NORMA
BRASILEIRA

ABNT NBR 15606-4

Primeira edição
13.04.2010

Válida a partir de
13.05.2010

Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital
Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais

Digital terrestrial television – Data coding and transmission specification for digital broadcasting
Part 4: Ginga-J - The environment for the execution of procedural applications

ICS 33.080; 33.160.01

ISBN 978-85-07-02022-6

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS

Número de referência
ABNT NBR 15606-4:2010
91 páginas

ABNT
Av.Treze de Maio, 13 - 28º andar
20031-901 - Rio de Janeiro - RJ
Tel.: + 55 21 3974-2300
Fax: + 55 21 3974-2346
abnt@abnt.org.br
www.abnt.org.br

Sumário

Página

Prefácio

A Associação Brasileira de Normas Técnicas (ABNT) é o Foro Nacional de Normalização. As Normas Brasileiras, cujo conteúdo é de responsabilidade dos Comitês Brasileiros (ABNT/CB), dos Organismos de Normalização Setorial (ABNT/ONS) e das Comissões de Estudo Especiais (ABNT/CEE), são elaboradas por Comissões de Estudo (CE), formadas por representantes dos setores envolvidos, delas fazendo parte: produtores, consumidores e neutros (universidade, laboratório e outros).

Os Documentos Técnicos ABNT são elaborados conforme as regras das Diretivas ABNT, Parte 2.

A Associação Brasileira de Normas Técnicas (ABNT) chama atenção para a possibilidade de que alguns dos elementos deste documento podem ser objeto de direito de patente. A ABNT não deve ser considerada responsável pela identificação de quaisquer direitos de patentes.

A ABNT NBR 15606-4 foi elaborada na Comissão de Estudo Especial de Televisão Digital (ABNT/CEE-85). O seu 1º Projeto circulou em Consulta Nacional conforme Edital nº 09, de 06.09.2007 a 05.11.2007, com o número de Projeto 00:001.85-006/4. O seu 2º Projeto circulou em Consulta Nacional conforme Edital nº 05, de 19.05.2009 a 17.07.2009, com o número de 2º Projeto 00:001.85-006/4. O seu 3º Projeto circulou em Consulta Nacional conforme Edital nº 03, de 05.03.2010 a 05.04.2010, com o número de 3º Projeto 85:000.00-006/4.

Esta Norma é baseada nos trabalhos do Fórum do Sistema Brasileiro de Televisão Digital Terrestre, conforme estabelecido no Decreto Presidencial nº 5.820, de 29.06.2006.

O Escopo desta Norma Brasileira em inglês é o seguinte:

Scope

This part of ABNT NBR 15606 specifies the requirements for the procedural part of the middleware for the Brazilian digital terrestrial televison system (SBTVD).

Introdução

A especificação Ginga-J é composta por um conjunto de Interfaces de Programação de Aplicativos (API - *Application Programming Interface*) projetadas para suprir todas as funcionalidades necessárias para a implementação de aplicativos para televisão digital, desde a manipulação de dados multimídia até protocolos de acesso.

A especificação Ginga se aplica aos receptores para sistemas de transmissão terrestre de televisão (*over-the-air*). Ginga é destinado a cobrir uma série completa de implementações, incluindo os receptores-decodificadores integrados (IRD), aparelhos de televisão integrados, computadores multimídia e *clusters* locais de aparelhos conectados via redes domésticas (*Home Area Networks* - HAN).

Esta parte da ABNT NBR 15606 é destinada aos desenvolvedores de receptores compatíveis com o sistema brasileiro de televisão digital terrestre (SBTVD) e aos desenvolvedores de aplicativos que utilizam a funcionalidade e API Ginga.

Esta parte da ABNT NBR 15606 tem como objetivo garantir a interoperabilidade dos aplicativos Ginga e diferentes implementações Ginga.

Esta parte da ABNT NBR 15606 é harmonizada com especificações internacionais, conforme detalhado no Anexo A.

Televisão digital terrestre — Codificação de dados e especificações de transmissão para radiodifusão digital
Parte 4: Ginga-J - Ambiente para a execução de aplicações procedurais

1 Escopo

Esta parte da ABNT NBR 15606 especifica os requisitos para a parte procedural do *middleware* para o sistema brasileiro de televisão digital terrestre (SBTVD).

2 Referências normativas

Os documentos relacionados a seguir são indispensáveis à aplicação deste documento. Para referências datadas, aplicam-se somente as edições citadas. Para referências não datadas, aplicam-se as edições mais recentes do referido documento (incluindo emendas).

ABNT NBR 15601:2007, *Televisão digital terrestre – Sistema de transmissão*

ABNT NBR 15603:2007 (todas as partes), *Televisão digital terrestre – Multiplexação e serviços de informação (SI)*

ABNT NBR 15604:2007, *Televisão digital terrestre – Receptores*

ABNT NBR 15606-1*, Televisão digital terrestre - Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 1: Codificação de dados*

ABNT NBR 15606-2:2007, *Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital - Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações*

ABNT NBR 15606-3:2007, *Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 3: Especificação de transmissão de dados*

ISO 639-2, *Codes for the representation of names of languages – Part 2: alpha-3 code*

ISO/IEC 8859-1:1998, *Information technology - 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet N° 1*

ISO/IEC 13818-1, *Information technology – Generic coding of moving pictures and associated audio information: Systems*

ARIB STD-B10:2008, *Service information for digital broadcasting system*

ARIB STD-B23:2004, *Application execution engine platform for digital broadcasting*

ARIB STD-B31:2007, *Transmission Coding Standard*

CDC 1.1:2008, *Connected Device Configuration 1.1 (JSR 218), disponível em http://jcp.org/en/jsr/detail?id=218*

FP 1.1:2008*, Foundation Profile 1.1 (JSR 219), disponível em http://jcp.org/en/jsr/detail?id=219*

JAR:2009, Sun Microsystems. *JAR File Specification.2009, disponível em http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html*

LWUIT 1.1:2008 , *LightWeight User Interface Toolkit, Sun Microsystems*

JAVADTV 1.3:2009, *Java DTV Specification, Sun Microsystems*

JAVATV 1.1:2008, *Java TV Specification 1.1 (JSR 927), Sun Microsystems disponível em: http://jcp.org/en/jsr/detail?id=927*

JCE:1.0.1:2006, *Sun Microsystem. Security (JCE - Java Cryptography Extension) Optional Package Specification v1.0.1, disponível em: http://jcp.org/en/jsr/detail?id=219*

JSSE 1.0.1:2006, *Sun Microsystem. Security (JSSE - Java Secure Socket Extension) Optional Package Specification v1.0.1, disponível: em http://jcp.org/en/jsr/detail?id=219*

JVM:1997, *Java(TM) Virtual Machine Specification, The (2nd Edition), T Lindholm, F Yellin – 1997 – Addison-Wesley*

PBP 1.1:2008, *Personal Basis Profile 1.1 (JSR 217), disponível em http://jcp.org/en/jsr/detail?id=217*

SATSA:1.0.1:2007*, Sun Microsystems , Security and Trust Services API for J2ME (JSR 177), disponível em: http://jcp.org/en/jsr/detail?id=177*

3 Termos e definições

Para os efeitos desta parte da ABNT NBR 15606, aplicam-se os seguintes termos e definições.

3.1
bytecode
forma intermediária de código interpretada pela JVM

3.2
contexto de serviço
ambiente no qual o serviço é apresentado no receptor digital

3.3
máquina virtual Java
Java virtual machine
JVM
processo que carrega e executa os aplicativos Java

3.4
serviço
conjunto de informações que contém áudio, vídeo e/ou dados, para apresentação em um receptor digital

NOTA Normalmente o serviço é referenciado pelos telespectadores como “canal de televisão”.

3.5
zapper
aplicação residente, tipicamente desenvolvida pelo fabricante do receptor, a qual o usuário pode ativar a qualquer momento

NOTA O *zapper* pode ser usado para selecionar serviços e aplicações para posterior execução.

4 Abreviaturas

Para os efeitos desta parte da ABNT NBR 15606, aplicam-se as seguintes abreviaturas.

API	*Application Programming Interface*
CA	*Conditional Access*
CDC	*Connected Device Configuration*
CSS	*Cascading Style Sheets*
ECMA	*European Computer Manufacturers Association*
EDT	*Event*
EPG	*Electronic Program Guide*
HAN	*Home Area Network*
IRD	*Integrated Receiver Decoder*
JPEG	*Joint Photographic Expert Group*
MIDP	*Móbile Information Devide Profile*
MPEG	*Moving Picture Expert Group*
NCL	*Nested Context Language*
PBP	*Personal Basis Profile*
PNG	*Portable Network Graphics*
SBTVD	*Sistema Brasileiro de Televisão Digital Terrestre*
TOT	*Time Offset Table*
TS	*Transport Stream*
UTF	*Unicode Transformation Format*
XHTML	*eXtensible Hypertext Markup Language*

5 Arquitetura do *middleware* Ginga

5.1 Visão geral da arquitetura Ginga

O universo das aplicações para televisão digital pode ser particionado em dois conjuntos: o das aplicações declarativas e o das aplicações procedurais. Uma aplicação declarativa é aquela em que sua entidade “inicial” é do tipo “conteúdo declarativo”. Analogamente, uma aplicação procedural é aquela em que sua entidade “inicial” é do tipo “conteúdo procedural”.

Um conteúdo declarativo deve ser baseado em uma linguagem declarativa, isto é, em uma linguagem que enfatiza a descrição declarativa do problema, ao invés da sua decomposição em uma implementação algorítmica. Um conteúdo procedural deve ser baseado em uma linguagem não declarativa. Linguagens não declarativas podem seguir diferentes paradigmas. Têm-se, assim, as linguagens baseadas em módulos, orientadas a objetos etc. A literatura sobre televisão digital, no entanto, utiliza o termo procedural para representar todas as linguagens que não são declarativas. Em uma programação procedural, o computador deve obrigatoriamente ser informado sobre cada passo a ser executado. Pode-se afirmar que, em linguagens procedurais, o programador possui maior poder de expressão em seu código, sendo capaz de estabelecer todo o fluxo de controle e execução de seu programa - como existem mais recursos disponíveis, o grau de complexidade é maior. A linguagem mais usual encontrada nos ambientes procedurais de um sistema de televisão digital é Java. O Ginga-NCL (ou Máquina de Apresentação) é um subsistema lógico do Sistema Ginga que processa documentos NCL. Um componente-chave do Ginga-NCL é o mecanismo de decodificação do conteúdo informativo (NCL *formatter*). Outros módulos importantes são o usuário baseado em XHTML, que inclui uma linguagem de estilo (CSS) e intérprete ECMAScript, e o mecanismo LUA, que é responsável pela interpretação dos *scripts* LUA.

O Ginga-J (ou Máquina de Execução) é um subsistema lógico do Sistema Ginga que processa aplicações procedurais (Xlets Java). Um componente-chave do ambiente do aplicativo procedural é o mecanismo de execução do conteúdo procedural, que tem por base uma máquina virtual Java.

É importante observar que em uma implementação unicamente Ginga-NCL ou unicamente Ginga-J, seja em receptores fixos ou móveis, é proibida a reivindicação de qualquer tipo de conformidade com o SBTVD. Isso garante que o Ginga ofereça perfis sempre compatíveis com versões anteriores.

Decodificadores comuns de conteúdo devem servir para as necessidades de aplicativos tanto procedurais quanto informativos de decodificação e apresentação de conteúdos comuns do tipo PNG, JPEG, MPEG e outros formatos. O Ginga-Core é composto por decodificadores e procedimentos comuns de conteúdo para obter conteúdos transportados em fluxos de transporte MPEG-2 (TS) e através de um canal de retorno. O Ginga-Core também deve suportar o modelo de exibição conceitual descrito na ABNT NBR 15606-1.

A arquitetura (ver Figura 1) e as facilidades da especificação Ginga devem ser destinadas à aplicação em sistemas e receptores de transmissão para transmissão terrestre (*over-the-air*). Além disso, a mesma arquitetura e facilidades podem ser aplicadas a outros sistemas de transporte (como sistemas de televisão via satélite ou cabo).

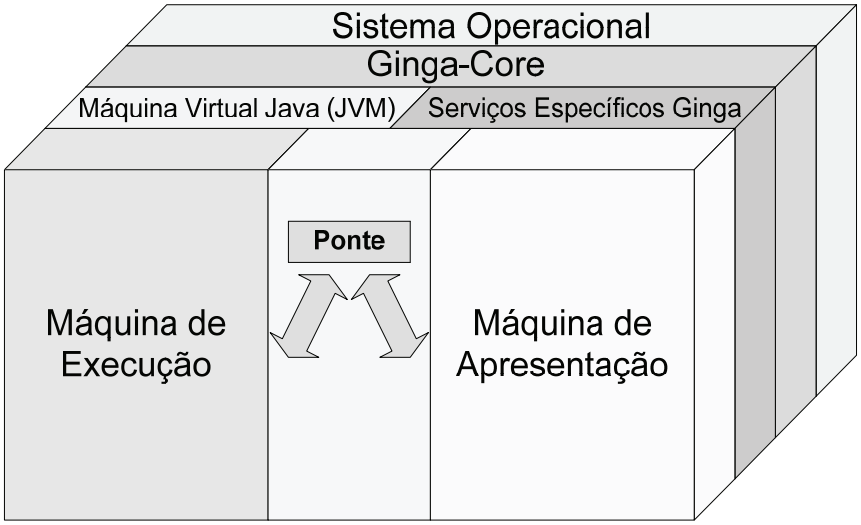

Figura 1 — Arquitetura em alto nível do *middleware* Ginga

5.2 Arquitetura Ginga-J

5.2.1 Contexto

A Figura 2 apresenta o contexto em que a pilha do *software* Ginga-J é executada. O Ginga-J é uma especificação de *middleware* distribuído, que reside em um dispositivo Ginga (dispositivo que embarque o *middleware* Ginga – um receptor de televisão digital).

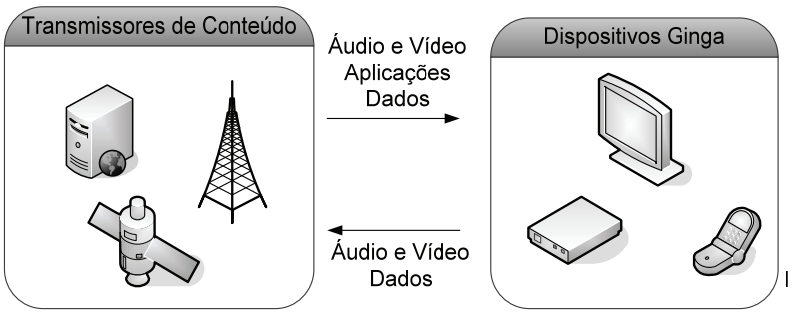

Figura 2 — Contexto do Ginga-J

O dispositivo Ginga deve ter acesso a fluxos de vídeo, áudio, dados e outros recursos de mídia, que devem ser transmitidos através do ar, cabo, satélite ou através de redes IP. As informações recebidas devem ser processadas e apresentadas aos telespectadores.

5.2.2 Arquitetura

O modelo Ginga-J distingue entre as entidades e recursos de *hardware*, *software* do sistema e aplicativos conforme descrito na Figura 3.

As aplicações residentes podem ser implementadas usando funções não padronizadas, fornecidas pelo Sistema Operacional do dispositivo de Ginga, ou por uma implementação particular do Ginga. Os aplicativos residentes também podem incorporar funcionalidades providas pelas API padronizadas Ginga-J. Aplicativos transmitidos (*Xlets*) sempre devem utilizar API padronizadas fornecidas pelo Ginga-J.

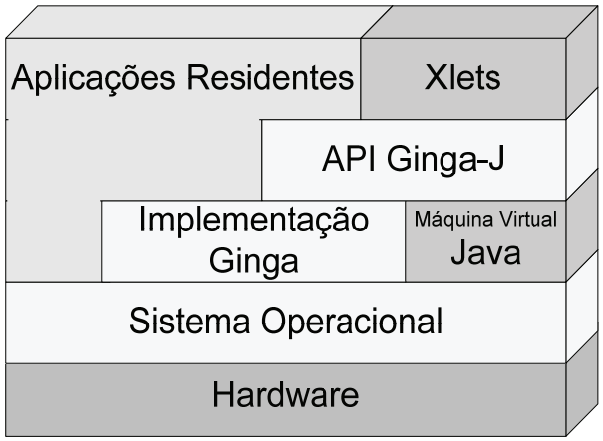

Figura 3 — Arquitetura Ginga-J e ambiente de execução

Em geral, o Ginga é alheio a quaisquer aplicativos residentes. Estas aplicações residentes incluem, mas não limitam-se a: *closed caption*, mensagens do sistema de acesso condicional (*Conditional* Access – CA), menus do receptor e guias de programação eletrônica (*Eletronic Program Guide* – EPG) residente.

