Universidade Federal da Paraíba

Laboratório de Aplicações de Vídeo Digital - LAVID

Plaver Framework

Manual do Usuário

João Pessoa, 31 de janeiro de 2013

Sumário

1. Introdução

O framework Plaver foi desenvolvido com o intuito de auxiliar no desenvolvimento de aplicações de testes no ambiente de desenvolvimento Java padrão (Ginga-J) do Sistema Brasileiro de TV Digital (SBTVD) de uma forma rápida e eficaz.

O Plaver foi projetado para realizar testes mais unitários possíveis, podendo, dessa forma, testar método-a-método a Especificação Ginga-J, o que torna mais fácil localizar e reportar possíveis falhas. O resultado de cada teste é armazenado e convertido em uma cadeia de caracteres que será exibida ao final dos testes através de uma imagem QR Code.

Dentre suas funcionalidades, estão:

- Desenvolvimento através de testes unitários;
- Testes individuais (a falha em um teste não influenciará nos demais);
- Testes de aplicações visuais;
- Testes de aplicações não visuais;
- Geração de resultado em QR Code.

Este framework é o resultado de um trabalho conjunto de professores e estagiários do Laboratório de Aplicações de Vídeo Digital (LAViD), trabalho esse que foi desenvolvido e aprimorado em um período aproximado de 4 (quatro) meses.

Acreditamos que, ao ler esse manual, o leitor possa facilmente utilizar o Plaver para a realização de qualquer conjunto de testes para a especificação Ginga-J.

2. Arquitetura

O framework Plaver é constituído de 2 (dois) módulos. O primeiro, **Módulo de Controle**, é responsável por todo controle da suíte de aplicações que será testada. O segundo, **Módulo de Resultado**, será responsável pela exibição do resultado dos testes através da geração de uma imagem QR Code que conterá uma String resultante dos testes.

Assim como toda aplicação Ginga-J, o framework usará uma Xlet, chamada **TestSuitXlet**, que além de carregar e iniciar a Xlet, é o ponto de partida e gerenciamento da suíte de testes. Esta se utiliza dos módulos citados e de um **arquivo de configuração** para selecionar informações a respeito do diretório das classes testadas, além de configurações de timeout e definição de tipos de aplicações.

