


Manual de desenvolvimento de aplicações interativas Ginga-J para os middlewares AstroTV e MOPA e envio de aplicações via plataforma Domm XStream

LAVID | Laboratório de Aplicações de Vídeo Digital
Departamento de Informática / Universidade Federal da Paraíba

Sumário

1. Introdução

1.1 O que é Ginga-J?

O Ginga-J é uma especificação que consiste em um conjunto de APIs concebidas para prover as funcionalidades necessárias no desenvolvimento de aplicações procedurais (utilizando a linguagem Java) para TV digital que possui o middleware Ginga. Sua arquitetura pode ser vista na figura 1.

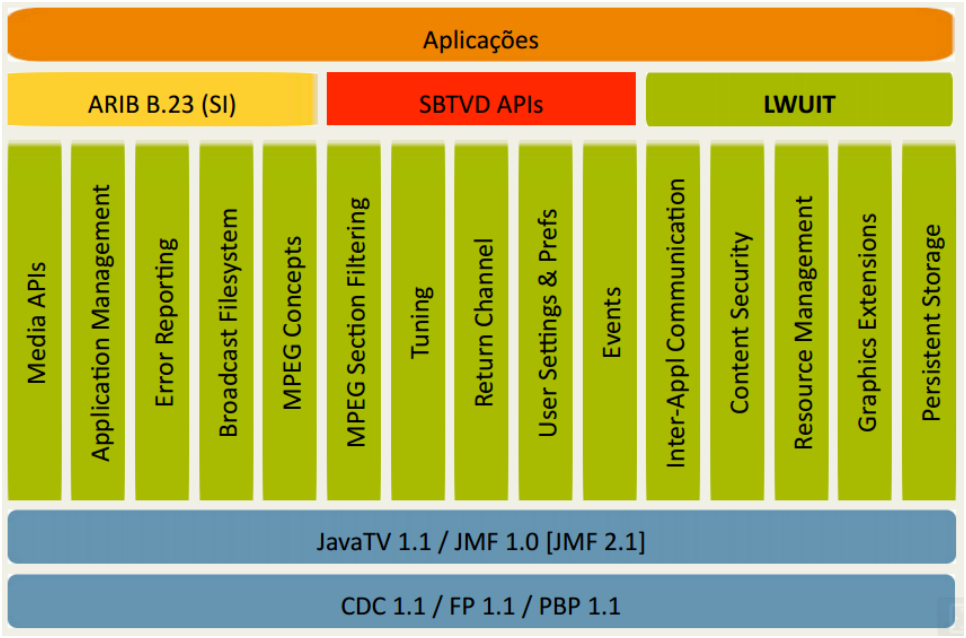

Figura 1. Arquitetura do Ginga-J [5]

1.2 Middlewares AstroTV e MOPA

O AstroTV e o MOPA são implementações da especificação do middleware Ginga feitas pelas empresas de softwares TOTVS e MOPA Embededd Systems, respectivamente.

Figura 2. Empresas que implementaram o Ginga

1.2 Domm XStream

O *Domm XStream* é um sistema que multiplexa e modula o sinal de TV Digital, gerando um fluxo MPEG-2 TS(Transport Stream) no padrão ISDB-T[2]. Esse fluxo possui dentre várias funcionalidades o Carrossel de Dados, que nos interessa, pois é o mecanismo cíclico responsável pelo envio das aplicações interativas para um receptor digital.

2. Objetivo

Esse guia tem como objetivo demonstrar a estrutura básica de um programa para Ginga-J nos middlewares AstroTV e MOPA, orientando programadores Java que desejam iniciar no desenvolvimento de aplicações interativas para essas implementações do Ginga. Também instrui os passos de configuração e envio de aplicações através da interface web do sistema *Domm XStream*.

3. Desenvolvendo aplicações procedurais para o AstroTV e MOPA.

Nesta parte iremos demonstrar como desenvolver uma aplicação procedural para o AstroTV e MOPA, pontuando algumas diferenças para ambas as implementações.