Os aplicativos residentes podem ter prioridade sobre os aplicativos Ginga. Como exemplo, o *closed caption* e mensagem de emergência devem ter prioridade no Sistema Ginga.

6 Formato do conteúdo

O formato do conteúdo para o sistema brasileiro de televisão digital terrestre deve estar de acordo com a ABNT NBR 15606-1.

7 Modelo de aplicação Ginga-J

7.1 Modelo de aplicação

7.1.1 Ciclo de vida

O modelo de aplicação Ginga-J deve estar de acordo com o modelo de aplicação definido em JAVADTV 1.3:2009. Desta forma, aplicações Ginga-J devem conter uma classe implementando a interface *javax.microedition.xlet.Xlet* (ver PBP 1.1:2008), que deve ser referenciada de acordo com as definições de sinalização de aplicação (ver ABNT NBR 15606-3:2007, 12.16). Do contrário, a classe (e a instância da aplicação) pode ser ignorada.

Aplicações Ginga-J devem ser executadas em um ambiente orientado a serviços e mantidas por um gerenciador de aplicações global do sistema. Todo serviço é apresentado em um contexto de serviço, que poderia ser definido como seu ambiente de execução. Para aplicações Ginga-J, o contexto de serviço é representado por uma instância da classe *javax.microedition.xlet.ServiceContext*.

Além disso, aplicações Ginga-J podem ser controladas tanto por um *zapper*, pela emissora (ver ABNT NBR 15606-3:2007, 12.16), por outra aplicação Ginga-J usando a API *"Application Management and LifeCycle Control"* (ver JAVADTV 1.3:2009) ou ainda por documentos Ginga-NCL. Considerando que a aplicação providencia uma classe que implementa a interface *javax.microedition.xlet.Xlet*, esta classe contém ao menos quatro métodos que são chamados pela plataforma para informar à aplicação de mudanças de ciclo de vida iminentes. Um diagrama exibindo o ciclo de vida de aplicações Ginga-J é mostrado na Figura 4.

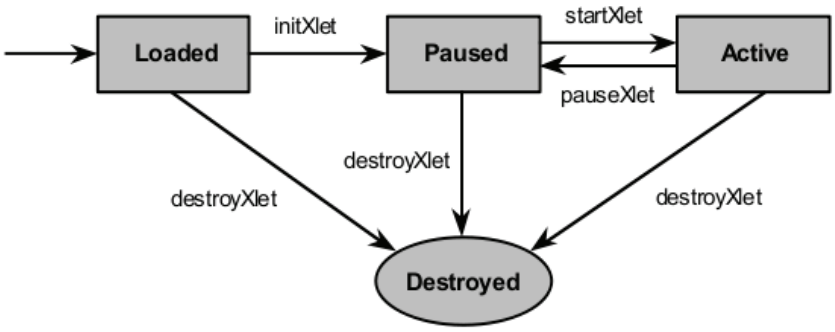

Figura 4 — Diagrama com estados do ciclo de vida de um Xlet

Inicialmente, após os dados da aplicação serem obtidos, o objeto que implementa javax.microedition.xlet.Xlet é criado usando seu construtor. Se o construtor padrão retornar sem disparar uma exceção, a instância da aplicação é considerada como no estado "Loaded", senão a instância da aplicação é considerada como no estado "Destroyed" e descartada.

NOTA A inicialização dos recursos utilizados pela aplicação é feita no método *initXlet()* e não no construtor da classe. A chamada ao construtor é dependente de implementação.

Para inicializar a aplicação do método *initXlet*, é chamado um objeto instância de *javax.microedition.xlet.XletContext*, que possui informação do contexto de execução para a aplicação, incluindo propriedades e mecanismos para notificação de mudanças de estados iniciadas pela aplicação. Assim que a instância da aplicação tiver sido carregada e instanciada com sucesso, o gerente de aplicações pode mudar o estado da instância de aplicação para "Paused".

NOTA É possível que o método *initXlet* seja chamado assincronamente.

O método *startXlet* pode então ser chamado para informar à aplicação que ela é convertida para o estado "Active", iniciando sua execução.

O método *pauseXlet* pode ser chamado para informar à aplicação que ela deve mover-se para o estado "Paused" e que ela deve minimizar seu consumo de recursos. A aplicação pode mover-se de volta para o estado "Active" após uma nova chamada ao método *startXlet*. Uma instância de aplicação no estado "Paused" deve reduzir seu consumo de recursos se ela tiver a intenção de maximizar sua probabilidade de sobrevivência. Isto não significa que ela não pode manter quaisquer recursos, mas que, caso mantenha, ela deve ter uma prioridade menor ao acesso de recursos do que se estivesse no estado "Active".

O método *destroyXlet* pode ser chamado em qualquer estado e é usado para avisar à aplicação que ela está prestes a ter sua execução terminada. A aplicação deve salvar informação de seu estado (se possível e necessário) e liberar recursos previamente utilizados o mais rápido possível. Este método possui um parâmetro booleano que indica se é incondicional que esta aplicação deve ser parada. Caso a aplicação esteja sendo parada devido a uma operação de seleção de serviço, a finalização da aplicação é incondicional (ver 7.1.3).

NOTA Uma instância de aplicação pode entrar neste estado apenas uma vez.

Se um método na interface Xlet disparar *javax.microedition.xlet.XletStateChangeException* (ver PBP 1.1:2008), por padrão, o Xlet permanece no estado em que estava imediatamente antes da chamada que disparou a exceção. A única exceção a esta regra é o método *destroyXlet*, quando o parâmetro booleano incondicional for passado com valor verdadeiro. Nesse cenário, disparar, *XletStateChangeException* não pode produzir nenhum efeito e o Xlet deve ser destruído.

O método *initXlet* deve ser chamado apenas uma vez. Além disso, o gerente de aplicações pode escolher mover o Xlet para o estado "Destroyed" (sem chamar *destroyXlet*) algum tempo específico da implementação depois, caso seja disparada uma *XletStateChangeException.*

Aplicações no estado "Destroyed" não podem ser iniciadas usando a API "*Application Management and LifeCycle Control* " (ver JAVADTV 1.3:2009) ou qualquer outro mecanismo disponível na plataforma.

7.1.2 Inicialização de aplicações

Quando um novo serviço é selecionado para exibição, o gerenciador de aplicações global do sistema deve verificar as aplicações disponíveis de acordo com a ABNT NBR 15606-3:2007, 12.16. Em particular, este gerenciador também identifica aplicações que devem ser iniciadas imediatamente após a seleção do serviço em questão.

Se já existir uma aplicação em execução no momento em que ocorre a seleção do serviço que a contém, esta pode continuar em execução, caso seja sinalizada como uma aplicação válida para o novo serviço selecionado.

Após a seleção de um serviço é possível que aplicações alternativas sinalizadas sejam iniciadas pelo usuário, pelo *zapper* ou por quaisquer outras aplicações usando a API *"Application Management and LifeCycle Control"* (ver JAVADTV 1.3:2009). Isto é, o usuário pode iniciar uma aplicação após receber uma oferta de aplicações através de alguma interface de usuário. Já que tal interface é dependente de implementação, serviços sinalizados devem indicar explicitamente, caso necessitem que a aplicação seja iniciada automaticamente (ver ABNT NBR 15606-3:2007, 12.16).

7.1.3 Finalização de aplicações

Aplicações Ginga-J podem voluntariamente terminar sua execução usando a API Xlet (ver PBP 1.1:2008) ou podem ser finalizadas pelo gerenciador de aplicações global do sistema.

Além disso, uma aplicação deve ter sua execução interrompida incondicionalmente sempre que uma das seguintes condições acontecerem:

— a tabela AIT (ver ABNT NBR 15603:2007) foi atualizada ou foi trocada e nesta nova versão não consta referência para a aplicação em questão;

— a tabela AIT (ver ABNT NBR 15603:2007) não é mais referenciada na tabela PMT (ver ABNT NBR 15603:2007) do serviço sendo exibido.

Quando uma instância de aplicação é eleita para ter sua execução terminada, o gerenciador de aplicações deve chamar seu método *destroyXlet*. Conforme descrito em 7.1.1, caso esta instância esteja sendo parada devido a uma operação de seleção de serviço, sua finalização deve ser incondicional.

7.1.4 Suporte a múltiplas aplicações

Aplicações Ginga-J devem ser executadas em um ambiente multitarefa orientado a eventos de mídia sinalizados por difusão e entradas de eventos do usuário. O modelo de aplicação foi projetado para ser extensível. É possível suportar múltiplas aplicações concorrentes que estão cooperando (projetadas para se comunicar entre si e compartilhar recursos) ou não (independentes e competindo por recursos).

O modelo de suporte a múltiplas aplicações do Ginga-J deve estar de acordo com o modelo de aplicação definido em JAVADTV 1.3:2009. Desse modo, uma aplicação não pode ser iniciada caso uma instância desta aplicação já

esteja ativa no serviço selecionado para exibição. Para os casos em que mais de um Xlet está em execução, não são permitidas quaisquer ações que possam afetar o estado global da plataforma (ver 7.1.1).

Considera-se cada instância de aplicação como se estivesse executando em sua própria instância de máquina virtual. Não obstante, é responsabilidade da emissora garantir que aplicações executadas simultaneamente em um dado serviço sejam compreensíveis ao usuário e não causem problemas perceptíveis por interferência mútua.

7.1.5 Compartilhamento de recursos entre aplicações

Permitir a execução simultânea de múltiplas aplicações significa que algumas regras devem ser definidas para que essas aplicações compartilhem recursos disponíveis no sistema. Em particular, aplicações em execução devem compartilhar o "Input Focus" e o "Output Focus".

Uma aplicação possui o "Input Focus" se e somente se o *java.awt.Component* ou o *com.sun.dtv.lwuit.Component* possuindo "Input Focus" pertencer à árvore de componentes daquela aplicação. "Input Focus" pode ser requisitado por aplicações chamando o método *requestFocus* em uma das classes supracitadas, dependendo do tipo de componente gráfico utilizado.

A aplicação que possui o "Input Focus" é em princípio capaz de receber eventos de entrada de usuário. Outras aplicações que não possuem o "Input Focus" podem requisitar a recepção de um subconjunto dos eventos de entrada de usuário através da API "TV *Specific UI functionality Event*" (ver JAVADTV 1.3:2009).

7.1.6 Controlando aplicações

É possível controlar o ciclo de vida de uma aplicação através da API "*Application Management and LifeCycle Control"* (JAVADTV 1.3:2009), que provê meios para permitir que aplicações requisitem ao *Application Manager* iniciar, parar, pausar e retomar outras aplicações.

7.1.7 Comunicação entre aplicações

O modelo de comunicação entre aplicações do Ginga-J deve estar de acordo com o modelo de aplicação definido em JAVADTV 1.3:2009. Aplicações devem usar os mecanismos definidos na API “Inter-Xlet Communication” (ver PBP 1.1:2008) para tal. A comunicação entre uma aplicação e outra é estabelecida pela ligação entre um objeto a um nome no *javax.microedition.xlet.ixc.IxcRegistry* (ver PBP 1.1:2008) e outra aplicação buscando este nome e invocando os métodos do objeto. Os possíveis "namespaces" para registro devem estar de acordo com as definições estabelecidas pela interface de comunicação entre aplicações de JAVADTV 1.3:2009.

7.1.8 Propriedades do ambiente

Além das propriedades já definidas pelo modelo Ginga-NCL (ver ABNT NBR 15606-2:2007, 7.2.4), o modelo de aplicação Ginga-J deve prover cada aplicação com um *javax.microedition.xlet.XletContext* (ver PBP 1.1:2008) incluindo um conjunto de propriedades específicas já definidas em JAVADTV 1.3:2009, as quais são mostradas na Tabela 1.

Tabela 1 — Propriedades do ambiente Ginga-J

Nome	Tipo de dados	Descrição
com.sun.dtv.persistent.root	Alfanumérico	Diretório-base para armazenamento persistente na plataforma
com.sun.dtv.orgid	Numérico	Identificador único para a organização responsável pela aplicação. Deve ser o mesmo valor transmitido no campo *organization_id do* descritor de identificador de aplicações (ver *A*BNT NBR 15606-3:2007, 12.7)
com.sun.dtv.appid	Numérico	Identificador único para a aplicação. Deve ser o mesmo valor transmitido no campo application_*id do* descritor de identificador de aplicações (ver ABNT NBR 15606-3:2007, 12.7)
com.sun.dtv.version	Alfanumérico	O número de versão da especificação JavaDTV implementada pela plataforma (JAVADTV 1.3:2009)
br.org.ginga.system.version	Alfanumérico	O número de versão da especificação Ginga-J implementada pela plataforma

7.1.9 Códigos de controle de aplicação

O controle dinâmico do ciclo de vida de aplicações é sinalizado através do campo “application_control_code” para a aplicação na AIT (ver ABNT NBR 15606-3:2007). Os códigos de controle das aplicações Ginga-J são mostrados na Tabela 2.

Tabela 2 — Códigos de controle de aplicações Ginga-J

Código	Identificador	Descrição
0x00	Não definido	Reservado para uso futuro
0x01	AUTOSTART	Aplicativos com o código de controle AUTOSTART são iniciados automaticamente, logo após a seleção do serviço que os contém
0x02	PRESENT	Aplicativos com o código de controle PRESENT não podem ser iniciados automaticamente e devem ser adicionados à lista de aplicativos disponíveis do receptor. Podem ser inicializados usando a API "*Application Management and LifeCycle Control"* (JAVADTV 1.3:2009)
0x03	DESTROY	Aplicativos com o código de controle DESTROY devem ser incondicionalmente finalizados pelo gerenciador de aplicações. Aplicativos sinalizados previamente com código de controle STORE devem ser removidos do *cache*
0x04	KILL	Aplicativos com o código de controle KILL devem ser finalizados pelo gerenciador de aplicações logo que possível. Caso seja lançada uma exceção do tipo *javax.microedition.xlet.XletStateChangeException* durante uma tentativa de finalização, o aplicativo deve continuar em execução Aplicativos sinalizados previamente com código de controle STORE podem opcionalmente ser mantidos no *cache*
0x05	Não definido	Reservado para uso futuro
0x06	REMOTE	Aplicativos com código de controle REMOTE não tem seus arquivos transmitidos no *transport stream* corrente. A fonte de dados para estes aplicativos deve estar de acordo com o descritor de aplicação "*Transport Protocol Descriptor*" (ver *A*BNT NBR 15606-3:2007) Aplicativos com o código de controle REMOTE não podem ser iniciados automaticamente e devem ser adicionados à lista de aplicativos disponíveis do receptor. Podem ser inicializados usando a API "*Application Management and LifeCycle Control*" (ver JAVADTV 1.3:2009)
0x07	UNBOUND	Aplicativos com código de controle UNBOUND são similares a aplicativos sinalizados com PRESENT, exceto que o usuário decide se a aplicação pode ser armazenada para execução posterior. Caso o receptor não possua capacidade de armazenagem disponível para armazenar a aplicação ou o usuário escolha não instalar a aplicação, esta deve ser tratada como não disponível (não pode ser listada entre as aplicações disponíves). Receptores sem suporte a armazenamento de aplicações devem ignorar as aplicações sinalizadas com este código de controle
0x08	STORE	Aplicativos com código de controle STORE não podem ser iniciados automaticamente, mas indicam que técnicas de *caching* podem ser utilizadas de modo a acelerar o carregamento de seus recursos durante a inicialização (ver 7.2 para outras informações) Caso a plataforma não tenha capacidade para realizar técnicas de *caching* pró-ativo ou armazenamento de dados, aplicações sinalizadas como STORE devem ser tratadas de modo idêntico a aplicações sinalizadas com PRESENT
0x09…0xFF	Não definido	Reservado para uso futuro

Caso seja recebido um código de controle desconhecido, a aplicação deve permanecer no mesmo estado em que se encontra. Quando uma mudança no código de controle provocar uma mudança de estado em uma aplicação Ginga-J, um evento deve ser gerado para todas as aplicações Ginga-J que estiverem registradas para receber notificações de mudanças no ciclo de vida da aplicação em questão.

Opcionalmente, a plataforma pode prover um menu com uma listagem de aplicações sinalizadas com STORED, PRESENT e UNBOUND (aquelas que o usuário escolheu instalar) para que o usuário decida o momento certo de inicializá-las.

Com a exceção daquelas aplicações sinalizadas com o campo service_bound_flag inicializado com 0, todas as aplicações devem ser destruídas durante uma troca de serviço. A execução de aplicações sinalizadas com o campo service_bound_flag setado para 0 não é restrita a um serviço específico e não pode ser interrompida durante a seleção entre vários serviços (ver 7.3.4 para mais detalhes). Na mudança de serviço o receptor sempre pode interromper a execução destas aplicações, caso exista a necessidade de liberação de recursos.