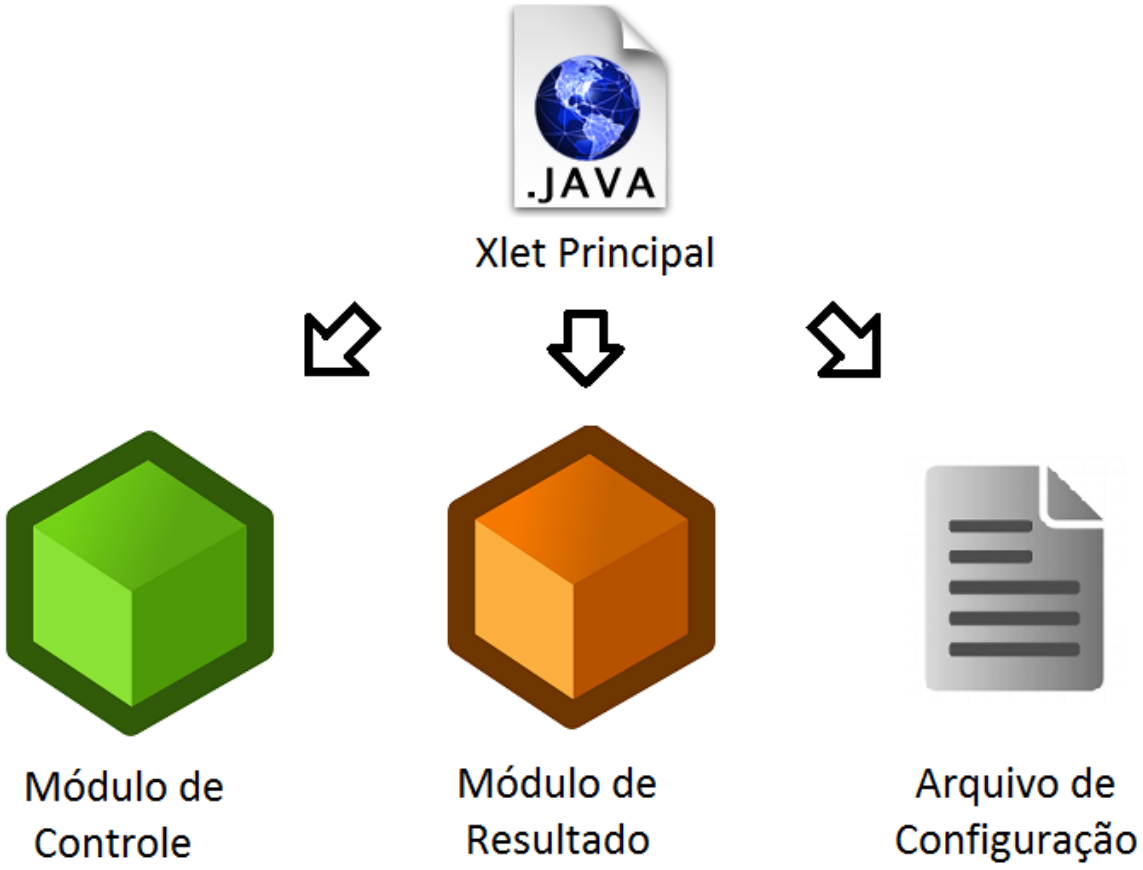

Figura 1. Visão geral do Plaver

2.1. Módulo de Controle

2.1.1. Descrição

Módulo responsável pela criação e controle de execução dos testes. Nele será realizada a leitura e instanciação das classes contidas no arquivo de configuração e, a partir de então, iniciar as aplicações uma a uma.

Sempre que um teste for executado corretamente, será gerado um evento que notificará o módulo de controle sobre o resultado do teste. Caso a aplicação não gere o evento dentro do tempo limite especificado no timeout do arquivo de configuração, será gerado, automaticamente, um evento para finalizar a thread e o teste será tido como falho.

Estado da Execução	Descrição
SUCCESS	Sinaliza que o teste foi finalizado e bem sucedido
FAILURE	Sinaliza que o teste foi finalizado e mal sucedido
TIMEOUT	Por algum motivo o teste não pôde ser realizado até o final da execução. Então, o módulo de controle irá gerar a interrupção da execução
VISUAL_APP	Sinaliza que uma aplicação visual terminou sua execução

Tabela 1. Estados de Execução

2.1.2. Diagrama de Classes

Figura 2. Diagrama de Classes do Módulo de Controle

2.1.3. DTVTest

Interface que deve ser implementada por **todas** as classes que irão realizar testes. Através dela serão chamados, de uma forma sequencial, os métodos que são descritos na Tabela 2.

Método	Descrição
setForm()	Atribui o único Form que será utilizado em todas as aplicações de teste. Dessa forma, evita-se a sobreposição de Forms diferentes. O Form deve apenas ser utilizado em aplicações visuais
setLog()	Atribui o log responsável pela notificação de sucesso, falha ou finalização de aplicação visual. O objeto atribuído deve ser utilizado em todas as aplicações
init()	Responsável pela instanciação de atributos, alocação de recursos e demais operações que venha a ser necessária para a realização do teste
start()	Responsável pela efetiva realização do teste. É neste método que o resultado do teste deverá ser notificado através das chamadas: log.success(), log.failure() ou log.visualAppFinished()
pause()	Pausa a execução do teste. Geralmente não há uma utilização prática
destroy()	Utilizado para remover listeners e liberar recursos alocados, caso existam. Em aplicações visuais, também é utilizado para remover componentes adicionados ao Form que não serão mais utilizados

Tabela 2. Métodos DTVTest

2.1.4. DTVTestController

Classe principal do Módulo de Controle que será responsável pelo controle dos testes. Através da chamada dos seus dois métodos principais (ver Tabela 3) serão iniciados os processos de controle e gerenciamento das aplicações.

