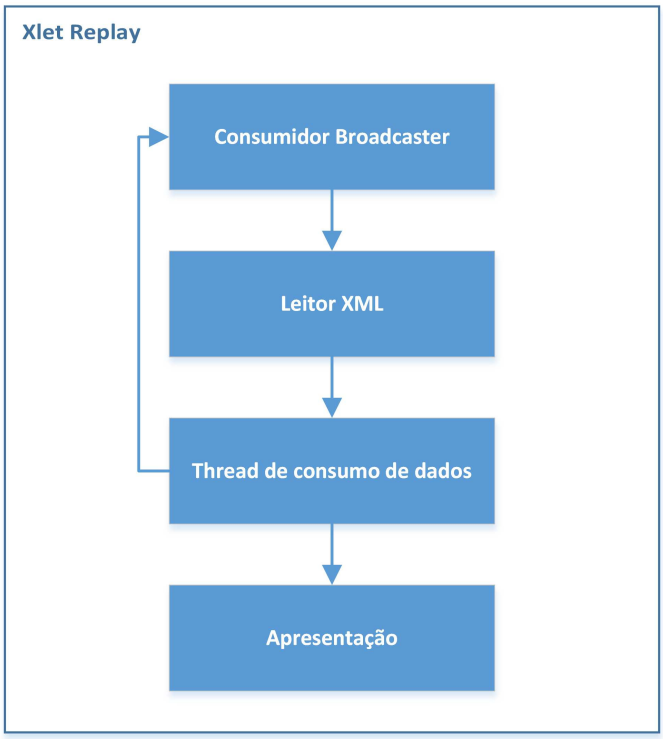

Figura 16. Arquitetura do Xlet Replay

1) Thread de consumo de dados: Módulo responsável por periodicamente solicitar atualizações de dados para o Consumidor Broadcaster. Ao fim de cada solicitação de dados este módulo envia informações para atualizações do módulo de apresentação.

2) Consumidor Broadcaster: Ao ser solicitado, este módulo verifica a existência de diretórios a serem consumidos na pasta de arquivos recebidos do receptor (estes arquivos chegam até esta pasta através do processo explicado na arquitetura do simulador). Cada diretório deve conter dois arquivos sendo um arquivo XML para descrição do conteúdo extra e um arquivo de vídeo. No caso da existência de conteúdo extra disponível os arquivos XML são enviados para o Leitor XML.

3) Leitor XML: Este módulo lê do disco (dado proveniente do Consumidor broadcaster) cada arquivo XML e encapsula as informações contidas no XML em um objeto que representa um replay dentro da aplicação. As informações encapsuladas são:

1) O caminho para o arquivo de vídeo (obtido através do diretório do XML).
2) O título do vídeo.
3) A descrição do conteúdo do vídeo (informação não utilizada para esta versão da aplicação).
4) Um número sequencial que indica a posição temporal da ocorrência do Replay.

Após o encapsulamento dos dados proveniente do XML uma lista de objetos que representam o Replay são enviados o módulo denominado Thread de consumo de dados. O código abaixo exemplifica a estrutura do XML consumido.

```
<root>
 <replays>
  <replay>
   <title>Largada</title>
   <description>
    Largada do Grand Prix
   </description>
   <sequence>0</sequence>
  </replay>
 </replays>
</root>
```

4) Apresentação: Este módulo recebe informações as informações do módulo Thread de consumo de dados e exibe as mesmas para o usuário utilizando a classes da biblioteca Lwuit (Ex: Form, List e etc).

Nesta aplicação é exibida uma lista (exibindo o nome de cada Replay) para que o usuário possa selecionar o Replay desejado para execução, ao selecionar um Replay o mesmo será exibido ao lado da lista de Replays disponíveis. Na Figura 17 o Xlet Replay pode ser visto em funcionamento.

Figura 17. Tela do Receptor: Aplicação para execução de replays para programas de televisão ao vivo (Xlet Replay)

X. Resultados

A execução da aplicação Xlet Clima utilizando o ambiente de simulação desenvolvido mostra que o mesmo dá suporte a testes de aplicações que consomem dados tanto provenientes do Broadcaster quanto da Internet, por outro lado a execução da aplicação Xlet Replay mostra que o ambiente de simulação suporta execuções e testes de aplicações que consomem dados e informações com relação semântica ao conteúdo principal exibido, mais especificamente, o ambiente de simulação desenvolvido dá suporte a aplicações voltadas para programas de televisão ao vivo.

Durante a validação do ambiente de simulação também foram testadas aplicações que consomem ou produzem dados e informações que não têm nenhum vinculo semântico com o conteúdo principal. Entre elas foram testadas:

1) Aplicação para envio de e-mail.
2) Aplicação para exibição da tabela de classificação do campeonato brasileiro (durante a transmissão de uma corrida de F1).
3) Aplicação que exibe uma imagem pre definida na tela do simulador.