7.2 Armazenamento e *caching* de aplicações

7.2.1 Modelos de armazenamento

O modelo de aplicação Ginga-J permite que aplicações sejam armazenadas e iniciadas a partir de memória persistente. Aplicações podem sofrer *caching* pró-ativamente ou em resposta a uma requisição de outra aplicação, sujeito a consentimento do usuário e limites de recursos da plataforma. Em ambos os casos o objetivo é melhorar a velocidade de carregamento da aplicação.

Assim, pode ser útil para aplicações onde for desejável ter uma exibição rápida, evitando a latência no início dela devido ao tempo de descarga. Estas aplicações podem ser relacionadas à difusão, caso em que o ciclo de vida da aplicação permanece controlado pela difusão da AIT, ou pode ser completamente auto-suficiente, caso em que a entrada da AIT para a aplicação é armazenada junto com os dados da aplicação.

Aplicações sinalizadas com o código de controle STORE ou UNBOUND na AIT devem adicionar informação extra no arquivo MANIFEST.MF, de modo a indicar quais arquivos devem ser armazenados. Para cada arquivo que deve ser armazenado, a entrada no MANIFEST.MF para este arquivo deve conter o valor *true* para o atributo “Persistent-Flag” e um valor para o atributo “File-Version” indicando a versão do arquivo. Receptores que suportem o armazenamento de aplicações devem verificar a versão do arquivo armazenado e atualizá-lo quando a aplicação recebida for maior. Além disso, devem ser adicionadas ao MANIFEST.MF entradas de arquivo para o *application_id* e o *organization_id*, ambos sinalizados na AIT para a aplicação em questão.

Aplicações armazenadas devem ser visíveis pela API "*Application Management and LifeCycle Control*" (JAVADTV 1.3:2009), da mesma forma como quaisquer outras aplicações.

7.2.2 Questões de armazenamento

O espaço disponível para o armazenamento das aplicações pode não ser suficiente para comportar todas as aplicações sinalizadas como persistentes (STORE ou UNBOUND), sendo assim necessário escolher quais delas devem ser efetivamente persistidas. Eventualmente, também pode ser necessário remover aplicações previamente armazenadas. Nestes casos, fica a critério da implementação do fabricante do receptor a política persistência e remoção de aplicações sinalizadas como STORE. As aplicações sinalizadas como UNBOUND devem ser instaladas e removidas com autorização explícita do usuário e devem ter prioridade em relação às aplicações sinalizadas como STORE.

Quando a emissora sinalizar uma nova versão de uma aplicação armazenada, o *middleware* pode sobrescrever a versão antiga da aplicação com uma nova versão. O momento em que isso acontece não é previsível. Se a versão antiga estiver em execução no momento da atualização, a plataforma pode decidir armazenar ambas as versões até que a cópia atualmente em execução termine. Neste momento, a cópia antiga é removida da armazenagem. Se a plataforma escolher apagar a cópia antiga antes que a aplicação termine, o comportamento da aplicação em execução deve permanecer o mesmo (na prática isto significa que todas as classes e recursos devem estar carregados na memória antes que a versão antiga seja apagada).

7.2.3 *Caching* pró-ativo

Quando uma aplicação é sinalizada com código de controle STORE, é permitido que a plataforma pró-ativamente armazene quaisquer arquivos que estejam indicados no arquivo de descrição de aplicação.

No entanto, não é obrigatório atender às requisições de prioridade caso sejam utilizadas técnicas de armazenamento pró-ativo de aplicações. Em particular, não é obrigatório que o *caching* proativo armazene todos os arquivos com prioridade crítica.

7.3 Transmissão de aplicações

7.3.1 Regras de sinalização

As regras-padrão de sinalização de aplicações Ginga-J devem obrigatoriamente estar de acordo com o descrito em 7.1.9 e com a ABNT NBR 15606-3:2007, 12.16.

7.3.2 Empacotamento de aplicações

Aplicações Ginga-J devem ser empacotadas, autenticadas e autorizadas de acordo com as definições especificadas em JAVADTV 1.3:2009. Cada aplicação Ginga-J pode conter um ou mais arquivos JAR (ver JAR:2009 e JVM:1997). O arquivo JAR principal contém os arquivos de classe da aplicação, arquivos de recurso e um manifesto que descreve a aplicação e seus requisitos. O mecanismo de assinatura de JAR permite que o JAR seja autenticado.

Se a transmissão for feita com uso do carrossel de objetos, não pode ser utilizado o empacotamento em um arquivo JAR, mas obrigatoriamente o sistema de arquivos transmitido deve estar organizado da mesma maneira.

Se a aplicação for transmitida pelo canal de interatividade, é obrigatório que ela seja transmitida em arquivos JAR.

7.3.3 Autenticação de aplicação

A autenticação de aplicações Ginga-J deve estar de acordo com a Norma Brasileira vigente no momento da transmissão da aplicação.

NOTA A Norma Brasileira sobre autenticação de aplicação está em elaboração.

7.3.4 Sinalizando a mesma aplicação em diversos serviços

Para que uma aplicação seja considerada sinalizada em diversos serviços, todas as seguintes condições devem ser atendidas:

- a sinalização deve estar presente em uma tabela AIT em todos os serviços;
- o identificador de aplicação deve ser igual em todos os serviços;
- o descritor de protocolo de transporte deve ser igual em todos os serviços ou a aplicação está sinalizada com o código de controle UNBOUND e já se encontra persistida no receptor.

Se, além das condições descritas acima a aplicação, também estiver sinalizada com o campo *service_bound_flag* com o valor 0, esta aplicação deve ser mantida em execução entre todas as trocas de serviço, a menos que haja alguma restrição de recursos no receptor.

7.3.5 *Download* de aplicações através do canal interativo

As regras de transporte de aplicação devem obrigatoriamente estar de acordo com a ABNT NBR 15606-3.

Aplicações obtidas através do canal de interatividade devem estar contidas em um arquivo JAR (ver 7.3.1). O suporte à compressão no arquivo JAR é opcional.

8 Plataforma Ginga-J

8.1 Plataforma Java

A plataforma Java utilizada para execução de aplicações Ginga-J é definida de acordo com PBP 1.1:2008.

Os *bytecodes* Java executados pela plataforma devem ter versões entre 45.3 e 47.

8.2 Considerações básicas da plataforma

8.2.1 Ambiente de execução

Cada aplicação Ginga-J deve ser processada em um ambiente de execução isolado, ou seja, deve haver uma entidade de sistema que represente uma JVM onde cada aplicação deve ser executada sem que haja interferência na execução de qualquer outra aplicação. Para tanto, este ambiente de execução deve permitir que cada aplicação seja executada por meio de um carregador (*ClassLoader*) próprio ou mesmo uma hierarquia própria de carregadores para acessar classes que não sejam parte da plataforma base do Ginga-J.

Aplicações concorrentes não podem compartilhar diretamente instâncias de objetos por estas definidas. Qualquer interação entre aplicações deve ser possível apenas por meio de API específica. Cada ambiente de execução deve ser estabelecido no momento em que a aplicação for iniciada e descarregada logo que a aplicação for destruída. Após a finalização de cada aplicação, o Gerenciador de Aplicações deve se assegurar de que finalizadores de classes contidas na aplicação sejam executados.

As aplicações Ginga podem ser executadas em modo concorrente. Uma vez carregada e iniciada uma instância de uma aplicação, é proibida a criação ou inicialização de outra instância da mesma aplicação. Aplicações são diferenciadas por meio de seus respectivos valores de organization_id e application_id (ver ABNT NBR 15606-3:2007).

Aplicações Ginga-J não podem sincronizar em classes de sistema ou mesmo outras instâncias estáticas de sistema. Caso contrário, o comportamento esperado é indefinido.

8.2.2 Hierarquia de pacotes e classes

Apenas métodos e campos (e suas respectivas dependências) das classes listadas nesta parte da ABNT NBR 15606 devem obrigatoriamente estar presentes em uma implementação Ginga. Além disso, onde houver dependência de um pacote específico, sua inclusão completa é permitida, mas não obrigatória. O comportamento para classes e métodos adicionais não é especificado para aplicações enviadas por radiodifusão.

As classes, interfaces e métodos listados ou referenciados por este documento e que estiverem marcados como obsoletos (*deprecated*) devem ter suas marcações sobrescritas de modo que tais classes, interfaces ou métodos se tornem obrigatórios nesta parte da ABNT NBR 15606. É fortemente recomendado que as aplicações Ginga-J não utilizem estes elementos depreciados, já que podem deixar de ser suportados em versões futuras desta parte da ABNT NBR 15606.

A inclusão de algum pacote por esta parte da ABNT NBR 15606 não implica diretamente a inclusão de seus subpacotes.

Aplicações Ginga-J não podem definir classes ou interfaces em qualquer pacote (ou *namespace*) definido nesta parte da ABNT NBR 15606.

Classes de implementação da plataforma Ginga-J não podem estar contidas no pacote vazio (*default package*).

8.2.3 Notificação de eventos

Para todas as classes listadas nesta parte da ABNT NBR 15606 onde haja métodos para registro/desregistro de notificações, sucessivas chamadas para registro de ouvintes (*listeners*) devem ter o mesmo efeito que uma só chamada. Logo, cada evento deve ser notificado apenas uma vez por ouvinte. Além disso, uma requisição de cancelamento de registro não pode surtir efeito caso o ouvinte em questão não esteja registrado.

O número de processos (*threads*) utilizados para notificação de eventos aos ouvintes é dependente de implementação. Aplicações Ginga-J não podem bloquear o processamento em seus ouvintes de modo a evitar que outros ouvintes não sejam notificados.

Todas as classes de eventos listadas nesta parte da ABNT NBR 15606 devem estender a classe *java.util.EventObject.*

8.2.4 Codificação de texto

A codificação de texto padrão para a plataforma Ginga-J deve ser UTF-8 conforme método *java.io.DataOutput.writeUTF* (ver PBP 1.1:2008). O padrão "Latin1" também deve ser suportado de acordo com ISO/IEC 8859-1:1998.

8.2.5 Ciclo de vida das aplicações

A máquina de estados definida em 7.1.1 deve funcionar de modo que o comportamento das aplicações respeite as seguintes restrições:

— a latência percebida durante a inicialização da aplicação deve ser a mínima possível;

— uma aplicação pode ser destruída a qualquer momento.

O Gerenciador de Aplicações deve usar a Xlet API (ver PBP 1.1:2008) para ordenar mudanças no ciclo de vida das aplicações. Logo, vários fatores podem estimular o Gerenciador de Aplicações, como, por exemplo:

— sinalizações originadas nas emissoras (ver 7.1.9);

— seleção a partir de menu proprietário com lista de aplicações;

— ordem originada em outra aplicação Ginga-J por meio da API " *Application Management and LifeCycle Control* " (JAVADTV 1.3:2009) (ver 7.1.6);

— ordem originada em documento NCL embarcando um ou mais Xlets.

A própria aplicação pode decidir mudar seu estado. Para tanto, deve usar sua instância de javax.microedition.xlet.XletContext para requisitar tal mudança ao gerenciador de aplicações.

8.3 Infra-estrutura comum

As API definidas em (ver CDC 1.1:2008; FP 1.1:2008 e PBP 1.1:2008) são incluídas por esta especificação como base para o funcionamento da plataforma Ginga-J.

Os pacotes específicos para televisão digital definidos em JAVATV 1.1:2008 também devem ser empregados nesta parte da ABNT NBR 15606. Complementarmente, as seguintes restrições devem ser respeitadas:

— o grupo de processos (*threads*) de uma aplicação Ginga-J não pode conter processos com prioridade maior que java.lang.Thread.NORM_PRIORITY;

— o retorno do método System.currentTimeMillis deve ser sincronizado com a data e hora transmitida na TOT;

— o retorno do método System.currentTimeMillis deve ter granularidade menor ou igual a 10 ms;

— o *TimeZone* utilizado pela JVM deve estar de acordo com as configurações do receptor.

— o *java.util.Calendar* deve ser sincronizado de acordo com a data e hora transmitida na TOT;

— o *java.util.Locale* padrão da JVM deve ser definido como “pt_BR”;

— as aplicações Ginga-J não podem utilizar o método java.util.TimeZone.setDefault. O comportamento deste método é dependente de implementação;

— os fluxos de saída System.out e System.err devem estar disponíveis para aplicações Ginga-J. No entanto, o fluxo de entrada System.in não pode estar disponível;

— os métodos Runtime.traceInstructions e Runtime.traceMethodCalls devem estar disponíveis sem que isto tenha impacto negativo na execução das aplicações e sem interferência no funcionamento de outras API;

— o método System.gc é dependente de implementação e não tem comportamento definido;

— o método Runtime.gc é dependente de implementação e não tem comportamento definido;

— a plataforma deve utilizar mecanismos previstos em java.lang.SecurityManager.checkPackageDefinition e java.lang.SecurityManager.checkPackageAccess para evitar uso indevido de classes de sistema por parte das aplicações;

— o pacote javax.tv.xlet é considerado obsoleto (*deprecated*) por esta especificação. No entanto, de modo a viabilizar uma maior integração com aplicações legadas, instâncias de javax.microedition.xlet.XletContext devem implementar também javax.tv.xlet.XletContext. Ambas as interfaces contêm métodos e comportamentos semelhantes. Deve ser evitado o uso da interface javax.tv.xlet.XletContext pelas aplicações Ginga-J;

— o pacote javax.tv.graphics é considerado obsoleto (*deprecated*) por esta especificação. As classes e interfaces definidas nele não podem ser utilizadas pelas aplicações Ginga-J;

— a granularidade mínima para a classe javax.tv.util.TVTimer deve ser menor ou igual a 10 ms;

— o menor intervalo de repetição para a classe javax.tv.util.TVTimer deve ser 40 ms;

— chamadas ao método javax.tv.util.TVTimer.scheduleTimerSpec devem provocar exceções do tipo TVTimerScheduleFailedException, caso não haja temporizadores (*timers*) disponíveis no sistema.

8.4 Apresentação gráfica e tratamento de eventos

8.4.1 LWUIT, LightWeight user interface toolkit

Esta parte da ABNT NBR 15606 utiliza JAVADTV 1.3:2009 para definir os componentes gráficos e o mecanismo de tratamento de eventos de usuário. As aplicações dispõem de funcionalidades gráficas como:

— componentes gráficos de alto nível;

— aplicação de temas customizáveis aos componentes gráficos;

— tratamento hierárquico através de contêineres e componentes;

— abstração de componentes nativos ao sistema.

8.4.2 Interface gráfica de usuário

A interface gráfica do usuário será feita utilizando os componentes específicos para aplicações de televisão e dos componentes gráficos providos pela LWUIT 1.1:2008 incorporada ao JAVADTV 1.3:2009. Os pacotes que definem estes componentes gráficos são:

- com.sun.dtv.ui – define os componentes gráficos especificamente ligados a televisão;
- com.sun.dtv.lwuit – contém os componentes gráficos que dão suporte a criação de interfaces gráficas de usuário;
- com.sun.dtv.lwuit.animations – habilita tanto componentes gráficos como transições animadas entre contêineres;
- com.sun.dtv.lwuit.geom – define os elementos geométricos básicos para desenho;
- com.sun.dtv.lwuit.layouts – define tipos úteis de layoults gráficos;
- com.sun.dtv.lwuit.list – define estruturas de lista customizáveis, utilizadas em componentes de outros pacotes, como o com.sun.dtv.lwuit;
- com.sun.dtv.lwuit.painter – permite desenhar arbitrariamente elementos gráficos a partir de imagens planas, escaladas e/ou tiled;
- com.sun.dtv.lwuit.plaf – permite customizar a aparência dos componentes gráficos;
- com.sun.dtv.lwuit.util – pacote de utilidades.

A interface com.sun.dtv.ui.MatteEnabled e as classes com.sun.dtv.ui.AnimatedMatte e com.sun.dtv.ui.*Static*Matte devem estar presentes e ser implementadas de forma a manter o contrato com as aplicações. Porém, a funcionalidade de composição gráfica utilizando as informações de transparência providas pelas instâncias destes objetos é opcional para as implementações da plataforma Ginga-J.

8.4.3 Tratamento dos planos da plataforma

8.4.3.1 Considerações gerais

O pacote com.sun.dtv.ui permite o acesso de forma genérica aos planos oferecidos pela plataforma, para exibição de conteúdo organizado em camadas na tela do dispositivo. Conforme ABNT NBR 15606-1, a organização dos planos ou camadas é apresentada de acordo com a Figura 5.