Principais Métodos DTVTestController	
createTestSuit(...)	Responsável pela leitura, instanciação e configuração das classes contidas no **arquivo de configuração**. Além da criação da lista de aplicações
start()	Responsável pela criação de uma nova linha de execução que executará a chamada dos métodos: setForm(...), setLog(...), init() e start() da atual aplicação em execução.

Tabela 3. Principais Métodos da classe DTVTestController

2.1.5. DTVTestEvent

Define os eventos de resultado dos testes: TEST_FAILURE, TEST_SUCCESS, TEST_TIMEOUT. Caso a aplicação seja visual, o estado do resultado será definido por: TEST_VISUAL_APP. A descrição de cada evento pode ser vista na Tabela 1.

2.1.6. DTVTestListener

Interface listener para a recepção de eventos que informarão sobre o resultado da execução de cada teste.

2.1.7. DTVTestLog

Define o comportamento a ser apresentado pelas aplicações em caso dos testes serem bem sucedidos ou não. As classes de testes devem manter relacionamento com esta classe, a fim de se adequarem ao comportamento esperado com relação ao resultado dos testes.

Métodos de Notificação	
success()	Teste não visual obteve sucesso
failure()	Teste não visual falhou
timeout()	Teste não visual não respondeu no intervalo de tempo especificado
visualAppFinished()	Teste visual terminou sua execução

Tabela 4. Métodos de Notificação de Eventos da classe DTVTestLog

2.2. Módulo de Resultado

2.2.1. Descrição

Módulo responsável pela conversão dos resultados em alfanumérico e geração de uma imagem QR Code que será exibida ao final dos testes.