3.1 Implementando a interface Xlet

O primeiro passo no desenvolvimento de uma aplicação é criar uma classe que implemente a interface *Xlet* do pacote *javax.microedition.xlet,* tanto no AstroTV quanto no MOPA. A figura 3 demonstra um exemplo com a classe *AppXlet.* Foram importadas também a interface XletContext e a classe XletStateChangeException, pois as mesmas são necessárias nos métodos da interface implementada.

```
import javax.microedition.xlet.Xlet;
import javax.microedition.xlet.XletContext;
import javax.microedition.xlet.XletStateChangeException;

public class AppXlet implements Xlet {

    public void initXlet(XletContext context) throws XletStateChangeException {

    }

    public void startXlet() throws XletStateChangeException {

    }

    public void pauseXlet() {

    }

    public void destroyXlet(boolean flag) throws XletStateChangeException {

    }

}
```

Figura 3. Estrutura básica de uma aplicação para o AstroTV e MOPA.

Cada método da interface *javax.microedition.xlet.Xlet* manipula o ciclo de vida da aplicação, onde a implementação determina o comportamento para cada ciclo. Os métodos serão descritos abaixo:

initXlet: É o primeiro método a ser chamado em um Xlet, devendo ter a inicialização dos recursos utilizados pela aplicação, bem como a recepção por parâmetro de um objeto *XletContext* que possui informações do contexto de execução e mecanismos de notificação de mudanças de estado da aplicação.

startXlet: Método chamado para iniciar a execução da aplicação.

pauseXlet: Método chamado para pausar a aplicação minimizando seu consumo de recursos. A aplicação pode executar novamente chamando outra vez o método *startXlet*.

destroyXlet: Método que pode ser chamado para o término definitivo da execução de um Xlet.

3.2 Criando uma aplicação visual e interativa

Nesta parte iremos descrever como criar sua primeira aplicação no AstroTV e no MOPA usando a API LWUIT, que possui recursos para interface gráfica e tratamento de eventos do usuário.

3.2.1 Obtenção de recursos do dispositivo

Para o desenvolvimento de aplicações visuais, tanto no AstroTV quanto no MOPA, precisamos obter recursos do dispositivo para assim ter acesso ao tamanho da tela, que será

utilizado na definição de dimensões das componentes gráficas, generalizando a aplicação para diferentes tamanhos de visor.

Como exposto anteriormente, é recomendado que esta parte da aplicação seja inserida no método *initXlet*, como pode ser visto abaixo:

```
private Dimension screenResolution;

public void initXlet(XletContext context) throws XletStateChangeException {

    Device device = Device.getInstance();
    Screen currentScreen = device.getDefaultScreen();
    PlaneSetup planeSetup = null;
    Plane[] planes = currentScreen.getAllPlanes();

    for(int i=0; i < planes.length; i++) {
        Capabilities cap = planes[i].getCapabilities();
        if(cap.isGraphicsRenderingSupported()) {
            Plane plane = planes[i];
            planeSetup = plane.getCurrentSetup();

            break;
        }
    }
    screenResolution = planeSetup.getScreenResolution();
}
```

Figura 4. Código da obtenção dos recursos do dispositivo

Para obter as dimensões do dispositivo são necessárias algumas etapas:

1 – Obtenção de um objeto *Device* que representa o dispositivo em funcionamento;

2 – Captura de uma instância da classe *Screen* que representa a tela padrão do dispositivo;

3 – Captura de um array de objetos do tipo *Plane* que representam todos os planos de exibição disponíveis na tela padrão do dispositivo;

4 – Em seguida é feito um laço *for* para varrer todos os planos. É obtido o objeto *Capabilities* de cada plano que representa as capacidades do mesmo;

5 – Uma instrução *if* é feita para saber se o plano é o que possui capacidade de renderização gráfica;

6 – Caso seja o plano de renderização gráfica, é capturado um objeto *PlaneSetup* pela chamada *plane.getCurrentSetup*;

7 – Nesse ponto, chamamos o método *planeSetup.getScreenResolution*, atribuindo a resolução da tela à variável global *screenResolution* do tipo *com.sun.dtv.lwuit.geom.Dimension*.

3.2.2 Criando um *Form* para adicionar componentes visuais

Vamos instanciar um objeto com.sun.dtv.lwuit.*Form* que pertence a API LWUIT. Esse objeto será a raiz dos componentes visuais e terá as dimensões da tela do dispositivo obtidas

através do procedimento descrito anteriormente. Em seguida iremos definir o gerenciador de layout *CoordinateLayout* para o *Form* e a cor de background do mesmo.