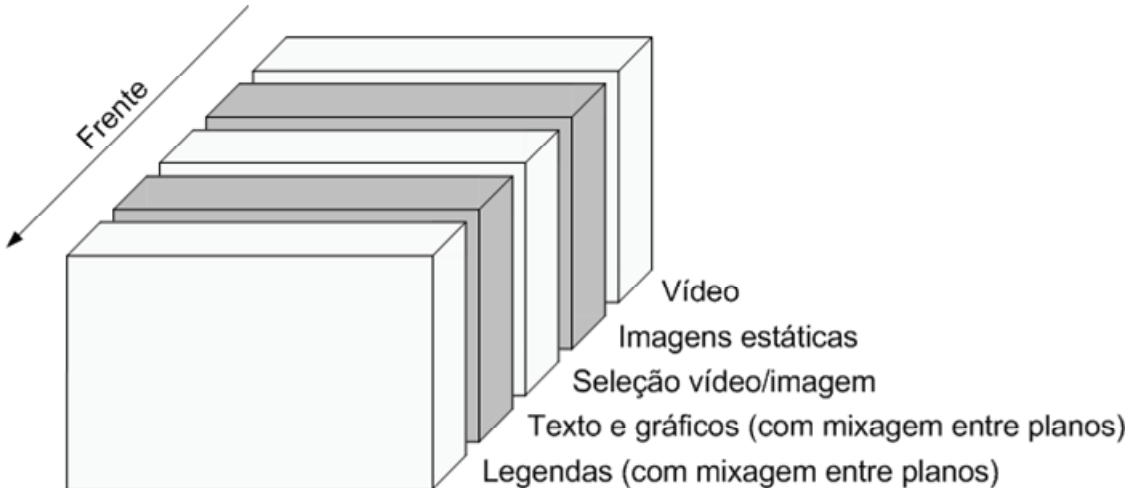

Figura 5 — Estrutura de camadas para a apresentação de serviços

O plano de legendas não é acessível por aplicações Ginga, sendo uma característica nativa do receptor. Restam, portanto, quatro planos sobre os quais uma aplicação Java pode operar. Seguindo esta convenção, estes planos são retornados sempre desta maneira, no método *getAllPlanes()* da classe com.sun.dtv.ui.*Screen* (que por sua vez é obtido através da classe com.sun.dtv.ui.Device):

— Plane[0]: Plano de texto e gráficos;

— Plane[1]: Plano de seleção vídeo/imagem;

— Plane[2]: Plano de imagens estáticas;

— Plane[3]: Plano de vídeo.

Para cada um destes planos, é possível obter suas características e efetuar operações gráficas sobre eles. Estas caracterísitcas, bem como as operações suportadas por cada plano, mantêm a aderência com as definições de ABNT NBR 15606-1. Este detalhamento é feito nas subseções abaixo, onde são definidos os valores de retorno para cada um dos métodos, dentro do objeto com.sun.dtv.ui.Capabilities correspondente ao respectivo plano.

8.4.3.2 Plano de texto e gráficos

O plano de texto e gráficos é o plano sobre o qual a aplicação pode desenhar elementos gráficos (primitivas geométricas e imagens) com alta definição de cores e canal de transparência, sobre o vídeo.

O retorno do método getID(): “GraphicPlane” está detalhado na Tabela 3.

Tabela 3 — Detalhamento de funcionalidades para o plano de texto e gráficos

Funcão	Descrição
getBitsPerPixel()	32
getColorCodingModel()	ColorCoding.ARGB8888
getSupportedPixelAspectRatios()	Um único objeto *Dimension*(1,1) é o valor recomendado (indica pixels com proporção 1:1). Qualquer valor diferente disto é considerado opcional
getSupportedPlaneAspectRatios()	Um único objeto *Dimension* com valor de construção (16,9) ou (4,3), dependendo da configuração da saída de vídeo do receptor
getSupportedScreenResolutions()	Um único objeto *Dimension* com as dimensões do plano de gráficos e textos no momento
isAlphaBlendingSupported()	*True*
isGIFRenderingSupported()	*True* apenas para plataformas que permitem exibição de imagens GIF no plano gráfico
isGraphicsRenderingSupported()	*True*
isImageRenderingSupported()	*True*
isJPEGRenderingSuppored()	*True*
isPNGRenderingSuppored()	*True*
isRealAlphaBlendingSupported()	*True*
isVideoRenderingSupported()	*True* apenas para plataformas que permitem exibição de monomídias de vídeo (ou mesmo um fluxo elementar de vídeo) no plano gráfico
isWidgetRenderingSupported()	*True*

O com.sun.dtv.ui.DTVContainer associado a este com.sun.dtv.ui.Plane deve suportar todos os tipos de com.sun.dtv.lwuit.component e operações gráficas definidas na API do LWUIT, exceto pela exibição de tipos de mídia não permitidos no plano de texto e gráficos (ver ABNT NBR 15606-1).

8.4.3.3 Plano de seleção vídeo/imagem

O plano de seleção vídeo/imagem permite definir áreas de precedência entre o plano de imagens estáticas e o plano de vídeo. Isto é, em que áreas retangulares da tela o plano de imagens estáticas será exibido sobre o plano de vídeo, ou vice-versa.

O retorno do método getID: “SwitchingPlane” está detalhado na Tabela 4.

Tabela 4 — Detalhamento de funcionalidades para o plano de vídeo e imagens

Função	Descrição
getBitsPerPixel()	1
getColorCodingModel()	ColorCoding.ONE_BPP
getSupportedPixelAspectRatios()	Idem a GraphicPlane
getSupportedPlaneAspectRatios()	Idem a GraphicPlane
getSupportedScreenResolutions()	Um único objeto *Dimension* com as dimensões do plano de vídeo no momento.
isAlphaBlendingSupported()	*True* (O plano de seleção vídeo/imagem é o modo pelo qual o plano de imagens estáticas implementa alpha blending)
isGIFRenderingSupported()	*False*
isGraphicsRenderingSupported()	*False*
isImageRenderingSupported()	*False*
isJPEGRenderingSuppored()	*False*
isPNGRenderingSuppored()	*False*
isRealAlphaBlendingSupported()	*False*
isVideoRenderingSupported()	*False*
isWidgetRenderingSupported()	*True*

O com.sun.dtv.ui.DTVContainer associado a este com.sun.dtv.ui.Plane suporta, além da definição de um layout manager (método setLayout()), apenas as chamadas addComponent() e removeComponent(), e somente quando o com.sun.dtv.lwuit.Component passado como parâmetro for do tipo br.org.sbtvd.ui.SwitchArea (ver Anexo E). Os demais métodos, embora não gerem erro na aplicação, não têm efeito, pois são referentes a operações não suportadas pelo plano de seleção vídeo/imagem.

Por meio do *com.sun.dtv.lwuit.plaf.Style* associado ao *com.sun.dtv.ui.DTVContainer* deste plano, pode ser definido o conteúdo de vídeo que será exibido por cima do Still Picture Plane ou vice-versa. As instâncias de com.sun.dtv.lwuit.plaf.Style associadas ao *com.sun.dtv.ui.DTVContainer* do plano de seleção vídeo/imagem só podem conter cores sólidas (java.awt.Color). A cor preta, java.awt.Color.BLACK, representa que o vídeo deve ser exibido por cima do Still Picture Plane. Com o uso de qualquer outra cor, o conteúdo do Still Picture Plane será exibido na frente do vídeo. Desta forma, as aplicações utilizam o modelo de cores do Java (RGB888 ou ARGB8888) para controlar o plano de selecção vídeo/imagem. A implementação desta API fará a conversão do modelo de cores do Java para o modelo de cores do plano de selecção vídeo/imagem de acordo com a seguinte função:

$$f(x)=\begin{cases} x = java.awt.Color.BLACK \Rightarrow \text{vídeo selecionado} \\ x \neq java.awt.Color.BLACK \Rightarrow \text{imagem estática selecionada} \end{cases}$$

8.4.3.4 Plano de imagens estáticas

Este é o plano sobre o qual a aplicação pode exibir imagens de alta resolução e profundidade de cores no formato JPEG. Tipicamente, ele é usado para definir um plano de fundo para a aplicação em telas onde o vídeo é redimensionado. Entretanto, através do uso conjunto com o plano de seleção vídeo/imagem, ele pode ser usado para exibir imagens JPEG em alta resolução sobre o vídeo.

O retorno do método getID: “StillPlane” está detalhado na Tabela 5.

Tabela 5 — Detalhamento de funcionalidades para o plano de imagens estáticas

Função	Descrição
getBitsPerPixel()	16
getColorCodingModel()	ColorCoding.YUV442
getSupportedPixelAspectRatios()	Idem a GraphicPlane
getSupportedPlaneAspectRatios()	Idem a GraphicPlane
getSupportedScreenResolutions()	Um único objeto *Dimension* com as dimensões do plano de Imagens Estáticas no momento
isAlphaBlendingSupported()	*True* (implementado através do plano de seleção vídeo/imagem)
isGIFRenderingSupported()	*False*
isGraphicsRenderingSupported()	*False*
isImageRenderingSupported()	*True*
isJPEGRenderingSuppored()	True
isPNGRenderingSuppored()	*False*
isRealAlphaBlendingSupported()	*False*
isVideoRenderingSupported()	*False*
isWidgetRenderingSupported()	True

O com.sun.dtv.ui.DTVContainer associado a este com.sun.dtv.ui.Plane suporta, além da definição de um layout manager (método setLayout()), apenas as chamadas addComponent() e removeComponent(), e somente quando o Component passado como parâmetro for do tipo br.org.sbtvd.ui.StillPicture (ver Anexo E). Os demais métodos, embora não gerem erro na aplicação, não têm efeito, pois são referentes a operações não suportadas pelo plano de imagens estáticas.

Por meio do *com.sun.dtv.lwuit.plaf.Style* associado à instância do objeto *com.sun.dtv.ui.DTVContainer* deste plano, pode ser definida uma cor de fundo sólida (sem transparência). Esta cor deve ser exibida como conteúdo do plano de imagens estáticas em todas as regiões deste plano que não são preenchidas por objetos do tipo br.org.sbtvd.ui.StillPicture. Esta cor deve ser definida no modelo de cores RGB888, próprio do ambiente Java, cabendo à plataforma a conversão para o modelo de cores YUV442.

8.4.3.5 Plano de vídeo

O plano de vídeo exibe o fluxo elementar de vídeo do serviço, ou opcionalmente, monomídias de vídeo. O conteúdo exibido neste plano pode ser manipulado através de controles JMF.

O retorno do método getID: “VideoPlane” está detalhado na Tabela 6.

Tabela 6 — Detalhamento de funcionalidades para o plano de vídeo

Função	Descrição
getBitsPerPixel()	Lança uma SetupException
getColorCodingModel()	Lança uma SetupException
getSupportedPixelAspectRatios()	Idem a GraphicPlane
getSupportedPlaneAspectRatios()	Idem a GraphicPlane
getSupportedScreenResolutions()	Um único objeto *Dimension* com as dimensões do plano de vídeo no momento
isAlphaBlendingSupported()	*False* (é o plano mais ao fundo da cadeia)
isGIFRenderingSupported()	*False*
isGraphicsRenderingSupported()	*False*
isImageRenderingSupported()	*False*
isJPEGRenderingSuppored()	*False*
isPNGRenderingSuppored()	*False*
isRealAlphaBlendingSupported()	*False*
isVideoRenderingSupported()	True
isWidgetRenderingSupported()	*False*

Não há objeto *com.sun.dtv.ui.DTVContainer* associado a este com.sun.dtv.ui.Plane. O efeito da invocação de qualquer outro método diferente de *getCapabilities()* nesta instância em particular é dependente da implementação.

8.4.3.6 Composição dos planos

A Figura 6 apresenta um exemplo de composição entre os diferentes planos. Na composição final pode-se observar como o conteúdo de vídeo é exibido na área indicada pela região preta no plano de seleção vídeo/imagem. Da mesma forma, os conteúdos do plano de imagens estáticas são exibidos na frente do vídeo na região “branca” do plano de seleção vídeo/imagem.

O conteúdo do plano de gráficos e textos é sempre exibido pela frente dos demais planos.

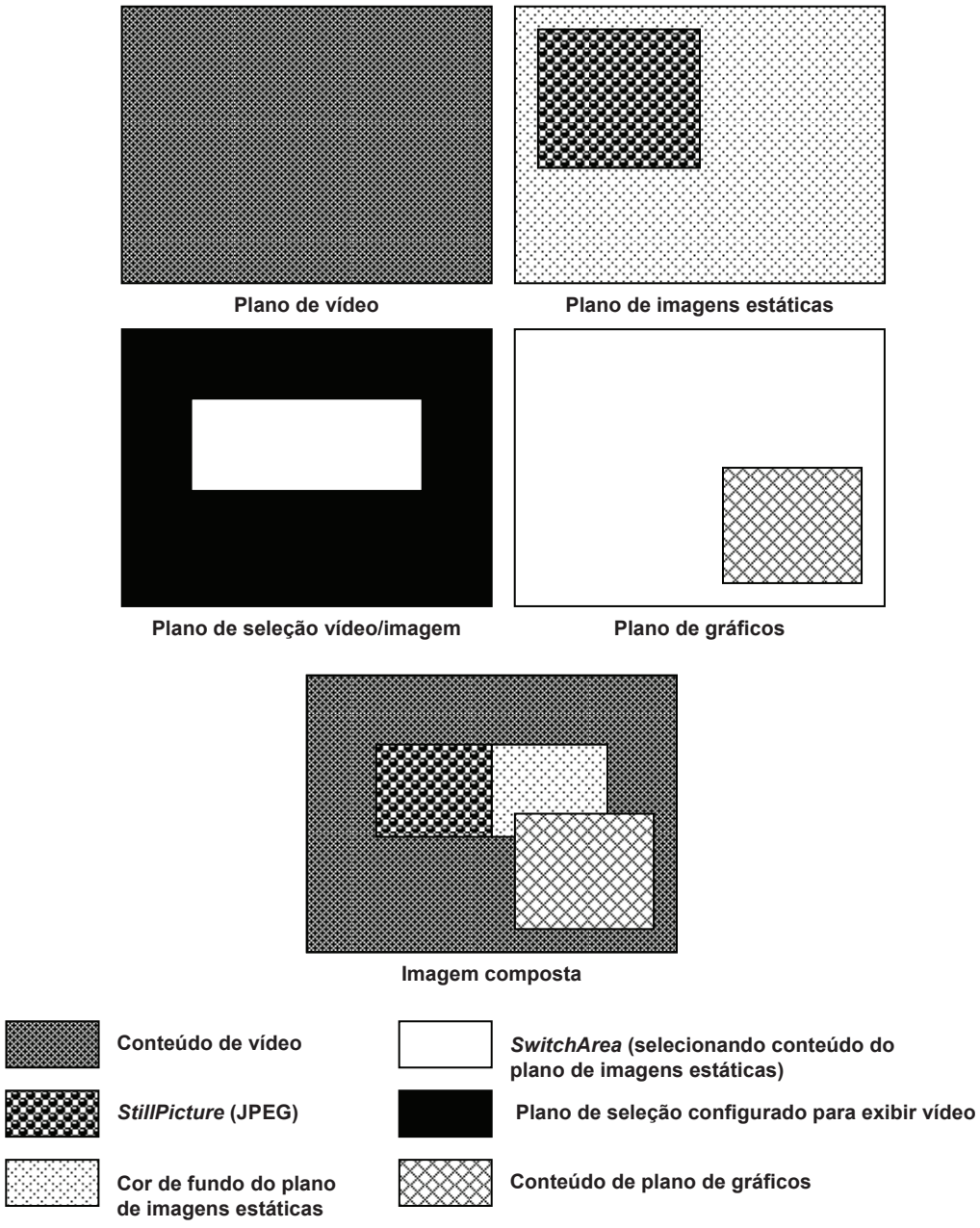

Figura 6 — Exemplo de composição dos planos de exibição de conteúdo

8.4.4 Tratamento de eventos do usuário

O mecanismo de tratamento de eventos do usuário é provido por componentes especializados em televisão e dos componentes providos pela LWUIT 1.1:2008 incorporada ao JAVADTV 1.3:2009. Os pacotes que definem estes componentes são:

- com.sun.dtv.ui.event – mecanismo para tratamento de eventos específicos de televisão;
- com.sun.dtv.lwuit.events – mecanismo de tratamento de eventos relacionados aos componentes gráficos definidos na LWUIT 1.1:2008 incorporada ao JAVADTV 1.3:2009.

Todo o mecanismo de tratamento de eventos do usuário é baseado no modelo EDT (*Event Dipatch Thread*), dando à plataforma toda a ciência dos eventos e repassando às aplicações o seu tratamento especializado.

A ABNT NBR 15604:2007, 7.2.28.3, define as funções mínimas do controle remoto para receptores que disponham de mecanismo de interatividade. Estas funções devem ser mapeadas para os tipos de eventos definidos na classe com.sun.dtv.lwuit.event.RemoteControlEvent da JAVADTV 1.3:2009. A Tabela 7 apresenta este mapeamento.