Para execução de aplicações que consomem ou produzem dados e informações que têm relação com a semântica do conteúdo principal porém sem restrições fortes de sincronização foi utilizada a aplicação que exibe a tabela de classificação do campeonato brasileiro durante a exibição (conteúdo principal) de uma partida do campeonato brasileiro.

Com a execução das aplicações citadas acima podemos concluir que o ambiente de simulação dá suporte à:

1) Recepção de conteúdo principal (stream de áudio e vídeo) através da sintonização de um canal.
2) Recepção de conteúdo extra proveniente do Broadcaster através da sintonização de um canal.
3) Recepção de conteúdo extra proveniente da Internet através da utilização da aplicações JDTV.
4) Execução de teste em aplicações pertencente a todas as categorias de aplicações digitais interativas definidas em [8] [7] [4], são elas:
   a) Aplicações cujo os dados não têm vinculo semântico com o conteúdo principal.
   b) Aplicações cujo os dados têm relação semântica com conteúdo principal porém não existe fortes restrições de sincronização.
   c) Aplicações cujo os dados têm relação semântica e fortes restrições de sincronização (AMC).

Com a utilização do ambiente de simulação como um aparato para o desenvolvimento das aplicações podemos levantar os seguintes benefícios durante as fases de implementação e testes das aplicações:

1) Ambiente para teste de baixo custo, utilizando apenas dois computadores em rede.
2) Maior velocidade de desenvolvimento pois não foi necessário migrar o código para outro ambiente para realização de testes.
3) Menores riscos e maior segurança, visto que as aplicações em teste só podem causar danos ao ambiente de simulação.

XI. Conclusão

Este artigo apresentou uma ferramenta que consiste em uma infraestrutura para simulação de um receptor de TV Digital e TV Conectada. O que difere este trabalho dos outros é a possibilidade de recepção de conteúdo principal e extra através de uma rede TCP/IP e a execução de aplicações JDTV de todas as categorias descritas por [8] [7] [4]. As principais contribuições deste ambiente de simulação são:

1) permitir a execução de AMC usando computadores conectados em rede;
2) aumentar fidelidade dos testes para esta categoria de aplicações; e

3) a sua arquitetura modular que permite estender os artefatos de software produzidos de forma a adaptar o simulador para outros contextos.

O ambiente de simulação apresentado neste artigo permite a execução das aplicações interativas de TVD, porém, não está limitado somente a esta plataforma. Este diferencial permite que outras aplicações interativas possam usar do este ambiente de simulação, por exemplo, aquelas que são voltadas para IPTV uma vez que o simulador está estruturado em redes TCP/IP. Neste trabalho não foram avaliadas as restrições relacionadas com a rede de comunicação como a largura de banda ou a qualidade de vídeo para conteúdo principal. Este é um dos tópicos que será tratado nos próximos passos da evolução do sistema.

XII. Trabalhos Futuros

Para aprimoramento do ambiente de simulação desenvolvido visando contemplar um simulador para televisão digital interativa fiel e completo pode-se ressaltar alguns possíveis trabalhos futuros.

A. Implementação fiel do DSCM-CC

Neste trabalho o protocolo de comunicação implementado foi baseado no DSM-CC e contem um conjunto de funcionalidades reduzida devido a complexidade de desenvolvimento do DSM-CC em sua plenitude.

Um dos trabalhos futuros é a substituição do protocolo implementado pela implementação completa do protocolo DSM-CC, resultando assim em uma ambiente de simulação mais fiel e mais completo já que mensagens para transmissão de streams não foram implementadas nesta versão do simulador.

B. Implementação da maquina de apresentação Ginga-NCL

O ambiente de simulação desenvolvido contempla apenas execução de aplicações JDTV pois sua implementação conta apenas com a máquina de execução Ginga-J. Com a implementação do Ginga-NCL o ambiente de simulação iria contemplar todos os tipos de aplicações previstos para televisão digital tornando-se assim um ambiente mais fiel a realidade dos set up boxes utilizados para recepção de sinal digital.

C. Implementação da ponte entre o Ginga-J e o Ginga-NCL

Na arquitetura do middleware Ginga a máquina de execução Ginga-J e a máquina de apresentação Ginga-NCL podem trabalhar de forma independente ou em cooperação, visto a existência de uma “ponte” que disponibiliza mecanismos para intercomunicação entre os mesmos.

Através da implementação desta ponte ao ambiente de simulação será possível que aplicações imperativas utilizem serviços disponíveis nas aplicações declarativas, e vice-versa.

D. Implementação para contemplar as funções inovadoras do Ginga

O Ginga conta com funções que dão suporte ao desenvolvimento de aplicações para TV Digital multiusuário e multi dispositivo através da conexão (utilizando uma rede local) do receptor de TV com dispositivos móveis como smartphones e tablets.

A versão desenvolvida ambiente de simulação não dá suporte a execução de aplicativos multiusuário e multi dispositivo. A Adaptação deste trabalho para se adequar a execução de aplicativos com estas características aumenta a importância do simulador o tornando mais fiel ao ambiente real.