2.2.2. Diagrama de Classes

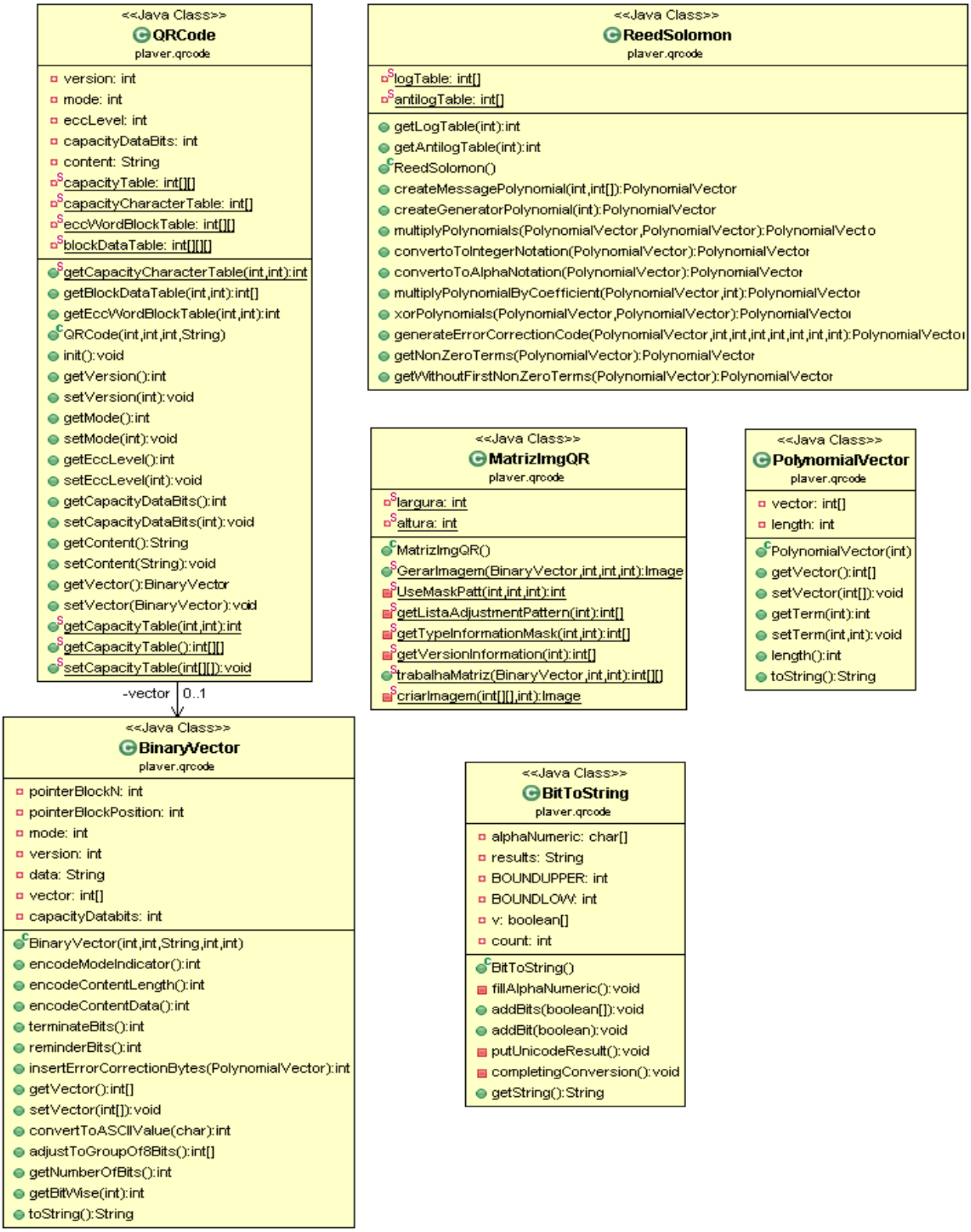

Figura 3. Diagrama de Classes do Módulo de Resultados

2.2.3. BitToString

Responsável pela conversão do vetor de booleanos, que representa o resultado de todos os testes, em uma String alfanumérica com 32 caracteres distintos (0-9 e A-V). O vetor booleano será interpretado a cada 5 (cinco) elementos. Caso o total de testes não seja múltiplo de 5 (cinco), esta classe tratará de complementar o conjunto com valores de "false".

O objetivo desta classe é diminuir a representação do resultado para que possa, mais facilmente, ser adicionada a um QR Code.

Vetor de Booleano	Resultado de Conversão
{false}	0
{true}	G
{true, false, false}	G
{true, true, true, true, true}	V
{true, true, true, true, true, true}	VG
{true, true, true, true, true, true true, true, true, true, true}	VVG

Tabela 5. Exemplos de conversão de booleano para alfanumérico de 32 caracteres

2.2.4. BinaryVector

[Não há documentação. Classe não mais utilizada na nova versão]

2.2.5. MatrizImgQR

Responsável pela transformação do vetor de bits codificados em uma matriz. Compõe o conjunto de manipulações e etapas que tornam possível a consolidação de uma matriz como representação de uma imagem final de QR Code.

A classe MatrizImgQR também lida diretamente com manipulação simples de imagem em Java, tornando possível a composição da imagem final pixel a pixel relacionado a matriz de representação.

2.2.6. PolynomialVector

Responsável por implementar um polinômio como um vetor inteiro e contém vários métodos para manipulá-lo.

2.2.7. QRCode

Esta classe é uma abstração de QRCode definido pela ISO / IEC 18004:2000 (E), implementado como um objeto com métodos para acessar seus elementos.

2.2.8. ReedSolomon

Classe que contém métodos necessários para implementar o algoritmo Reed-Solomon. Os métodos reproduzem as operações desse algoritmo usando as variáveis logTable e antilogTable para acessar os valores fora do Galois Field Arithmetic.

2.3. Arquivo de Configuração

Arquivo de texto responsável por configurar a suíte de testes, definindo as aplicações que serão testadas e o diretório até elas, além de configurar individualmente cada teste em relação ao modo de execução, ID e timeout. Mas detalhes sobre cada configuração possível pode ser visto na Tabela 6.