Tabela 7 — Mapeamento das funções da ABNT NBR 15604:2007 nos eventos definidos pela JAVADTV 1.3:2009

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

8.5 Informação e seleção de serviços

8.5.1 Considerações gerais

A API de informações de serviço desta especificação é responsável por fornecer aos aplicativos o acesso às informações presentes nas tabelas de informações de serviço MPEG. Tais informações devem incluir os fluxos de áudio e vídeo presentes em cada serviço multiplexado, incluindo também descrição textual dos serviços e eventos que o compõem, entre outros.

Esta API tem como base a definição das API da ARIB STD-B.23:2006, Anexo M, que é baseada no GEM e fornece funcionalidades de acesso à informação de serviços de acordo com o modelo de referência do padrão de televisão digital japonês ARIB STD-B10:2008, o mesmo que o SBTVD estende. Entretanto, uma vez que esta parte da ABNT NBR 15606 tem como núcleo a especificação JAVADTV 1.3:2009, foi necessário realizar alterações nos elementos dos pacotes *jp.or.arib.tv.si* e *jp.or.arib.tv.net* com o objetivo de integrar as funcionalidades de acesso a informações de serviço dependentes de protocolo. O Anexo B detalha essas funcionalidades.

8.5.2 Integração com API independente de protocolo

A API de informações de serviço independente de protocolo deve estar de acordo com a JAVADTV 1.3:2009, 6.4.

8.6 Apresentação e execução de mídias

A API de fluxos de mídia adota a Java Media Framework (JMF) 1.0, e opcionalmente pode incorporar API adicionais definidas na Java Media Framework (JMF) 2.1. A JMF 2.1 é retrocompatível com a JMF 1.0.

8.7 Acesso a dados

8.7.1 Considerações gerais

Deve ser utilizada a API *java.io* (ver PBP 1.1:2008) para acesso a objetos de dados de modo genérico. Classes e interfaces contidas deste pacote relacionadas a arquivos e sistemas de arquivos devem obedecer à seguinte restrição:

— o método *java.io.ObjectInputStream.readLine* é marcado como obsoleto (*deprecated*) por esta parte da ABNT NBR 15606. Logo, não pode ser usado por aplicações enviadas por difusão.

8.7.2 Acesso a arquivos

Para uma aplicação enviada por radiodifusão e sinalizada em um serviço específico, objetos de dados acessados por meio de caminhos relativos devem ser buscados a partir do caminho indicado no descritor de localização da aplicação "*ginga_j_application_location_descriptor*" (ver ABNT NBR 15606-3:2007, 12.18.2), no campo *base_directory*. O caminho definido no descritor em questão deve ser considerado como diretório base da aplicação.

8.7.3 Protocolo de transporte por radiodifusão

Quando a aplicação é transmitida via protocolo de radiodifusão, utilizando os protocolos de Carrossel de Objetos DSMCC ou Carrossel de Dados DSMCC, devem ser suportadas as extensões providas pela API "Broadcast file and streams handling" (ver JAVADTV 1.3:2009). Esta API provê acesso a arquivos e dados internos de *streams* como extensão ao conjunto de funcionalidades já disponíveis no pacote *java.io* (ver PBP 1.1:2008).

Esclarecimentos acerca de operações de entrada/saída para objetos enviados por difusão podem ser encontradas na documentação da classe *com.sun.broadcast.BroadcastFile* (ver JAVADTV 1.3:2009), bem como em JAVADTV 1.3:2009, 8.2.7.

Os esclarecimentos supracitados devem ser aplicados também a operações com objetos de dados do tipo *java.io.File* (ver PBP 1.1:2008).

8.7.4 Armazenamento persistente

Para fins de armazenamento persistente deve ser disponibilizado o pacote *java.io* (ver PBP 1.1:2008), bem como a API "Persistent storage access rights and properties" (ver JAVADTV 1.3:2009).

A propriedade "*com.sun.dtv.persistent.root*" deve ser acessível por meio do método *java.lang.System.getProperty* (ver PBP 1.1:2008) e deve identificar o diretório raiz para armazenamento persistente. Caminhos relativos não podem ser utilizados para acessar objetos em armazenamento persistente, sob pena de funcionamento indefinido em diversas plataformas. A estrutura de diretórios prevista para armazenamento persistente é descrita em detalhes em JAVADTV 1.3:2009, 6.5.

O acesso a arquivos ou diretórios acima do diretório-base da aplicação deve sempre resultar em uma exceção do tipo *java.lang.SecurityException* (ver PBP 1.1:2008) para aplicações Ginga-J.

Aplicações Ginga-J assinadas, autenticadas e com permissões de acesso a armazenamento persistente outorgadas devem ter concedidos os privilégios previstos em JAVADTV 1.3:2009, 6.5.

Para aplicações com permissões de acesso outorgadas, os diretórios e subdiretórios aos quais têm acesso devem ser criados automaticamente pela plataforma, caso não existam. É recomendado que diretórios e subdiretórios necessários sejam criados logo que as permissões sejam concedidas. O identificador do proprietário (*owner*) dos diretórios e subdiretórios criados automaticamente pela plataforma deve ter o mesmo valor do "application_id" (ver ABNT NBR 15606-3:2007, 12.7) da aplicação em questão.

Aplicações Ginga-J devem criar arquivos ou diretórios somente onde tiverem permissões de escrita.

Detalhes acerca da liberação de espaço por parte da plataforma são dependentes de implementação. Porém a persistência dos arquivos deve ser garantida enquanto a aplicação estiver em execução ou sinalizada na tabela AIT com um *control code* diferente de KILL ou DESTROY.

8.7.5 Acesso às propriedades do sistema

O acesso às propriedades do sistema é feito pelos métodos *java.lang.System.getProperty*, *java.lang.System.getProperties* e *java.lang.System.setProperty.*

O uso do método *java.lang.System.setProperties()* não é permitido para aplicações Ginga-J.

8.7.6 Suporte a IP sobre canal de interatividade

Para dispositivos onde for necessário o estabelecimento (*setup*) de conexão, o uso direto de java.net.Socket ou java.net.URLConnection deve obrigatoriamente resultar em uma tentativa de conexão segundo parâmetros enviados no arquivo de permissões da aplicação. Para mais informações, ver JAVADTV 1.3:2009, "*Per Application Policy Schema*".

Para manipulação dos dispositos de interatividade disponíveis na plataforma, deve-se utilizar a API "*Extensive communication device control*" (ver JAVADTV 1.3:2009). Definições adicionais sobre o uso da API padrão para conexão *java.net e javax.microedition.io* (ver PBP 1.1:2008) são encontradas na documentação da classe *com.sun.dtv.net.NetworkDevice* (ver JAVADTV 1.3:2009), bem como em JAVADTV 1.3:2009, 6.2.5.

8.7.7 Filtragem de seção MPEG-2

Para obtenção de filtros de seção MPEG-2 deve ser utilizada a API "*Support MPEG-2 Section Filtering*" (ver JAVADTV 1.3:2009).

8.8 Gerenciamento de aplicações

Nesta parte da ABNT NBR 15606, as aplicações de televisão digital são denominadas de Xlets (ver JAVATV 1.1:2008), tendo seu ciclo de vida gerenciado como descrito em 7.1. Estas aplicações executam em um ambiente orientado a serviços controlado por um Gerenciador de Aplicações (ver JAVADTV 1.3:2009), sendo da responsabilidade deste componente as ações de carregar, configurar, instanciar e executar aplicações de televisão digital, bem como controlar o ciclo de vida e estados destas aplicações. É também de responsabilidade do gerenciamento de aplicações atribuir níveis de prioridade de execução às aplicações, além de identificar e mitigar eventuais falhas que ocorram durante a execução destas.

Todo este gerenciamento e monitoramento são realizados internamente pelo sistema. Às aplicações de televisão devidamente assinadas e certificadas, como descrito em JAVADTV 1.3:2009, 6.2, é permitido controlar e/ou monitorar o ciclo de vida destas aplicações. Estas funcionalidades estão acessíveis na JAVADTV 1.3:2009 através do pacote:

- com.sun.dtv.application

8.9 Sintonização

A API br.org.sbtvd.net.tuning é uma extensão do pacote com.sun.dtv.tuner da JAVADTV 1.3:2009. As novas funções são *zapping* de canais de forma interativa e varredura em todas as interfaces da rede existentes em um receptor. Mais detalhes sobre essas funcionalidades podem ser consultados no Anexo C.

8.10 Ponte NCL

A API de ponte NCL, contém o conjunto de classes disponíveis para a ponte entre os aplicativos de informação e processo, em ambiente Ginga. As funções disponíveis nas classes que são descritas abaixo permitem o desenvolvimento de aplicativos de procedimento Ginga-J incluindo aplicativos Ginga-NCL, e vice-versa. A API de ponte NCL Ginga-J controla a apresentação de um documento NCL e oferece recursos para que uma aplicação Ginga-J incluída em um documento NCL seja notificada sobre a ocorrência de eventos de transição realizados sobre o nó de mídia que a encapsula.

Se o Java for instanciado a partir do NCL, o gerenciamento de aplicações deve estar de acordo com ABNT NBR 15606-2:2007, 8.5. Se o NCL for instanciado a partir do Java, o gerenciamento das aplicações deve estar de acordo com o descrito em 8.8.

Informações complementares podem ser obtidas na ABNT NBR 15606-2:2007, 11.2 e 10.3.4.3 e Anexo D.

8.11 Propriedades da plataforma

8.11.1 Propriedades de sistema

Aplicações Ginga-J devem ter acesso a propriedades de plataforma que indiquem características gerais, como perfis e/ou funcionalidades suportadas. Logo, além das propriedades já especificadas no método *java.lang.System.getProperties* (ver PBP 1.1:2008) e no Ginga-NCL (ver ABNT NBR 15606-2:2007, 7.2.4), as propriedades listadas na Tabela 8 devem estar disponíveis por meio dos métodos *getProperty()* e *getProperties()* da classe *java.lang.System* (ver PBP 1.1:2008).

As propriedades de sistema estão descritas na Tabela 8 e não podem ser alteradas pelas Aplicações Ginga-J.

Tabela 8 — Propriedades de sistema

Nome	Tipo de dados	Descrição
com.sun.dtv.version	Alfanumérico	O número de versão da especificação JavaDTV implementada pela plataforma (ver JAVADTV 1.3:2009)
com.sun.dtv.net.default.timeout	Numérico	Tempo máximo de espera (timeout) para estabelecimento de conexão
system.gingaj_version	Alfanumérico	O número de versão da especificação Ginga-J implementada pela plataforma
system.internet_access	Booleano	Indica se o perfil de acesso a internet está disponível (valores possíveis: “true” ou “false”)
system.gingaj_profile	Alfanumérico	O número do perfil da especificação Ginga-J suportado pela plataforma

8.11.2 Propriedades de usuário

Em adição ao conjunto de propriedades de sistema, aplicações Ginga-J podem manter propriedades específicas por usuário da plataforma. Para tanto, deve ser utilizada a API "*User preferences*" (ver JAVADTV 1.3:2009).

A API "User preferences" (ver JAVADTV 1.3:2009) também deve ser usada para recebimento de notificações de mudanças em quaisquer das propriedades de sistema definidas em 8.11.1.

8.12 Canal de interatividade

O sistema Ginga possui a habilidade de identificar a existência de dispositivos de canal de interatividade no receptor, podendo estabelecer a conexão caso ele esteja desconectado. A comunicação de dados após a conexão estabelecida se dá através da API *java.net*.*.

8.13 Lista de pacotes mínimos do Ginga-J

8.13.1 Pacotes da plataforma Java

Os seguintes pacotes (ver CDC 1.1:2008; FP 1.1:2008; PBP 1.1:2008) são incluídos esta parte da ABNT NBR 15606:

- java.awt
- java.awt.color
- java.awt.event
- java.awt.font
- java.awt.im
- java.awt.image
- java.beans
- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.rmi
- java.rmi.registry
- java.security
- java.security.acl
- java.security.cert
- java.security.interfaces
- java.security.spec
- java.text

- java.util
- java.util.jar
- java.util.zip
- javax.microedition.io
- javax.microedition.pki
- javax.microedition.xlet
- javax.microedition.xlet.ixc
- javax.security.auth.x500

8.13.2 Pacotes da especificação JavaTV 1.1 e JMF 1.0

Os seguintes pacotes (ver JAVATV 1.1:2008) são incluídos por esta parte da ABNT NBR 15606:

- javax.media
- javax.media.protocol
- javax.tv.graphics
- javax.tv.locator
- javax.tv.media
- javax.tv.net
- javax.tv.service
- javax.tv.service.guide
- javax.tv.service.navigation
- javax.tv.service.selection
- javax.tv.service.transport
- javax.tv.util
- javax.tv.xlet

8.13.3 Pacotes da especificação JAVADTV 1.3

Os seguintes pacotes (ver JAVADTV 1.3:2009) são incluídos por esta parte da ABNT NBR 15606:

- com.sun.dtv.application
- com.sun.dtv.broadcast
- com.sun.dtv.broadcast.event

- com.sun.dtv.filtering
- com.sun.dtv.io
- com.sun.dtv.locator
- com.sun.dtv.lwuit
- com.sun.dtv.lwuit.animations
- com.sun.dtv.lwuit.events
- com.sun.dtv.lwuit.geom
- com.sun.dtv.lwuit.layouts
- com.sun.dtv.lwuit.list
- com.sun.dtv.lwuit.painter
- com.sun.dtv.lwuit.plaf
- com.sun.dtv.lwuit.util
- com.sun.dtv.media
- com.sun.dtv.media.audio
- com.sun.dtv.media.control
- com.sun.dtv.media.dripfeed
- com.sun.dtv.media.format
- com.sun.dtv.media.language
- com.sun.dtv.media.text
- com.sun.dtv.media.timeline
- com.sun.dtv.net
- com.sun.dtv.platform
- com.sun.dtv.resources
- com.sun.dtv.security
- com.sun.dtv.service
- com.sun.dtv.smartcard
- com.sun.dtv.test
- com.sun.dtv.transport

- com.sun.dtv.tuner
- com.sun.dtv.ui
- com.sun.dtv.ui.event

8.13.4 Pacotes da especificação JSSE 1.0.1

Os seguintes pacotes (ver JSSE 1.0.1) são incluídos por esta parte da ABNT NBR 15606:

- com.sun.net.ssl
- javax.net
- javax.net.ssl
- javax.security.cert

8.13.5 Pacotes da especificação JCE 1.0.1

Os seguintes pacotes (ver JCE 1.0.1) são incluídos por esta parte da ABNT NBR 15606:

- javax.crypto
- javax.crypto.interfaces
- javax.crypto.spec

8.13.6 Pacotes da especificação SATSA 1.0.1

O seguinte pacote (ver SATSA 1.0.1) é incluído por esta parte da ABNT NBR 15606:

- javax.microedition.apdu

8.13.7 Pacotes específicos Ginga-J

Os seguintes pacotes específicos do Ginga-J são incluídos por esta parte da ABNT NBR 15606:

- br.org.sbtvd.net
- br.org.sbtvd.net.si
- br.org.sbtvd.net.tuning
- br.org.sbtvd.bridge
- br.org.sbtvd.ui

Os requisitos mínimos para um receptor compatível com Ginga devem estar de acordo com a ABNT NBR 15604:2007.

Anexo A
(normativo)

Especificação JavaDTV 1.3

A.1 Considerações gerais

A especificação JAVADTV 1.3:2009 é composta pelas API Java DTV adicionadas à base dos componentes do ambiente de execução Java, incluindo ainda *Connected Device Configuration*, *Foundation Profile* e *Personal Basis Profile*.

Java DTV é uma especificação que oferece funcionalidades de um receptor de televisão digital para o desenvolvimento de aplicações baseadas em Java. Neste Anexo são descritos os principais elementos da especifação JAVADTV 1.3:2009.

A.2 API Java DTV

NOTA Para mais detalhes sobre a API Java DTV, consultar http://java.sun.com/javame/technology/javatv/index.jsp

A.2.1 Pacote com.sun.dtv.broadcast

O pacote com.sun.dtv.broadcast permite o acesso aos arquivos de *broadcast* e *streams* e deve ser implementado de acordo com a JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.1, devem ser empregadas.