No AstroTV, para atribuir cor de background, a chamada *getStyle().setBgColor(Color color)* deverá ser efetuada. No nosso exemplo, definiremos a cor branca para o *Form*, como visto na figura 5. Caso desejado que o Form seja transparente, a chamada *form.getStyle().setBgTransparency(0)* deve ser realizada.

```
public void startXlet() throws XletStateChangeException {

    form = new Form();
    form.setSize(screenResolution);
    form.setLayout(new CoordinateLayout(screenResolution));
    form.getStyle().setBgColor(Color.WHITE);

    form.show();

}
```

Figura 5. Código da criação, configuração e exibição de um Form para o AstroTV.

No MOPA, para determinar a cor ao background do *Form*, deve-se instanciar um objeto *com.sun.dtv.lwuit.plaf.Style,* definir a cor com o método *setBgColor(Color color)* e posteriormente atribuir ao *Form* com a chamada *setStyle(Style style)*, como podemos ver na figura 6. Para qualquer outra alteração no estilo de uma componente no MOPA (p.e. transparência de um *Form*), deve-se realizar o mesmo procedimento.

```
public void startXlet() throws XletStateChangeException {

    form = new Form();
    form.setSize(screenResolution);
    form.setLayout(new CoordinateLayout(screenResolution));

    Style style = new Style();
    style.setBgColor(Color.WHITE);
    form.setStyle(style);

    form.show();

}
```

Figura 6. Código da criação, configuração e exibição de um *Form* para o MOPA.

O campo que terá a referência do objeto *Form* instanciado deverá ser global, pois esse requisito será necessário posteriormente, quando for acrescentado o tratamento de eventos do usuário.

A exibição do *Form* na tela para ambos os middlewares, depende da chamada *form.show()* após todos os passos citados acima.

A execução da aplicação no momento atual simplesmente deixará branca a tela do dispositivo, como pode ser visto na simulação da figura 7.

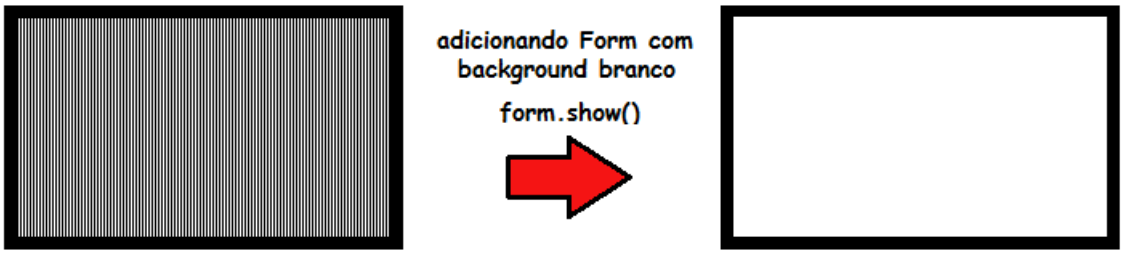

Figura 7. Simulação da tela da TV antes e depois da adição do Form .

3.2.3 Adicionando uma imagem na tela

Uma das formas de adicionar uma imagem no *Form* é instanciando um objeto *Label* que a receberá através do método *setIcon(Image.createImage(String path))*, onde o caminho da imagem é considerado a partir do diretório raiz do projeto. Esse procedimento é idêntico para ambos os middlewares.

Por questões de padrão, assim como no *Form*, o tamanho da tela do dispositivo é utilizado para determinar as dimensões e o posicionamento do *Label*, como visto na figura 8.

A transparência de background do *Label* deve ser atribuída de forma análoga à descrita na seção anterior, quando o componente utilizado foi um *Form*. Sendo este procedimento particular para cada implementação.

Após instanciado, o *Label* é adicionado ao *Form* com o código *form.addComponent(label)* e a chamada *form.show()* pode ser feita no fim da aplicação.

```
public void startXlet() throws XletStateChangeException {

    form = new Form();
    form.setSize(screenResolution);
    form.setLayout(new CoordinateLayout(screenResolution));
    form.getStyle().setBgColor(Color.WHITE);

    int dim = (int) (screenResolution.getHeight() * 0.3);

    label = new Label();
    label.getStyle().setBgTransparency(0);
    label.setSize(new Dimension(dim,dim));
    label.setX(screenResolution.getWidth()/3);
    label.setY(screenResolution.getHeight()/5);

    try{
        label.setIcon(Image.createImage("img.png"));
    }catch(IOException e){
        e.printStackTrace();
    }

    form.addComponent(label);
    form.show();

}
```