Timeout Default	O timeout que será padrão a todas as aplicações que não tiverem seus timeouts especificados
Classname	Classe a ser testada que implementa a interface DTVTest. O diretório dessa classe deve ser especificado a partir do diretório raiz que conterá a Xlet principal (i.e., TestSuitXlet)
ID	Identificador do teste. Esse item não necessariamente precisa estar ordenado
Modo de Execução	Responsável por informar ao Módulo de Controle a forma de execução do teste. Há dois modos de execução: automático (ou não visuais) e interativo (ou visual)
Timeout	O timeout de um teste específico. É utilizado caso alguma aplicação necessite de mais tempo do que o especificado no Timeout Default

Tabela 6. Possíveis configurações do Arquivo de Configuração

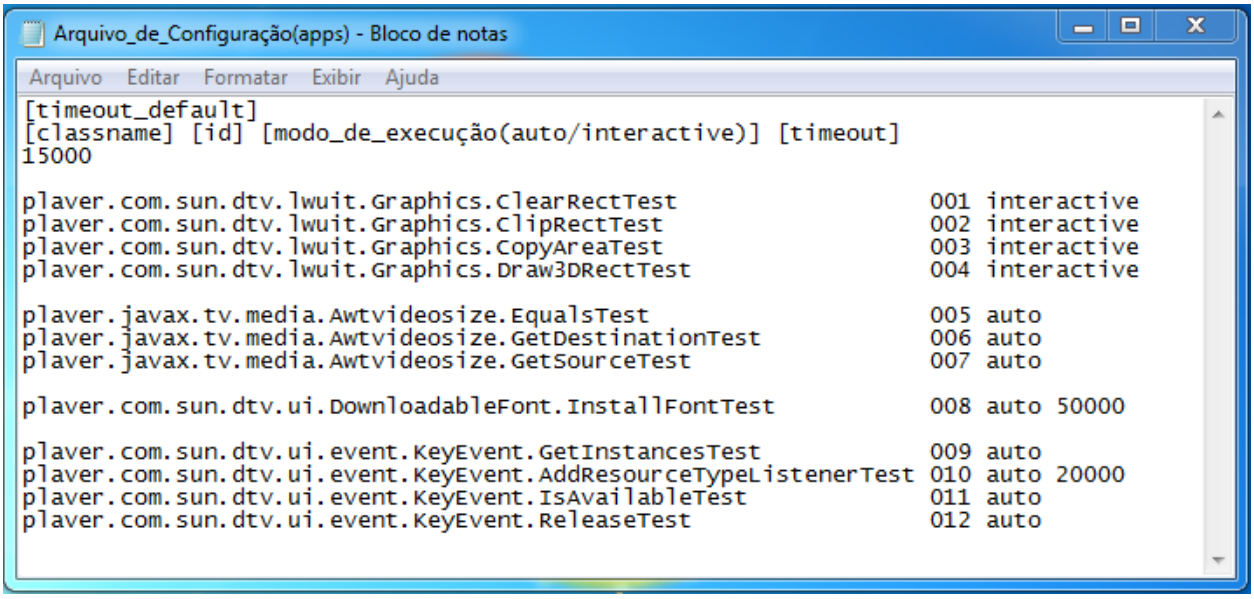

Figura 4. Arquivo de Configuração

2.4. TestSuitXlet

A Xlet principal do framework. É a partir dela que serão iniciados os módulos de controle e resultado.