Tabela A.1 — Classes do pacote com.sun.dtv.broadcast

Classe	Descrição
Classe BroadcastFile	Representa dados do arquivo ou diretório obtido a partir de do canal de difusão
Classe BroadcastFileEvent	Indica notificações de mudança para os dados do BroadcastFile
Classe BroadcastFilesystem	Representa instâncias do sistema de arquivo obtido do canal de difusão e montado dentro do sistema de arquivos local
Interface BroadcastFileListener	Implementada por classes de aplicação que desejem receber notificações de mudança para os dados de BroadcastFile
Classe BroadcastStream	Representa fluxos obtidos através de arquivos obtidos do canal de difusão
Classe BroadcastException	Todas as exceções relacionadas com broadcast devem usar esta subclasse (que por sua vez é uma extensão de java.io.IOException) com o intuito de tornar mais fácil a identificação dos motivos que levaram a ocasionar a exceção

A.2.2 Pacote com.sun.dtv.smartcard

O pacote com.sun.dtv.smartcard fornece suporte a funcionalidades adicionais para utilização de smartcards.

A implementação deste pacote precisa estar de acordo com a JAVADTV 1.3:2009, incluindo as interfaces e classes descritas na Tabela A.2.

Tabela A.2 — Classes do pacote com.sun.dtv.smartcard

Classes	Descrição
Interface CardTerminalListener	Interface ouvinte, responsável por receber eventos provenientes de leitores de smartcards do dispositivo
Interface PassThroughAPDUConnection	Interface de marcação que identifica os objetos de conexão específicos da APDU (*Application Protocol Data Unit*), que realizam comunicação direta com o smartcard
Classe TerminalFactory	Fornece acesso ao(s) dispositivo(s) leitor(es) de smartcards implementados ou conectados ao dispositivo
Classe CardTerminalEvent	Define um objeto de evento que informa aos *listeners* sobre as mudanças de estado ocorridas no leitor de smartcard
Classe CardTerminal	Encapsula as funcionalidades da parte física do leitor de smartcard

A.2.3 Pacote com.sun.dtv.lwuit.events

O pacote com.sun.dtv.lwuit.events implementa o padrão de projeto Observable (também utilizado na API AWT 1.1) que define uma arquitetura de tratamento para lançamento e tratamento de eventos em interfaces gráficas. Todos os eventos são disparados por um *thread* chamado de *Event Dispatch Thread* (EDT). Consultar a documentação da API para mais detalhes.

Este pacote deve ser implementado de acordo com a JAVADTV 1.3:2009, onde as interfaces e classes deste pacote, descritas na Tabela A.3, devem ser empregadas.

Tabela A.3— Classes do pacote com.sun.dtv.lwuit.events

Classes	Descrição
Interface DataChangedListener	Interface para tratamento de evento de *callback*, que são invocados quando ocorrem mudanças de estado e indicam que a visualização deve ser atualizada
Interface FocusListener	Ouvinte de eventos de mudança de foco para *Form* permite que funcionalidades sejam atribuídas de acordo com o foco atual do componente
Interface SelectionListener	Invocado para indicar uma mudança na seleção da lista modelo
Interface StyleListener	Invocado para indicar mudanças em uma propriedade
Classe ActionEvent	Evento de objeto disparado quando um *callback* é invocado
Interface ActionListener	Interface de eventos de *callback* invocado quando uma ação de componente ocorre

A.2.4 Pacote com.sun.dtv.filtering

O pacote com.sun.dtv.filtering oferece suporte para acesso a seções de fluxo de transporte MPEG-2. Além de permitir diversos tipos de filtragem, a API utiliza um modelo assíncrono que notifica sobre eventos ou erros na leitura das seções.

Este pacote deve ser implementado de acordo com a JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.4, devem ser empregadas.

Tabela A.4 — Classes do pacote com.sun.dtv.filtering

Classes	Descrição
Classe DataSectionFilterCollection	Esta classe representa uma coleção de filtros de seções de dados, onde eles podem ser ativados e desativados como uma única operação
Classe FilterTimedOutEvent	Este evento é disparado se operações de filtros de seção de dados expirarem de acordo com o período especificado pelo método *setTimeOut*()
Classe DataSectionFilterException	Classe-base para o lançamento de exceções da API de filtros de seção de dados
Classe DataSectionAvailableEvent	Este evento indica que uma seção de dados foi completamente processada
Classe DataSectionFilter	Esta é a classe raiz, de todas as classes de filtro
Classe DataSection	Esta classe encapsula uma sessão de dados de um fluxo de transporte. Os objetos dessa classe também são clonáveis
Classe IncompatibleSourceException	Classe que informa quando é fornecido um fluxo de origem incompatível
Classe DataSectionDataBuffer	Esta classe encapsula uma parte da sessão do fluxo de transporte carregado
Classe FilteringStoppedEvent	Esta classe é utilizada para informar o final de uma operação de filtragem, com uma excepção: Ela não informa quando uma *SimpleSectionFilter* termina em condições normais (por exemplo, após uma seção de filtragem ter sido realizada com sucesso)
Classe DataUnavailableException	Informa quando as informações de um objeto *DataSection* não estão disponíveis
Classe DataSectionFilterEvent	É uma classe-base para eventos da API de filtros de seção
Classe FilterInterruptException	Sinaliza que um filtro foi interrompido antes de todos os dados terem sido filtrados
Classe DisconnectedException	Indica que quando o *DataSectionFilterCollection* perdeu a conexão por causa de uma falha na chamada do método *startFiltering*()
Classe InvalidFilterException	Sinaliza que um filtro foi definido incorretamente
Classe FilterResourceUnavailableException	Informa que os requisitos necessários para uma operação de filtragem não foram satisfeitos
Classe CircularFilter	Esta classe define um filtro de seção destinado para captar fluxos contínuos de dados sem que seja preciso reiniciar o filtro continuamente
Interface FilterEventListener	Esta interface de ouvinte pode ser implementada por classes que exigem eventos de filtragem
Classe ListFilter	Esta classe define um filtro de seção que pode processar um conjunto completo de segmentos de uma seção de dados que representam uma tabela de seção simples
Classe SingleFilter	Esta classe define um filtro de seção destinado a capturar uma única seção de dados

A.2.5 Pacote com.sun.dtv.ui.event

O subpacote com.sun.dtv.ui.event tem a função de tratamento de eventos de interface gráfica com o usuário específica para televisão digital.

Este pacote deve ser implementado de acordo com a JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.5, devem ser empregadas.

Tabela A.5 — Classes do pacote com.sun.ui.event

Classes	Descrição
Interface *KeyListener*	Funciona apenas como um encapsulador para introduzir *java.awt.event.KeyListener* na API LWUIT, já que ela não suporta objetos de eventos de teclas e fornece apenas um ActionListener
Classe *PlaneSetupEvent*	Evento enviado para todos os registrados quando uma configuração de Plane é alterada
Classe *MouseEvent*	Estende *java.awt.event.MouseEvent*, implementando a interface *com.sun.dtv.ui.eventUserInputEvent*. Funciona apenas como uma interface de marcação, já que *UserInputEvent* é derivada da interface *ScarceResource*
Interface *UserInputEvent*	Abstração para eventos de entrada de usuário enviados a todos os consumidores registrados quando ocorre uma nova entrada de usuário através de um *UserInputDevice*
Interface *UserInputEventListener*	Deve ser implementada por classes que desejem receber *UserInputEvent*
Classe *KeyEvent*	Estende *java.awt.event.KeyEvent*, implementando a interface com.sun.dtv.ui.eventUserInputEvent. Funciona apenas como uma interface de marcação, já que *UserInputEvent* é derivada da interface *ScarceResource*
Interface *MouseListener*	Funciona apenas como um encapsulador para introduzir *java.awt.event.MouseListener* na API LWUIT, já que ele não suporta objetos de eventos de mouse e fornece apenas um *ActionListener*
Classe *UserInputEventManager*	As instâncias desta classe são gerentes de eventos que tratam eventos de entrada do usuário gerados por componentes gráficos
Classe *RemoteControlEvent*	Estende *com.sun.dtv.ui.event.KeyEvent* adicionando códigos de tecla específicos de televisão
Interface *PlaneSetupListener*	Utilizada para monitorar mudanças na configuração de um *Plane*

A.2.6 Pacote com.sun.dtv.lwuit.plaf

O pacote com.sun.dtv.lwuit.plaf permite que a aparência da aplicação seja customizada. Neste caso, utiliza-se uma abstração de uma camada de renderização que pode ser acoplada e configurada (com temas, por exemplo) em tempo de execução.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.6, devem ser empregadas.

Tabela A.6 — Classes do pacote com.sun.lwui.plaf

Classes	Descrição
Classe *LookAndFeel*	Permite aos programadores personalizar completamente a aparência da aplicação, através de métodos adequados de sobreposição de imagens e dimensionamento
Classe *UIManager*	O ponto central para o gerenciamento da aparência da aplicação permite personalizar os estilos (temas), bem como a aparência
Classe *Style*	Representa a aparência de um determinado componente
Classe *Border*	Classe-base que permite criar uma moldura para um componente; a moldura é desenhada antes do componente preenchendo sua região marginal
Classe *DefaultLookAndFeel*	Utilizado para renderizar a aparência padrão do LWUIT 1.1:2008

A.2.7 Pacote com.sun.dtv.media.timeline

O pacote com.sun.dtv.media.timeline permite obter e definir os tempos das mídias a serem notificados sobre eventos temporais lançados durante a reprodução das mídias.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.7, devem ser empregadas.

Tabela A.7 — Classes do pacote com.sun.dtv.media.timeline

Classes	Descrição
Interface *MediaTimeListener*	Interface de escuta para o recebimento de eventos temporais de mídias
Interface *MediaTimePositionControl*	Controla a definição e obtenção de *mediaTime* de um fluxo de mídia. A duração máxima da mídia pode ser obtida a partir da interface Duration se ela for implemenada pelo tipo de mídia em reprodução
Interface *MediaTimeEvent*	Evento que informa a aplicação sobre mudanças no tempo das mídias

A.2.8 Pacote com.sun.dtv.media.language

O pacote com.sun.dtv.media.language fornece as funcionalidades básicas de consulta e definição dos idiomas de: áudio, legendas e *Close Captioning*. Os idiomas são codificados em três línguas, de acordo com as ISO 639-2 e ABNT NBR 15603-2, para o uso em descritores de serviço.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.8, devem ser empregadas.

Tabela A.8 — Classes do pacote com.sun.dtv.media.language

Classes	Descrição
Classe *LanguageNotSupportedException*	Exceção levantada quando um idioma solicitado não é suportado
Interface *LanguageControl*	Controle para consulta dos idiomas suportados e para definir um idioma específico para um *player*

A.2.9 Pacote com.sun.dtv.application

O pacote com.sun.dtv.application define o modelo de aplicações JavaDTV, seu ciclo de vida e gerenciamento. As aplicações JavaDTV executam através de um ambiente de execução multitarefa Java DTV e é através deste pacote que o desenvolvedor pode acessar este ambiente para verificar quais aplicações estão disponíveis (*AppManager*), os atributos de cada aplicação (*Application*) e realizar monitoração e controle sobre elas (*ApplicationProxy*). Este pacote também implementa o ciclo de vida de aplicações através da utilização de classes da API JavaTV.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote descritas na Tabela A.9 devem ser incluídas.

Tabela A.9 — Classes do pacote com.sun.dtv.application

Classes	Descrição
Interface *AppManagerListener*	Ouvinte que notifica sobre alterações nos aplicativos disponíveis
Classe *AppManager*	Fornece controle e acesso às aplicações
Interface *ApplicationProxy*	Fornece controle a uma aplicação através do gerente de aplicação
Interface *Application*	Contém os atributos de uma aplicação
Classe *AppFilter*	*AppFilter* é chamado para permitir uma aplicação em um filtro ou não
Class AppManagerPermission	Necessária para as consultas ou controles das aplicações
Interface AppProxyListener	Ouvinte que recebe notificações de alterações de estado da aplicação

A.2.10 Pacote com.sun.dtv.media.audio

O pacote com.sun.dtv.media.audio oferece funcionalidades referentes ao controle do idioma do áudio, como, por exemplo:

— consultar os idiomas que podem ser selecionados para associação com um *Service Component*;

— selecionar o idioma associado a um *Service Component*;

— permitir que as aplicações recebam notificações quando o idioma de um Service Component for alterado.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.10, devem ser empregadas.

Tabela A.10 — Classes do pacote com.sun.dtv.media.auto

Classes	Descrição
Interface *AudioEvent*	Indica mudanças relacioanadas com a seleção do idioma do áudio
Interface *AudioControl*	Fornece controle de áudio para registrar eventos específicos de áudio
Interface *AudioListener*	Ouvinte para mudança no áudio

A.2.11 Pacote com.sun.dtv.test

O pacote com.sun.dtv.test provê um *framework* para realização de testes de conformidade. Como normalmente é encontrado em ambientes de teste Java, o ambiente definido para uso do pacote envolve um servidor e um cliente de testes que pode utilizar qualquer meio de comunicação.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.11, devem ser empregadas.

Tabela A.11 — Classes do pacote com.sun.dtv.test

Classes	Descrição
Interface *TestCase*	Define os métodos necessários que devem ser implementados pelos testes para serem capazes de executar no framework de teste
Classe *TestHarness*	Fornece um ponto de entrada e um conjunto de ferramentas para os casos de testes
Classe *Report*	Agrega o resultado de um teste: o código, a referência para o teste e os motivos relacionados

A.2.12 Pacote com.sun.dtv.tuner

O pacote com.sun.dtv.tuner fornece API para o acesso e o controle de uma interface de rede de difusão (ou "tuner"), utilizados para a recepção de fluxos de transporte.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.12, devem ser empregadas.

Tabela A.12 — Classes do pacote com.sun.dtv.tuner

Classes	Descrição
Classe *Tuner*	Representa uma interface de rede ou "*tuner*" para a recepção de fluxos de transporte de radiodifusão
Classe *TuningCompletedEvent*	Evento que indica a conclusão bem-sucedida de uma operação de sintonização
Classe *TuningFailedEvent*	Evento que indica a conclusão mal-sucedida de uma operação de sintonização
Classe *TuningException*	Exceção que indica relatórios síncronos de falhas de sintonização
Classe *TuningEvent*	Classe-base para os eventos gerados pelas operações de sintonização
Interface *TunerListener*	Ouvinte que recebe eventos de sintonização vindos do *Tuner*
Classe *TuningInitiatedEvent*	Indica o início de uma operação de sistonização

A.2.13 Pacote com.sun.dtv.lwuit.layouts

O pacote com.sun.dtv.lwuit.layouts contém o modelo de gerenciamento de layout da LWUIT 1.1:2008, que é similar ao modelo utilizado pelas APIS AWT/Swing. Neste caso, os gerenciadores de *layout* permitem que um contêiner arranje seus componentes por um conjunto de regras predefinidas e que podem se adaptar para tamanho de fontes ou tela específicos.

Este pacote deve ser implementado em conformidade com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.13, devem ser empregadas.

Tabela A.13 — Classes do pacote com.sun.dtv.lwuit.layouts

Classes	Descrição
Classe *GridLayout*	Através do uso dessa classe como gerenciador de layout, os componentes são dispostos em uma grade de tamanho igual, baseado no espaço disponível
Classe BorderLayout	Define um contêiner que organiza e redimensiona os seus componentes numa disposição baseada em cinco regiões: norte, sul, leste, oeste e centro
Classe *LayoutStyle*	*LayoutStyle* é usado para determinar quanto espaço será utilizado entre os componentes
Classe *BoxLayout*	Dispõe elementos em uma linha ou coluna, de acordo com uma orientação de caixa
Classe *FlowLayout*	Dispõe os componentes em uma fila de modo que, quando chega um fim de uma linha, eles passam para a linha seguinte
Classe *Layout*	Classe abstrata, que pode ser usada para organizar componentes em um contêiner utilizando um algoritmo predefinido
Classe *GroupLayout*	Gerenciador de layout que agrupa hierarquicamente componentes
Classe *CoordinateLayout*	Permite que os componentes baseados em posições e tamanhos absolutos sejam adaptados com base no espaço disponível para o layout

A.2.14 Pacote com.sun.dtv.broadcast.event

Trata os eventos gerados pelo canal de broadcast. Neste caso, a classe *BroadcastEventManager* lida com todos os eventos e repassa para os BroadcastEventListener registrados.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.14, devem ser empregadas.

Tabela A.14 — Classes do pacote com.sun.dtv.broadcast.event

Classes	Descrição
Classe *BroadcastReceivedEvent*	Representa os eventos que foram recebidos através do canal de difusão
Classe *BroadcastEventManager*	Representa os eventos obtidos através do sistema de arquivos do canal de difusão
Interface *BroadcastEventListener*	Implementada pelas classes da aplicação que requisitam notificação de recebimento dos dados do *BroadcastEvent*

A.2.15 Pacote com.sun.dtv.lwuit.list

O pacote com.sun.dtv.lwuit.list relacionado com a manipulação de *List* que emprega o mesmo modelo MVC do *Swing*, incluindo o padrão de projeto *renderer*. O pacote possui duas interfaces e duas classes. A interface *ListCellRenderer* possibilita a customização da aparência da *List* e a *ListModel* define a representação da estrutura de dados que é utilizada pela List como fonte das suas informações. As classes são implementações default das interfaces.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.15, devem ser empregadas.