Figura 8. Adicionando uma imagem através de um *Label* para o AstroTV

```
public void startXlet() throws XletStateChangeException {

    form = new Form();
    form.setSize(screenResolution);
    form.setLayout(new CoordinateLayout(screenResolution));

    Style styleForm = new Style();
    styleForm.setBgColor(Color.WHITE);
    form.setStyle(styleForm);

    int dim = (int) (screenResolution.getHeight() * 0.3);

    label = new Label();

    Style styleLabel = new Style();
    styleLabel.setBgTransparency(0);

    label.setStyle(styleLabel);
    label.setSize(new Dimension(dim,dim));
    label.setX(screenResolution.getWidth()/3);
    label.setY(screenResolution.getHeight()/5);

    try{
        label.setIcon(Image.createImage("img.png"));
    }catch(IOException e){
        e.printStackTrace();
    }

    form.addComponent(label);
    form.show();

}
```

Figura 9. Adicionando uma imagem através de um *Label* para o MOPA

Podemos ver na figura 10 a simulação de como deve ficar a tela após a execução da aplicação no estado atual. A imagem escolhida para esta aplicação teve como critério o auxílio intuitivo da próxima seção que é interatividade via controle remoto.

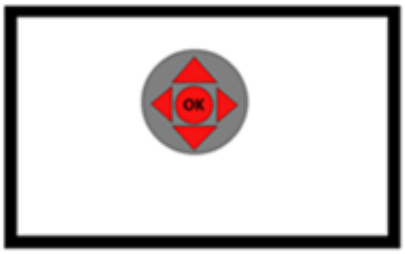

Figura 10. Simulação da tela da TV após a adição de um *Label* com imagem.

3.2.4 Interface ActionListener para interação do usuário no AstroTV

Para acrescentar a interação na nossa aplicação no middleware AstroTV, precisamos utilizar uma das APIs providas pelo Ginga-J para tal procedimento. A interface *ActionListener* pode ser uma opção para aplicações visuais, pois está associada a ocorrência de eventos em componentes do LWUIT. Ao implementar o método *actionPerformed(ActionEvent e)* definimos o comportamento da aplicação perante uma interação, prevista no método, feita pelo usuário. Os possíveis eventos da especificação Ginga-J são representados pelas constantes definidas na classe *RemoteControlEvent* e são exibidos na tabela 1.

Funções definidas na ABNT NBR 15604:2007		**Classe com.sun.dtv.lwuit.event.RemoteControlEvent**
Confirma		VK_CONFIRM
Sair		VK_ESCAPE
Voltar		VK_BACK
Direcional	Acima	VK_UP e VK_KP_UP
	Abaixo	VK_DOWN e VK_KP_DOWN
	Esquerda	VK_LEFT e VK_KP_LEFT
	Direita	VK_RIGHT e VK_KP_RIGHT
Coloridas	Vermelha	VK_COLORED_KEY_0
	Verde	VK_COLORED_KEY_1
	Amarelo	VK_COLORED_KEY_2
	Azul	VK_COLORED_KEY_3

Tabela 1. Constantes da classe *RemoteControlEvent* que representam as interações [1].

A figura 11 exibe o código-fonte da função addKeyEvent(), criada para nossa aplicação e responsável pela implementação de uma instância de *ActionListener* por classe interna anônima, definindo o tratamento dos eventos de *setas de seleção* e *ok* do controle remoto. Também realiza mapeamento do objeto criado com a componente *Form* com a chamada *form.addKeyListener(..., action)* para cada evento desejado.


```
public void addKeyEvent(){
   action = new ActionListener(){

     public void actionPerformed(ActionEvent event) {
        int code = event.getKeyEvent();

        if(code == RemoteControlEvent.VK_UP){
           if(label.getY() > 0){
              label.setY(label.getY() - form.getHeight()/10);
              form.repaint();
           }
        }else if(code == RemoteControlEvent.VK_DOWN){
           if(label.getY() < form.getHeight()){
              label.setY(label.getY() + form.getHeight()/10);
              form.repaint();
           }

        }else if(code == RemoteControlEvent.VK_RIGHT){
           if(label.getX() < form.getWidth()){
               label.setX(label.getX() + form.getWidth()/10);
               form.repaint();
           }

        }else if(code == RemoteControlEvent.VK_LEFT){
           if(label.getX() > 0){
              label.setX(label.getX() - form.getWidth()/10);
              form.repaint();
           }
        }else if(code == RemoteControlEvent.VK_CONFIRM){
           try {
              label.setIcon(Image.createImage("ginga.png"));
           } catch (IOException e) {
              e.printStackTrace();
           }
           form.repaint();
           removeEvents();

        }

     }

   };
       /* Adiciona listeners de eventos ao Form */
   form.addKeyListener(RemoteControlEvent.VK_UP, action);
   form.addKeyListener(RemoteControlEvent.VK_DOWN, action);
   form.addKeyListener(RemoteControlEvent.VK_LEFT, action);
   form.addKeyListener(RemoteControlEvent.VK_RIGHT, action);
   form.addKeyListener(RemoteControlEvent.VK_CONFIRM, action);

}
```