Principais Métodos	
initXlet(...)	Inicia a leitura e instanciação das classes presente no Arquivo de Configuração
setPlane(...)	Configura o Plane e o Form que serão usados em todas as aplicações visuais
startXlet()	Inicia o Módulo de Controle, inicializando a primeira aplicação de teste da lista
testEventLogReceived(...)	Após ser notificado por algum evento (ver Tabela 4), este método iniciará a execução do próximo teste, caso ainda exista. Caso contrário, os testes serão finalizados e será feita a chamada do método de exibição de QR Code
exibeQRCode()	Inicia o Módulo de Resultado que irá converter o Vetor de Booleano em String e irá gerar uma imagem QR Code a partir dela

Tabela 7. Principais Método da classe TestSuitXlet

3. Usando o Framework

3.1. Aplicações Não Visuais

Para testar esse tipo de aplicação, dois passos são necessários:

1 – Implementar os métodos da Interface plaver.dtvtest.DTVTest;

2 – Criar um atributo do tipo DTVTestLog e atribuir o objeto "log" (Figura 4, linha 15);

```
1  import com.sun.dtv.lwuit.Form;
2
3  import plaver.dtvtest.DTVTest;
4  import plaver.dtvtest.DTVTestLog;
5
6  public class APPNãoVisual implements DTVTest{
7
8      private DTVTestLog dtvTestLog;
9
10     public void setForm(Form form) {
11
12     }
13
14     public void setLog(DTVTestLog log) {
15         this.dtvTestLog = log;
16     }
17
18     public void init() {
19
20     }
21
22     public void start() {
23
24     }
25
26     public void pause() {
27
28     }
29
30     public void destroy() {
31
32     }
33
34 }
35
```

Figura 4. Modelo de Aplicação Não Visual

Após esses passos, a implementação do teste será realizada de acordo com a funcionalidade do método a ser testado. Métodos com retorno são, de modo geral, mais fáceis de serem testados, uma vez que o valor/referência retornado pode ser comparado a resultados esperados.

Métodos sem retorno, no entanto, não têm uma "receita" para realizar os testes. O desenvolvedor terá que encontrar uma forma de verificar se o método testado realizou sua tarefa corretamente. Alguns métodos sem retorno que alteram o estado do objeto, como métodos "sets" e "adds", podem ser verificados através da chamada e verificação de métodos "gets". O código de programa abaixo testa o método getProtocol() da classe javax.media.MediaLocator.

```
1  package plaver.javax.media.MediaLocator;
2
3  import java.net.MalformedURLException;
12
13 public class GetProtocolTest implements DTVTest {
14
15     private static final String PROTOCOLO = "file";
16     private static final String LOCATOR = "file:cagin/javax/media/Manager/medias_jmf/v1.MPG";
17
18     private MediaLocator mediaLocator;
19     private DTVTestLog dtvTestLog;
20     private URL url;
21
22     public void setForm(Form form) {
23
24     }
25
26     public void setLog(DTVTestLog log) {
27         this.dtvTestLog = log;
28     }
29
30     public void init() {
31
32         try {
33             url = new URL(LOCATOR);
34         } catch (MalformedURLException e) {
35             e.printStackTrace();
36         }
37
38         mediaLocator = new MediaLocator(url);
39
40     }
41
42     /** Testa getProtocol() verificando se o retorno corresponde ao protocolo da mídia construída.
43      *
44      *  @see MediaLocator#getProtocol()
45      */
46     public void start() {
47         if(mediaLocator.getProtocol().equalsIgnoreCase(PROTOCOLO))
48             dtvTestLog.success();
49         else
50             dtvTestLog.failure();
51     }
52
53
54     public void pause() {
55
56     }
57
58     public void destroy()  {
59
60     }
61
```

Figura 5. Exemplo de Aplicação Não Visual

De acordo com sua documentação [2], a função do método é: “Obter o início da String do Locator, não incluindo o primeiro dois pontos”. Dessa forma, o teste poderá ser feito com a criação de um objeto MediaLocator e, posteriormente, verificando se o protocolo retornado é o mesmo protocolo passado como argumento.

Feito a verificação, resta sinalizar, ao controlador do framework, o evento do resultado obtido (linha 48 e 50, Figura 5).