E. Implementação de aparatos para testes de aplicações

É muito comum em ambientes integrados de desenvolvimento a presença de aparatos para tornar a etapa de testes de software mais eficaz. Estes aparatos variam desde simples saídas de texto (outputs) onde o desenvolvedor pode acompanhar as saídas do programa como aparatos complexos para depuração linha a linha do código fonte executado.

O proposito do ambiente de simulação não é o mesmo de um ambiente de desenvolvimento, mas como uma das justificativas centrais para construção do mesmo é a possibilidade de execução de testes em AMCs tais aparatos são de grande serventia para aumentar o grau de contribuição que o simulador agrega aos testes em aplicações.

F. Considerações finais

A construção do simulador foi a primeira etapa de um projeto que visa apontar soluções para problemas apontados em [7] [6] e que são relacionados as aplicações multimídia complexas.

Referências

[1] Teixeira, H. P. Lauro. TELEVISÃO DIGITAL: Interação e Usabilidade. São Paulo. 2008.

[2] 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 2006. New York, NY, USA: ACM, 2006, pp. 203 a 212.

[3] S. Soursos and N. Doulamis, “Connected tv and beyond,” in Consumer Communications and Networking Conference (CCNC), 2012 IEEE, jan. 2012, pp. 582 a 586.

[4] Neto, C. M. Manoel. Contribuições para a modelagem de aplicações multimídia em TV digital interativa. 2011. Universidade Federal da Bahia.

[5] W. I. Grosky, C. Zhang, and S.-C. Chen, “Intelligent and pervasive multimedia systems,” MultiMedia, IEEE, vol. 16, no. 1, pp. 14 a 15, jan. march 2009.

[6] M. C. Maques Neto and C. A. Santos, “An approach based on events for treating the late tuning problem in interactive live tv shows,” in Proceedings of the 1st ACM international workshop on Events in multimedia. ACM, 2009, pp. 41 a 48.

[7] NETO,M. C. M.; SANTOS, C. A. An event-based model for interactive live tv shows. In: MM 2008: Proceeding of the 16th ACM international conference on Multimedia. New York, NY, USA: AC 2008. p. 845-848. ISBN 978-1-60558-303-7.

[8] VEIGA, E. G.; TAVARES, T. A. Um modelo de processo para o desenvolvimento de programas para tv digital e interativa. In: WebMedia 2006: Workshop de Teses e Dissertações. New York, NY, USA: ACM, 2006. p. 53-57. ISBN 85-7669-100-0.

[9] Kulesza R.; Ferreira J.; Filho, S. M.; Lívio Á. ; Brandão, R. R. M.; Araújo, J. P. C.; Filho, G. L. S. Ginga-J: Implementação de Referência do Ambiente Imperativo do Middleware Ginga. 2010.

[10] R. Kulesza, C. Santos, T. Tavares, M. Neto, and G. de Souza Filho, “Desenvolvimento de aplicações imperativas para tv digital no middleware ginga com java.”
[11] Fernandes, J.; Lemos, G.; Silveira, G.; Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação, JAI-SBC. Salvador. Anais do JAI-SBC, 2004.
[12] ITU-T. Recommendation J.200: Worldwide common core - Application environment for digital interactive television services. jul. 2012. Disponível em: http://www.itu.int
[13] Laureano, M. Máquinas Virtuais e emuladores. Conceitos,Técnicas e Aplicações. 2006. Novatec Editora, São Paulo.
[14] Silva, R. P. Fernanada. Xtation: Um ambiente para execução e teste de aplicações interativas para o middleware Ginga. 2010. Universidade federal da Paraíba.
[15] Soares, G. F. Luiz. TV interativa se faz com Ginga. 2011. Fórum SBTVD.
[16] S. Carvalho e V. Araújo, “Emuladores para TV Digital - OpenMHP e Xletview”, 2007.
[17] 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.
[18] D. Terra, N. Kumar, N. Lourenço, L. N. Alves, and R. L. Aguiar, “Design, development and performance analysis of dsss-based transceiver for vlc,” in EUROCON 2011, 2011, pp. 1 - 4.
[19] Weiser, M. The Computer for the 21st Century. Scientific American , September, 1991.
[20] NETO,M. C. M.; SANTOS, C. “StoryToCode a new model for specification of convergent interactive digital TV applications,” WebMedia 2009 Proceedings of the XV Brazilian Symposium on Multimedia and the Web
[21] PAGANI, M. Multimedia and Interactive Digital TV: Managing the Opportunities Created by Digital Convergence. Hershey, PA, USA: IGI Publishing, 2003. ISBN 1931777381.
[22] Weber, Tayse S., Um roteiro para exploração dos conceitos básicos de tolerância a falhas, 2002
[23] 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. 6164.
[24] MUELLER, Milton. Digital Convergence and its consequences. Javnost - The Public, Syracuse, v. 6, n.3, p.11-28, mar. 1999.
[25] ALVES, Luiz et al. Análise Comparativa de Metadados em TV Digital. In: II WORKSHOP DE TV DIGITAL DO SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, 24., 2006, Curitiba. Anais... São Paulo: USP, 2006. p. 87-98.