Figura 11. Código de tratamento de eventos do usuário

As setas de seleção causam um deslocamento de 10% do tamanho da tela na sua direção para cada interação que o usuário realiza. A tecla *OK* causa a mudança da imagem e a perda da capacidade de interagir, chamando o método *removeEvents()*, feito para aplicação, contendo o código mostrado na figura 12. Uma simulação da execução da aplicação é demonstrada na figura 13.

```
public void removeEvents(){
    /*  Remove tratamento de eventos */
    form.removeKeyListener(RemoteControlEvent.VK_UP, action);
    form.removeKeyListener(RemoteControlEvent.VK_DOWN, action);
    form.removeKeyListener(RemoteControlEvent.VK_LEFT, action);
    form.removeKeyListener(RemoteControlEvent.VK_RIGHT, action);
    form.removeKeyListener(RemoteControlEvent.VK_CONFIRM, action);
}
```

Figura 12. Remoção de listeners do Form

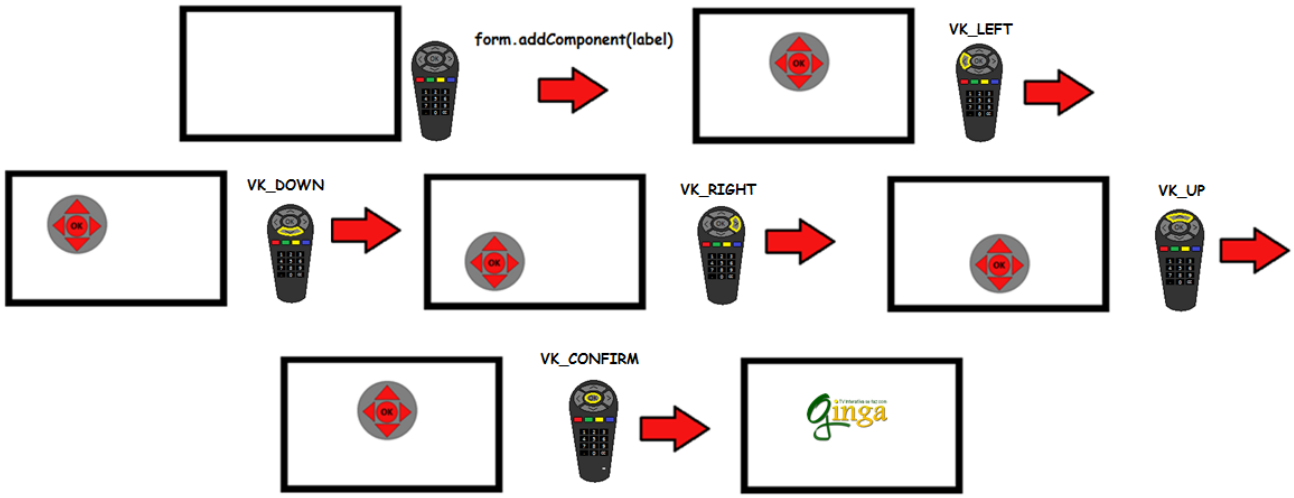

Figura 13. Simulação da execução da aplicação na TV com interações via controle remoto.

Existem muitas componentes na biblioteca LWUIT que podem ser utilizadas na criação de uma aplicação, bem como outras formas de tratamento de eventos. Esse exemplo tem o intuito de familiarizar programadores iniciantes nesse tipo de desenvolvimento, pois a plataforma é bastante extensa e provê uma enorme gama de possibilidades para aplicações bem mais complexas.