Tabela A.15 — Classes do pacote com.sun.dtv.lwuit.list

Classes	Descrição
Classe *DefaultListCellRenderer*	Implementação padrão do renderizador baseado em um rótulo
Interface *ListModel*	Representa a estrutura dos dados da lista, permitindo assim que uma lista represente qualquer fonte de dados em potencial através da referencia para diferentes implementações dessa interface
Interface *ListCellRenderer*	Interface que define um renderizador de rótulos de uma List. Neste caso, representa apenas o marcador de cada célula selecionada na *List*
Classe *DefaultListModel*	Implementação padrão da list model baseada em um vetor de elementos

A.2.16 Pacote com.sun.dtv.ui

O pacote com.sun.dtv.ui é o pacote com funcionalidades de interface gráfica com o usuário específicas para televisão digital.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.16, devem ser empregadas.

Tabela A.16 — Classes do pacote com.sun.dtv.ui

Classes	Descrição
Classe *MatteException*	Um *MatteException* é disparado quando um aplicativo, por algum motivo, não é capaz de realizar uma associação com um elemento gráfico
Classe *DTVContainerPattern*	O *DTVContainerPattern* é um meio para descrever as características de um DTVContainer válido
Classe *Device*	Esta classe é a representação do dispositivo de televisão
Classe *Screen*	Esta classe é a representação da tela do dispositivo de televisão
Classe *DefaultTextLayoutManager*	Esta classe fornece um mecanismo padrão de renderização de textos
Classe *FontSpecificationException*	Exceção lançada quando uma tentativa de especificar características de uma fonte n de forma incorreta. Neste caso, apenas para fontes não definidas num arquivo
Interface *ViewOnlyComponent*	Esta classe representa qualquer tipo de componente não interativo do sistema. Utiliza também um mecanismo de configuração da sua aparência
Classe *UserInputDevice*	Base para todos os dispositivos de entrada que podem ser usados para controlar a tela do dispositivo
Classe *Mouse*	Esta classe representa o mouse que pode ser usado para controlar uma Screen particular de um *UserInputDevice*
Classe *AlphaComposite*	Implementa todas as regras de composição de alfa (transparência)
Classe *DTVContainer*	Um container de nível alto na hierquarquia de components da API Java DTV que representa um elemento gráfico contendo qualquer coisa visível em um determinado Plane. Neste caso, sempre existe apenas um DTVCointaner para cada Plane
Interface *TextLayoutManager*	Define as funcionalidades para o layout dos textos e de sua exibição na tela
Classe *Capabilities*	Descreve as capacidades de um Plane
Classe *SetupException*	Exceção que pode ser lançada em várias situações onde é feita uma tentativa de realizar uma mudança ilegal na configuração de um ou mais Planes de uma *Screen*
Classe *SophisticatedTextLayoutManager*	Esta classe fornece uma *TextLayoutManager* com mais funcionalidades, de modo a permitir um layout mais sofisticado
Classe *PlaneSetupPattern*	Esta classe possibilita um meio de descrever as configurações de um plano de exibição, especificando várias propriedades e sua importância para a aplicação
Classe *AnimatedMatte*	Esta classe representa um matte animado com uma máscara de imagem dinâmica, onde os valores dos pixels determinam a transparência do matte em um determinado tempo
Classe *RemoteControl*	Esta classe representa um controle remoto de televisão que pode ser utilizado para controlar uma tela em particular como um *UserInputDevice*
Interface *TextOverflowListener*	Notifica se uma cadeia de caracteres não couber em um componente durante a tentativa de renderizá-lo
Classe *Keyboard*	Esta classe representa um teclado que pode ser utilizado para controlar determinadas telas como um *UserInputDevice*
Classe *FontFileException*	Esta exceção será lançada numa tentativa de ler um arquivo de fonte com um formato inapropriado
Classe *PlaneSetup*	Descreve as características de um *Plane*
Classe *DownloadableFont*	Introduz a possibilidade de baixar fontes
Interface *Matte*	Interface básica para todas as classe *Matte*
Interface *Animated*	Esta interface fornece métodos para definir e obter parâmetros de uma animação
Interface *MatteEnabled*	Permite que componentes façam composição de matte
Classe *StaticMatte*	Esta classe representa *matte* não animados, por exemplo, *mattes* que não mudam durante um intervalo de tempo
Classe *Plane*	Representa uma saída de vídeo de um dispositivo de televisão

A.2.17 Pacote com.sun.dtv.media.control

O pacote com.sun.dtv.media.control dispõe de controles adicionais para obtenção de informações sobre a mídia em exibição.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.17, devem ser empregadas.

Tabela A.17 — Classes do pacote com.sun.dtv.media.control

Classes	Descrição
Interface *FrameRateControl*	Controle para obter a taxa de quadros
Interface *MpegAudioControl*	Controle para obter os parâmetros de um fluxo de áudio MPEG
Interface *BitRateControl*	Controle para obter a taxa de bits

A.2.18 Pacote com.sun.dtv.media.dripfeed

O pacote com.sun.dtv.media.dripfeed permite a entrega de dados de uma imagem estática para um JMF Player, de forma a permitir que a aplicação tenha o controle sobre os dados. As imagens podem ser de *frames* retirados de uma fonte de vídeo.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.18, devem ser empregadas.

Tabela A.18 — Classes do pacote com.sun.dtv.media.dripfeed

Classes	Descrição
Interface *DripFeedControl*	Permite a alimentação progressiva de partes de um vídeo em um *Player*
Classe *DripFeedPermission*	Representa as permissões para acessar o *DripFeedControl*

A.2.19 Pacote com.sun.dtv.security

O pacote com.sun.dtv.secutity inclui funcionalidades adicionais de segurança. As características básicas são fornecidas pelas classes no java.security pacote.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.19, devem ser empregadas.

Tabela A.19 — Classes do pacote com.sun.dtv.security

Classes	Descrição
Interface *CallbackHandler*	Uma aplicação implementa um *CallbackHandler* e repassa para os serviços de segurança. Dessa forma a aplicação pode interagir com os serviços de segurança com o objetivo de recuperar informações específicas de autenticação, como, por exemplo, *username* e *password* ou para exibir determinada informação, por exemplo, mensagens de erros ou avisos
Interface *Callback*	Implementações dessa interface são passadas um *CallbackHandler,* permtindo que serviços de segurança possam interagir com a aplicação chamada e assim recuperar informações específicas de autenticação, como, por exemplo, *username* e *password* ou para exibir determinada informação, por exemplo, mensagens de erros ou avisos
Classe *AuthProvider*	Esta classe define métodos de *login* e *logout* para um provedor
Classe *LoginException*	Exceção relacionada a operações de *login*
Classe *UnsupportedCallbackException*	Disparado quando uma chamada de *callback* é passada e não pode ser tratada pelo destinatário da chamada

A.2.20 Pacote com.sun.dtv.lwuit.painter

O pacote com.sun.dtv.lwuit.painter contém classes que estendem funcionalidades da interface com.sun.dtv.lwuit.Painter e que permite desenhar elementos gráficos arbitrários no background de componentes (com.sun.dtv.lwuit.Component).

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.20, devem ser empregadas.

Tabela A.20 — Classes do pacote com.sun.dtv.lwuit.painter

Classes	Descrição
Classe *PainterChain*	Permite encadear vários painters de modo a obter um efeito em "camada" e onde cada painter desenha apenas um elemento
Classe *BackgroundPainter*	Desenha o fundo de tela de um componente baseado no seu estilo

A.2.21 Pacote com.sun.dtv.locator

O pacote com.sun.dtv.locator define todos os *Locators* a serem utilizados no sistema Java DTV.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.21, devem ser empregadas.

Tabela A.21 — Classes do pacote com.sun.dtv.locator

Classes	Descrição
Classe *EntityLocator*	Locator de entidades nos sectores dos transportes de fluxos
Classe *URLLocator*	Locator baseados em URL
Interface *TransportDependentLocator*	Locator que referencia as entidades de um fluxo de transporte
Classe *NetworkBoundLocator*	Locator que referencia entidades que estão vinculadas à rede

A.2.22 Pacote com.sun.dtv.resources

O pacote com.sun.dtv.resources fornece um *framework* básico para dispositivos com recursos limitados.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.22, devem ser empregadas.

Tabela A.22 — Classes do pacote com.sun.dtv.resources

Classes	Descrição
Classe *TimeoutException*	Sinaliza quando um *timeout* ocorre
Interface *ScarceResourceListener*	Notifica sobre liberação de um determinado recurso escasso
Interface *ScarceResource*	Representa recursos que precisam de tratamento especial para reservar e libertar
Classe *ScarceResourcePermission*	Utilizada para lidar com as diversas permissões relacionadas com recursos escassos
Interface *ResourceTypeListener*	Notifica o status de alterações ocorridas nos recursos do mesmo tipo do objeto para que o ouvinte tenha se conectado

A.2.23 Pacote com.sun.dtv.net

O pacote com.sun.dtv.net estende o pacote java.net para suporte a controle de extensivo de comunicação com dispositivos. Neste caso representa um dispositivo através da classe *NetworkDevice*.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.23, devem ser empregadas.

Tabela A.23 — Classes do pacote com.sun.dtv.net

Classes	Descrição
Interface *NetworkDeviceStatusListener*	Ouvinte para eventos relacionados com dispositivos de rede
Classe *NetworkDevicePermission*	Utilizada para lidar com as diversas permissões relacionadas com recursos escassos de dispositivos de rede
Classe *NetworkDevice*	Representa cada instância física de qualquer interface de rede em suporte o protocolo IP (TCP, UDP). A comunicação pode ser obtida através da plataforma

A.2.24 Pacote com.sun.dtv.media.text

O pacote com.sun.dtv.media.text permite acesso ao controle de legenda e "*Closed Captioning*". Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.24, devem ser empregadas.

Tabela A.24 — Classes do pacote com.sun.dtv.media.text

Classes	Descrição
Interface *OverlayTextEvent*	Eventos que reportam mudanças em: “*OverlayText*”, “*Subtitle*” e “*Closed Captioning*”
Interface *OverlayTextControl*	Fornece controle sobre "*Subtitles*" e "*Closed Captioning*"
Interface *OverlayTextListener*	Recebe eventos relacionados a “*OverlayText*”

A.2.25 Pacote com.sun.dtv.media.format

O pacote com.sun.dtv.media.format é responsável pelas configurações do formato de vídeo.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.25, devem ser empregadas.

Tabela A.25 — Classes do pacote com.sun.dtv.media.format

Classes	Descrição
Interface *VideoFormatControl*	Fornece os meios para obter informações sobre o formato e relação de aspecto do vídeo
Interface *VideoFormatListener*	Interface de escuta que receber notificação de eventos sobre mudanças na apresentação do vídeo
Interface *AspectRatioEvent*	A interface *VideoFormatEvent* informa sobre mudanças ocorridas na razão de aspecto
Interface *VideoPresentationControl*	Fornece os meios para consultar e manipular a apresentação do vídeo
Classe *VideoPresentationEvent*	Evento que informa sobre mudanças ocorridas na apresentação do vídeo
Interface *ActiveFormatEvent*	O *VideoFormatEvent* informa sobre as mudanças no *Active Format*
Interface *VideoPresentationListener*	Relata sobre mudanças na apresentação do vídeo, bem como todos os tipos de *ContollerEvent*
Interface *DecoderFormatEvent*	Evento que informa que o formato do decodificador mudou
Interface *ClippingControl*	Controle que recupera e define o recorte retangular do vídeo
Interface *VideoFormatEvent*	Evento que informa sobre as mudanças no formato vídeo
Interface *BackgroundVideoPresentationControl*	Controle de vídeos mostrados no fundo de tela
Interface *ArbitraryVideoScalingControl*	Controle para recuperar os fatores arbitrários de escala do video
Classe *Transformation*	Representa um container para informações de transformação para um vídeo

A.2.26 Pacote com.sun.dtv.platform

O pacote com.sun.dtv.platform fornece classes que são específicas da plataforma Java DTV, em particular as classes relacionadas ao tratamento de usuários.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.26, devem ser empregadas.

Tabela A.26 — Classes do pacote com.sun.dtv.platform

Classes	Descrição
Classe *UserPropertyPermission*	Descreve permissões para as propriedades do usuário
Interface *UserPropertyListener*	Como alternativa, uma aplicação pode anexar um *UserPropertyListener* nas propriedades do usuário do subsistema, com o intuito de ser notificado sobre quaisquer alterações nas propriedades dos usuários
Classe *User*	Contém vários campos e métodos que são específicos para cada usuário da plataforma

A.2.27 Pacote com.sun.dtv.io

O pacote com.sun.dtv.io estende o pacote *java.io,* fornecendo o acesso aos direitos e propriedades dos arquivos.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.27, devem ser empregadas.

Tabela A.27 — Classes do pacote com.sun.dtv.io

Classes	Descrição
Classe *FileProperties*	Utilizado para associar propriedades (ou metadados) com um arquivo identificado pelo seu “pathname” em um determinado sistema de arquivos
Classe *FileAccessRights*	Fornece um meio para definir grupos de níveis de direito de acesso a um arquivo ou diretório

A.2.28 Pacote com.sun.dtv.lwuit.animations

No pacote com.sun.dtv.lwuit.animations, todos os componentes são animações em potencial e podem ser rodados em tempo de execução; transições entre *Form*s também são tratadas como parte deste pacote. As *threads* de animação são tratadas de modo uniforme para diminuir a complexidade para a execução em dispositivos computacionalmente limitados.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.28, devem ser empregadas.

Tabela A.28 — Classes do pacote com.sun.dtv.lwuit.animations

Classes	Descrição
Classe *Transition*	Representa uma transição com animação entre dois Forms; esta classe é usada internamente pelo *DTVContainer* para reproduzir uma animação quando se desloca de um *Form* para o seguinte
Classe *CommonTransitions*	Contém animações de transição comuns
Classe *Motion*	Abstração da noção de movimento físico ao longo do tempo entre dois pontos representados por valores numéricos
Interface *Animation*	Permite que qualquer componente receba enventos de animação em intervalos de tempos e seja atualizado

A.2.29 Pacote com.sun.dtv.service

O pacote com.sun.dtv.service fornece uma interface para acessar o banco de dados do SI (*Service Information*).

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.29, devem ser empregadas.

Tabela A.29 — Classes do pacote com.sun.dtv.service

Classes	Descrição
Classe *SIDatabase*	Fornece uma forma genérica para acessar o banco de dados do SI que reside na plataforma

A.2.30 Pacote com.sun.dtv.media

O pacote com.sun.dtv.media é para as funcionalidades relevantes e para os controles de congelar e retornar.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.30, devem ser empregadas.

Tabela A.30 — Classes do pacote com.sun.dtv.media

Classes	Descrição
Interface *FreezeResumeListener*	Permite a aplicação executar os eventos de “congelar” e “retornar” sua reprodução
Interface *FreezeResumeEvent*	Indica se os eventos de congelar ou retomar aconteceram e identificam se eles são provenientes de uma aplicação ou de um usuário
Interface *FreezeEvent*	Indica se uma ação de congelamento aconteceu proveniente de uma aplicação ou de um usuário
Interface *MediaPresentedEvent*	Este evento é gerado após um *javax.media.Player* ser transferido para o estado inicial
Classe *FreezeResumeException*	Esta exceção indica que o método de congelamento ou de retomar foi mal-sucedido
Interface *FreezeResumeControl*	Deve ser implementada para permitir que a aplicação “congele” o *Player*
Classe *ConditionalAccessException*	Indica que uma mídia sobre o controle de um *Player* ou *DataSource* está protegida por acesso condicional
Interface *ResumeEvent*	Indica que a ação de continuar a reprodução aconteceu a partir de uma aplicação ou de um usuário

A.2.31 Pacote com.sun.dtv.transport

O pacote com.sun.dtv.transport fornece acesso as entidades contidas em um fluxo de transporte.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.31, devem ser empregadas.

Tabela A.31 — Classes do pacote com.sun.dtv.transport

Classes	Descrição
Classe *TransportStream*	Representação de um fluxo de transportes e os seus serviços associados
Classe *ConditionalAccessDeniedException*	Esta classe lançada quando é solicitado o acesso às informações que estão codificadas e cujo acesso não é permitido pelo sistema de segurança
Classe *ElementaryStream*	Representação de um fluxo elementar
Classe *Service*	Representação de um serviço contido no fluxo de transporte

A.2.32 Pacote com.sun.dtv.lwuit.util

O pacote com.sun.dtv.lwuit.util provê funcionalidades utilitárias que são específicas de domínio ou não se adéquam a nenhum outro pacote da API.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.32, devem ser empregadas.