3.2. Aplicações Visuais

Para testar esse tipo de aplicação, três passos são necessários:

1 – Implementar os métodos da Interface plaver.dtvtest.DTVTest;

2 – Criar um atributo do tipo Form e atribui o objeto “form” (Figura 6, linha 12);

3 – Criar um atributo do tipo DTVTestLog e atribuir o objeto “log” (Figura 6, linha 15).

```
1  import com.sun.dtv.lwuit.Form;
5
6  public class APPVisual implements DTVTest{
7
8      private Form       form;
9      private DTVTestLog dtvTestLog;
10
11     public void setForm(Form form) {
12         this.form = form;
13     }
14
15     public void setLog(DTVTestLog log) {
16         this.dtvTestLog = log;
17     }
18
19     public void init() {
20
21     }
22
23     public void start() {
24
25     }
26
27     public void pause() {
28
29     }
30
31     public void destroy() {
32
33     }
34
35 }
```

Figura 6. Modelo de Aplicação Visual

Similar às aplicações não visuais, os testes serão realizados de acordo com a funcionalidade dos métodos a serem testados. Para realizar os testes, será necessária uma aplicação de processamento de imagens que reconhecerá a aplicação na tela e irá comparar com o resultado esperado de acordo com sua especificação.

Há vários exemplos de aplicações visuais na classe com.sun.dtv.lwuit.Graphics. Por exemplo, para testar um método que desenha um quadrado na tela, basta verificar se há quatro retas de dimensões iguais sendo apresentadas. O código da Figura 7 testa o método drawLine() da classe Graphics.

De acordo com a especificação JavaDTV 1.3 [1], a função do método drawLine(...) é: “Desenhar uma linha entre duas coordenadas X/Y”. Dessa forma, o teste poderá ser feito através do reconhecimento de reta em uma região específica na tela.

Feito a verificação, resta sinalizar o evento do resultado obtido ao controlador do framework (linha 60, Figura 7). Para aplicações visuais, só há a possibilidade da chamada do método dtvTestLog.appVisualFinished().

```java
package plaver.com.sun.dtv.lwuit.Graphics;

import java.awt.Color;

public class DrawLineTest implements DTVTest {

    private DTVTestLog dtvTestLog;
    private Form form;

    private Label labelImg;

    public void setForm(Form form) {
        this.form = form;
    }

    public void setLog(DTVTestLog log) {
        this.dtvTestLog = log;
    }

    public void init() {

    }

    public void start() {

        Dimension dimImg;
        Dimension dimLabel;
        Graphics grap;

        dimImg   = new Dimension(form.getWidth()/100 * 30,form.getHeight()/100 * 30);
        dimLabel = new Dimension((int) (dimImg.getWidth()*1.3), (int) (dimImg.getHeight()*1.3));

        Image img = Image.createImage(dimImg.getWidth(), dimImg.getHeight());

        grap = img.getGraphics();
        grap.setColor(Color.BLACK);

        grap.drawLine(0, 0, 100, 100);

        labelImg = new Label();
        labelImg.getStyle().setBgColor(Color.WHITE);
        labelImg.setSize(dimLabel);
        labelImg.setX((int)((form.getWidth()  - dimImg.getWidth())/1.8));
        labelImg.setY((int)((form.getHeight() - dimImg.getHeight())/1.2));

        labelImg.setIcon(img);

        form.addComponent(labelImg);
        form.repaint();

        dtvTestLog.visualAppFinished();

    }

    public void pause() {

    }

    public void destroy() {
        form.removeComponent(labelImg);
    }

}
```

Figura 7. Exemplo de Aplicação Visual

4. Referências

[1] **JavaDTV 1.3**, [online] Disponível na internet via WWW.URL: http://forumsbtvd.org.br/acervo-online/javadtv-download/. Acessado em 31 de janeiro de 2013.

[2] **MediaLocator**, [online] Disponível na internet via WWW.URL: http://docs.oracle.com/cd/E17802_01/j2se/javase/technologies/desktop/media/jmf/2.1.1/apidocs/javax/media/MediaLocator.html. Acessado em 31 de janeiro de 2013.