4. Enviando aplicações com o Domm XStream

Após desenvolvermos a nossa primeira aplicação interativa, podemos enviá-la para um dispositivo de Ginga. No nosso caso, será realizado o procedimento de envio para uma TV Digital através de uma infraestrutura de estação de trabalho para pequenas transmissões utilizando o sistema *Domm XStream* (figura 14). Será explicado como enviar suas aplicações através da interface web do sistema.

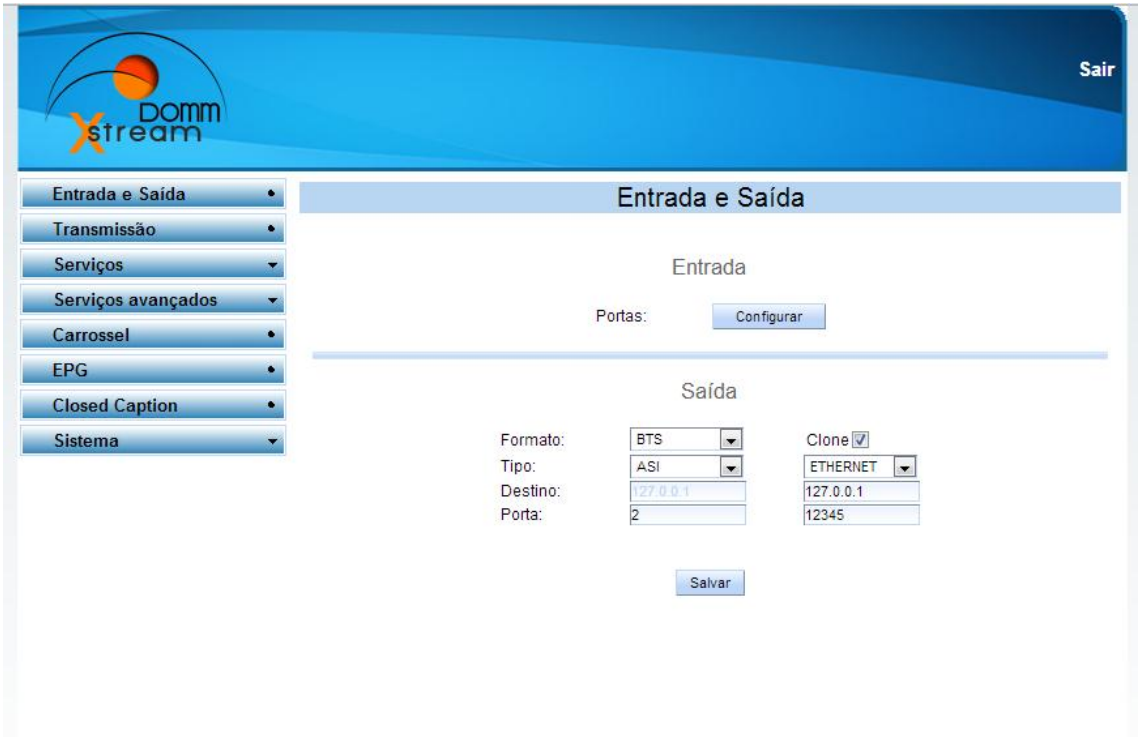

Figura 14. Screenshot da interface web do Domm XStream.

4.1 Fazendo upload da aplicação

Para enviar uma aplicação via Domm XStream é preciso fazer upload da mesma seguindo algumas instruções:

1. Colocar o arquivo *.class* da classe que implementa a interface *Xlet* dentro de uma pasta, junto com outras classes (também .class) e recursos da aplicação. O nome da pasta não precisa ser igual ao nome da aplicação;
2. Comprimir a pasta da aplicação para um arquivo no formato ZIP;
3. No *Domm XStream*, entre na opção Carrossel e escolha o serviço do TS que irá conter a aplicação (Figura 15);
4. Clique no botão *Selecionar* e faça o upload da pasta da aplicação (Figura 16);
5. Selecione *GINGA-J* como o tipo da aplicação;
6. O campo *Código Controle* é referente ao comportamento da TV com a chegada da aplicação. A opção *AUTOSTART* faz a aplicação executar assim que carregar.
7. O *Nome* da aplicação pode ser qualquer um, não tendo relação com o nome do arquivo *.class*;
8. O campo *Classe Inicial* deve ser preenchido com o nome da aplicação enviada, *sem a extensão .class*.

Depois dos passos acima, basta clicar no botão *Salvar* que a aplicação será enviada (figura 17), estando multiplexada com o conteúdo audiovisual.

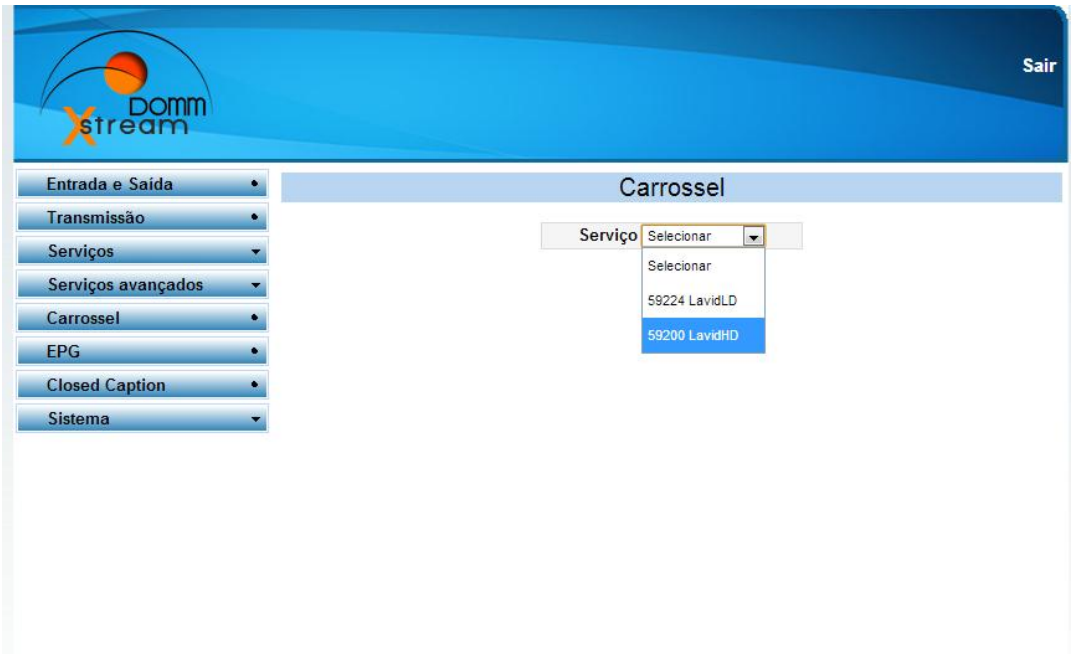

Figura 15. Escolha do Serviço do Carrossel.

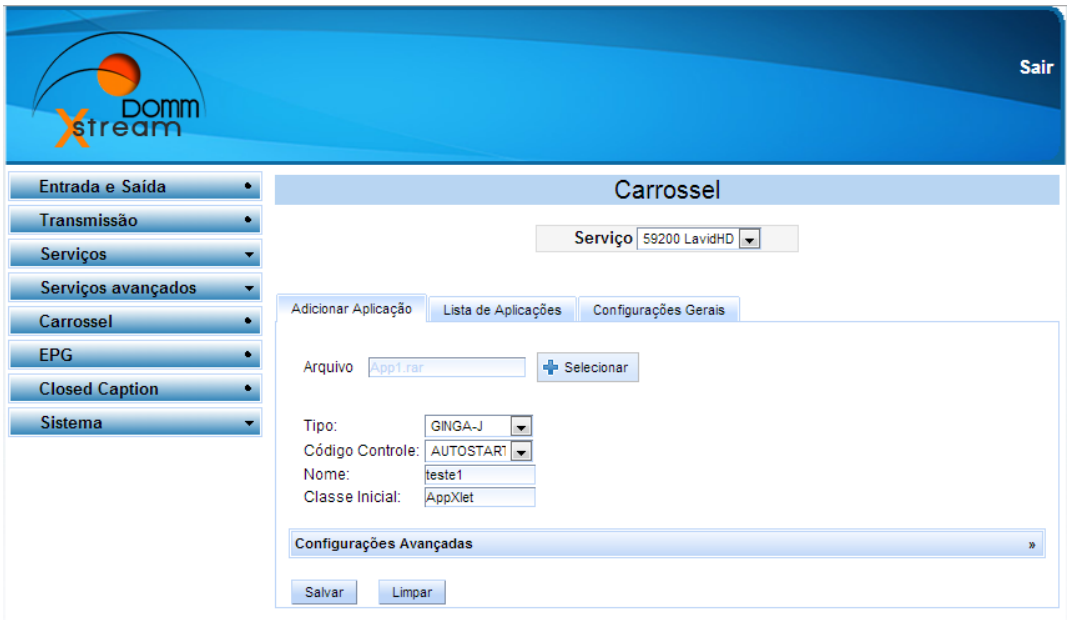

Figura 16. Upload e atributos da aplicação antes do envio.

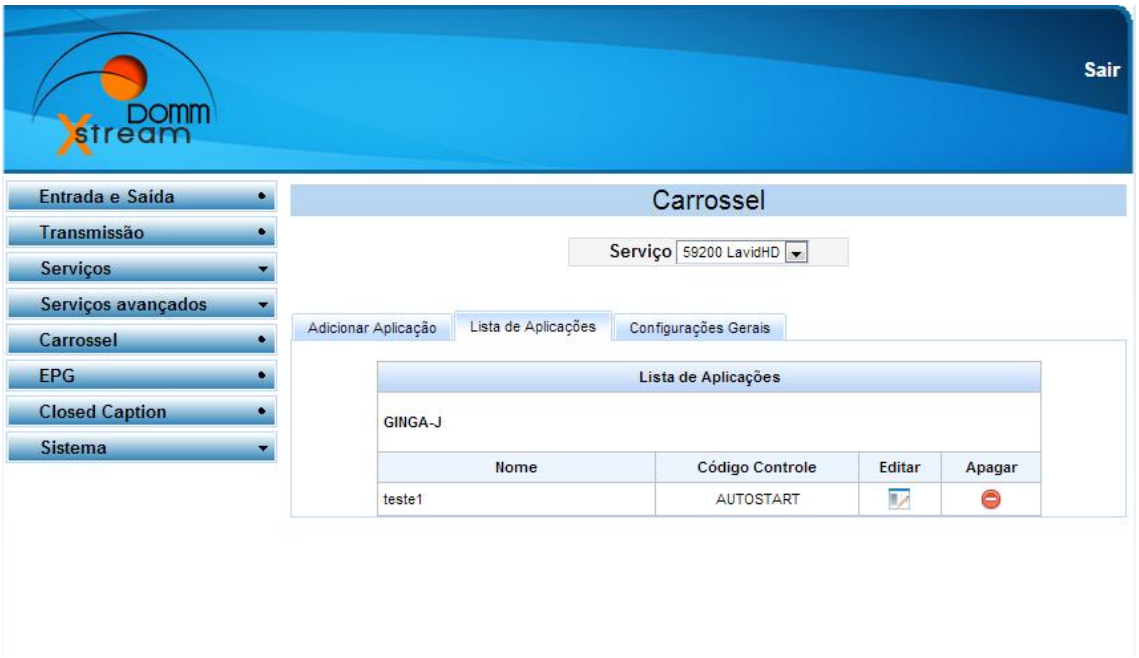

Figura 17. Aplicação já está sendo enviada para TV.

6. Referências

[1] ABNT NBR 15606-4 (2010) "Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais". *NBR 15606-4*.

[2] Descrição do sistema Domm xStream no site da empresa Domm. Diponível em: http://www.domm.com.br/xstream.jsf Acessado em: 08/02/2013

[3] KULESZA, Raoni; SAIBEL SANTOS A.C.; AIRES TAVARES T.; MARQUES NETO M.; SOUSA FILHO G. L. Desenvolvimento de Aplicações Imperativas para TV Digital no middleware Ginga com Java.

[4] KULESZA, Raoni. Criando e controlando aplicações Java no Ginga. Disponível em: http://www.tvdi.inf.br/site/artigos/Ginga-J/Desenvolvimento%20Ginga-J,%20JavaDTV,%20OpenGinga%20-%20Parte%202%20-%20KULESZA,%20FERREIRA.pdf . Acessado em: 08/02/2013.

[5] KULESZA, Raoni; FERREIRA Jefferson. Desenvolvimento Ginga-J. JavaDTV – OpenGinga. Disponível em http://www.tvdi.inf.br/site/artigos/Ginga-J/Desenvolvimento%20Ginga-J,%20JavaDTV,%20OpenGinga%20-%20Parte%201%20-%20KULESZA,%20FERREIRA.pdf. Acessado em: 08/02/2013.