Tabela A.32 — Classes do pacote com.sun.dtv.lwuit.util

Classes	Descrição
Classe *Log*	Permite que o desenvolvedor através de um *framework* plugável de logging utilize funcionalidades de log utilizando a API de *file connector*
Classe *Resources*	Está relacionada ao carregamento de recursos (animações, imagens, temas, fontes e etc.) a partir de um arquivo binário gerado no processo de build

A.2.33 Pacote com.sun.dtv.lwuit

O pacote com.sun.dtv.lwuit contém a hierarquia principal de composição de elementos gráficos (com.sun.dtv.lwuit.Component e com.sun.dtv.lwuit.Container) da API LWUIT que segue um modelo idêntico ao das API Swing/AWT. Porém, ao contrário do Swing/AWT, não é utilizado um sistema de janelas de tela cheia. Neste caso é utilizado um modelo parecido com o da API MIDP, que utiliza uma abstração de *display* no qual os elementos gráficos podem ser dispostos.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.33, devem ser empregadas.

Tabela A.33 — Classes do pacote com.sun.dtv.lwuit

Classes	Descrição
Classe *MediaComponent*	Possibilita a inserção e controle de conteúdo de mídia rica
Classe *StaticAnimation*	Uma imagem capaz de animação
Classe *Graphics*	Abstrai a plataforma de contexto gráfico, permitindo a portabilidade entre dispositivos MIDP e CDC
Classe *Container*	Implementa o padrão de projeto *Composite* para *com.sun.dtv.lwuit.Component* de um modo que permite arranjar e relacionar components utilizando um arquitetura de gerenciadores de layout plugáveis
Classe *ComboBox*	Elemento gráfico que representa uma lista que permite apenas uma seleção por vez através da escolha do usuário
Classe *Font*	Uma simples abstração de fontes de plataforma e de biblioteca que permite o uso de fontes que não são suportadas pelo dispositivo
Interface *Painter*	Esta interface pode ser usada para desenhar em componentes de fundo de tela
Classe *RadioButton*	Tipo específico de *com.sun.dtv.lwuit.Button* que mantém um estado de seleção exclusivamente de um *com.sun.dtv.lwuit.ButtonGroup.*
Classe *Calendar*	Possibilita a seleção de valores de data e hora
Classe *TabbedPane*	Permite ao usuário alternar entre um grupo de componentes clicando em um guia com um determinado título e/ou ícone
Classe *Command*	Ação referente aos "*soft buttons*" e menu do dispositivo, similar à abstraçao de comando do MIDP e ações do Swing
Classe *CheckBox*	Botão que pode ser marcado ou desmarcado e ao mesmo tempo exibir seu estado para o usuário
Classe *Form*	Componente de alto nível que é classe-base para interfaces gráficas com o usuário da LWUIT 1.1:2008. O contêiner é dividido em três partes: Title (barra de títulos localizada normalmente na parte superior), ContentPane (espaço central para disposição de elementos entre Title e MenuBar) e MenuBar (barra de menu localizada normalmente na parte inferior)
Classe *Dialog*	Um tipo de *Form* que ocupa uma parte da tela e aparece como uma entidade modal para o desenvolvedor
Classe *Image*	Abstração da plataforma que trata imagens, permitindo manipulá-las como objetos uniformes
Classe *TextField*	Componente para recebimento de entrada de texto de usuário que utiliza uma API mais leve, sem utilizar o suporte nativo para texto do dispositivo
Classe *ButtonGroup*	Esta classe é utilizada para criar um escopo de múltipla exclusão para um conjunto de RadioButtons
Classe *Label*	Permite exibir labels e imagens com diferentes opções de alinhamento, também funciona como classe base para alinhamento da disposição de componentes
Classe *Button*	Componente base para outros elementos gráficos que são clicláveis
Classe *List*	Um conjunto (lista) de elementos que são criados utilizando uma ListCellRenderer e são extraídos através da ListModel
Classe *Component*	Classe base para todos os elementos gráficos da LWUIT 1.1:2008. Utiliza o padrão de projeto *Composite* de forma semelhante à relação de *Container* e *Component* do AWT
Classe *TextArea*	Componente gráfico que permite entrada de textos com múltiplas linhas editáveis, também permite exibir e editar o texto
Classe *AWTComponent*	Estende a classe com.sun.dtv.lwuit.Component como uma variante especial que delega as ações de renderização para java.awt.Component

A.2.34 Pacote com.sun.dtv.lwuit.geom

Contém classes relacionadas à localização geométrica e cálculos de dimensões de componentes gráficos.

Este pacote deve ser implementado de acordo com JAVADTV 1.3:2009. As interfaces e classes deste pacote, descritas na Tabela A.34, devem ser empregadas.

Tabela A.34 — Classes do pacote com.sun.dtv.lwuit.geom

Classes	Descrição
Classe *Point*	Representa uma localização no espaço de coordenadas x e y. Sua precisão é baseada em inteiros
Classe *Rectangle*	Representa uma posição retangular com a dimensão baseada em largura e altura, é útil para medir coordenadas dentro uma aplicação
Classe *Dimension*	Classe utilitária que armazena valores de largura e altura e representa uma dimensão de um componente ou elemento gráfico

Anexo B
(normativo)

Especificação da API de informações de serviço dependente de protocolo

B.1 Considerações gerais

Este Anexo descreve a API de informações de serviço dependente de protocolo do Ginga-J. Esta API é baseada em alterações na especificação ARIB STD-B23:2004, Anexo M. Isto justifica-se pela adoção do ISDB-T (ver ARIB STD-B31:2007) como base da ABNT NBR 15601:2007 Parâmetros relevantes para as informações sobre o serviço de televisão digital são associados ao método de transmissão utilizado, assim, as informações sobre o serviço brasileiro especificados pela ABNT NBR 15603:2007 são em grande parte compatíveis com a especificação ARIB STD-B10.2008. Entretanto, a ARIB STD-B23:2004 é baseada nas API GEM. Esta parte da ABNT NBR 15606 é compatível com a plataforma Java DTV, o que torna necessário introduzir adaptações na API de SI. Por este motivo, uma nova API foi definida e especificada.

B.2 API informação de serviço dependente de protocolo

B.2.1 Pacote br.org.sbtvd.net

B.2.1.1 Classe *SBTVDLocator*

SBTVDLocator encapsula SBTVDURL no objeto. Esta classe estende a classe com.sun.dtv.locator.EntityLocator (ver JAVADTV 1.3:2009).

Os métodos públicos da classe *SBTVDLocator* são:

- SBTVDLocator(java.lang.String url) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator
- SBTVDLocator(java.lang.String scheme, int onid, int tsid) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado no seguinte formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado no seguinte formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado no seguinte formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado no seguinte formato.

- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int[] componenttags) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int[] componenttags, java.lang.String filePath) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, int channelid) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, java.lang.String modulename) throws javax.tv.locator.InvalidLocatorException
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, java.lang.String modulename, java.lang.String resourcename)
    - Geração do SBTVD Locator baseado num dos seguintes formatos.
- int getChannelId ()
    - Recupera o id do canal.
- int[] getComponentTags()
    - Recupera o array de component_tag.
- int getContentId()
    - Recupera o content_id.
- int getEventId()
    - Recupera o event_id.
- java.lang.String getFilePath()
    - Recupera parte do caminho do locator’s do nome do arquivo.
- java.lang.String getModuleName()
    - To acquire moduleName.
- int getOriginalNetworkId()
    - Recupera o original_network_id.

- java.lang.String getResourceName()
    - Recupera o resourceName.
- java.lang.String getScheme()
    - Recupera o scheme.
- int getServiceId()
    - Recupera service_id.
- int getTransportStreamId()
    - Recupera transport_stream_id.
- java.net.URL getURL()
    - Recupera a SBTVD URL encapsulada no objeto SBTVDLocator.

B.2.1.2 Classe *SBTVDNetworkBoundLocator*

SBTVDLocator é conectado com a rede. Esse tipo de objeto identifica de forma única certa entidade, incluindo a distribuição do sistema que transmite a entidade. Por exemplo, se um determinado serviço for transmitido através de dois tipos de redes, o serviço pode ser identificado como um serviço comum no *SBTVDLocator*. No entanto, cada serviço transmitido tem um *SBTVDNetworkBoundLocator* diferente. Esta classe implementa a interface com.sun.dtv.locator.TransportDependentLocator (ver JAVADTV 1.3:2009) e estende a classe *br.org.sbtvd.net.SBTVDLocator*.

Os métodos públicos da classe *SBTVDNetworkBoundLocator* são:

- SBTVDNetworkBoundLocator(SBTVDLocator unboundLocator, int networkId)
    - Geração do network bound locator.
- int getNetworkId()
    - Recupera o network_id.

B.2.2 Pacote br.org.sbtvd.si

B.2.2.1 Interface *DescriptorTag*

A interface *DescriptorTag d*efine as constantes que correspondem aos valores mais comuns do descriptor tag.

As constantes públicas estáticas da Classe *DescriptorTag* são:

- *static short* AUDIO_COMPONENT
    - A constante indica o valor da tag do audio component descriptor especificado na ARIB STD-B10.
- *static short* BASIC_LOCAL_EVENT
    - A constante indica o valor da tag do basic local event descriptor especificado na ARIB STD-B10.

- *static short* BOARD_INFORMATION
    - A constante indica o valor da tag do board information descriptor especificado na ARIB STD-B10.
- *static short* BOUQUET_NAME
    - A constante indica o valor da tag do bouquet name descriptor especificado na ARIB STD-B10.
- *static short* BROADCASTER_NAME
    - A constante indica o valor da tag do broadcaster name descriptor especificado na ARIB STD-B10.
- *static short* CA_CONTRACT_INFO
    - A constante indica o valor da tag do CA contractor information descriptor especificado na ARIB STD-B10.
- *static short* CA_EMM_TS
    - A constante indica o valor da tag do CA_EMM_TS descriptor especificado na ARIB STD-B10.
- *static short* CA_IDENTIFIER
    - A constante indica o valor da tag do CA identification descriptor especificado na ARIB STD-B10.
- *static short* CA_SERVICE
    - A constante indica o valor da tag do CA service descriptor especificado na ARIB STD-B10.
- *static short* CABLE_DELIVERY_SYSTEM
    - A constante indica o valor da tag do cable delivery system descriptor especificado na ARIB STD-B10.
- *static short* CAROUSEL_COMPATIBLE_COMPOSITE
    - A constante indica o valor da tag do carousel compatible composite descriptor especificado na ARIB STD-B10.
- *static short* COMPONENT
    - A constante indica o valor da tag do component descriptor especificado na ARIB STD-B10.
- *static short* COMPONENT_GROUP
    - A constante indica o valor da tag do component group descriptor especificado na ARIB STD-B10.
- *static short* CONNECTED_TRANSMISSION
    - A constante indica o valor da tag do connected transmission descriptor especificado na ARIB STD-B10.
- *static short* CONTENT
    - A constante indica o valor da tag do content descriptor especificado na ARIB STD-B10.

- *static short* CONTENT_AVAILABILITY
    - A constante indica o valor da tag do contents availability descriptor especificado na ARIB STD-B10.
- *static short* COUNTRY_AVAILABILITY
    - A constante indica o valor da tag do country receiving availability descriptor especificado na ARIB STD-B10.
- *static short* DATA_COMPONENT
    - A constante indica o valor da tag do data component descriptor especificado na ARIB STD-B10.
- *static short* DATA_CONTENTS
    - A constante indica o valor da tag do data contents descriptor especificado na ARIB STD-B10.
- *static short* DIGITAL_COPY_CONTROL
    - A constante indica o valor da tag do digital copy control descriptor especificado na ARIB STD-B10.
- *static short* DOWNLOAD_CONTENT
    - A constante indica o valor da tag do download contents descriptor especificado na ARIB STD-B10.
- *static short* EMERGENCY_INFORMATION
    - A constante indica o valor da tag do emergency information descriptor especificado na ARIB STD-B10.
- *static short* EVENT_GROUP
    - A constante indica o valor da tag do event group descriptor especificado na ARIB STD-B10.
- *static short* EXTENDED_BROADCASTER
    - A constante indica o valor da tag do extended broadcaster descriptor especificado na ARIB STD-B10.
- *static short* EXTENDED_EVENT
    - A constante indica o valor da tag do extended event descriptor especificado na ARIB STD-B10.
- *static short* HIERARCHICAL_TRANSMISSION
    - A constante indica o valor da tag do hierarchical transmission descriptor especificado na ARIB STD-B10.
- *static short* HYPER_LINK
    - A constante indica o valor da tag do hyper link descriptor especificado na ARIB STD-B10.
- *static short* LDT_LINKAGE
    - A constante indica o valor da tag do LDT linkage descriptor especificado na ARIB STD-B10.

- *static short* LINKAGE
    - A constante indica o valor da tag do linkage descriptor especificado na ARIB STD-B10.
- *static short* LOCAL_TIME_OFFSET
    - A constante indica o valor da tag do local time offset descriptor especificado na ARIB STD-B10.
- *static short* LOGO_TRANSMISSION
    - A constante indica o valor da tag do logo transmission descriptor especificado na ARIB STD-B10.
- *static short* MOSAIC
    - A constante indica o valor da tag do mosaic descriptor especificado na ARIB STD-B10.
- *static short* NETWORK_IDENTIFICATION
    - A constante indica o valor da tag do network identification descriptor especificado na ARIB STD-B10.
- *static short* NETWORK_NAME
    - A constante indica o valor da tag do network name descriptor especificado na ARIB STD-B10.
- *static short* NODE_RELATION
    - A constante indica o valor da tag do node relation descriptor especificado na ARIB STD-B10.
- *static short* NVOD_REFERENCE
    - A constante indica o valor da tag do NVOD reference service descriptor especificado na ARIB STD-B10.
- *static short* PARENTAL_RATING
    - A constante indica o valor da tag do parental rating descriptor especificado na ARIB STD-B10.
- *static short* PARTIAL_RECEPTION
    - A constante indica o valor da tag do partial reception descriptor especificado na ARIB STD-B10.
- *static short* PARTIAL_TRANSPORT_STREAM
    - A constante indica o valor da tag do partial transport stream descriptor especificado na ARIB STD-B10.
- *static short* PARTIALTS_TIME
    - A constante indica o valor da tag do partial transport stream time descriptor especificado na ARIB STD-B10.
- *static short* REFERENCE
    - A constante indica o valor da tag do reference descriptor especificado na ARIB STD-B10.

- *static short* SATELLITE_DELIVERY_SYSTEM
    - A constante indica o valor da tag do satellite delivery system descriptor especificado na ARIB STD-B10.
- *static short* SERIES
    - A constante indica o valor da tag do series descriptor especificado na ARIB STD-B10.
- *static short* SERVICE
    - A constante indica o valor da tag do service descriptor especificado na ARIB STD-B10.
- *static short* SERVICE_LIST
    - A constante indica o valor da tag do service list descriptor especificado na ARIB STD-B10.
- *static short SHORT*_EVENT
    - A constante indica o valor da tag do *short* form event descriptor especificado na ARIB STD-B10.
- *static short SHORT*_NODE_INFORMATION
    - A constante indica o valor da tag do *short* form node information descriptor especificado na ARIB STD-B10.
- *static short* SI_PARAMETER
    - A constante indica o valor da tag do SI transmission parameter descriptor especificado na ARIB STD-B10.
- *static short* SI_PRIME_TS
    - A constante indica o valor da tag do SI prime TS descriptor especificado na ARIB STD-B10.
- *static short* STC_REFERENCE
    - A constante indica o valor da tag do STC reference descriptor especificado na ARIB STD-B10.
- *static short* STREAM_IDENTIFIER
    - A constante indica o valor da tag do stream identification descriptor especificado na ARIB STD-B10.
- *static short* STUFFING
    - A constante indica o valor da tag do stuffing descriptor especificado na ARIB STD-B10.
- *static short* SYSTEM_MANAGEMENT
    - A constante indica o valor da tag do system management descriptor especificado na ARIB STD-B10.
- *static short* TARGET_AREA
    - A constante indica o valor da tag do target area descriptor especificado na ARIB STD-B10.

- *static short* TERRESTRIAL_DELIVERY_SYSTEM
    - A constante indica o valor da tag do terrestrial delivery system descriptor especificado na ARIB STD-B10.
- *static short* TIME_SHIFTED_EVENT
    - A constante indica o valor da tag do time shifted event descriptor especificado na ARIB STD-B10.
- *static short* TIME_SHIFTED_SERVICE
    - A constante indica o valor da tag do time shifted service descriptor especificado na ARIB STD-B10.
- *static short* TS_INFORMATION
    - A constante indica o valor da tag do TS information descriptor especificado na ARIB STD-B10.
- *static short* VIDEO_DECODE_CONTROL
    - A constante indica o valor da tag do video decode control descriptor especificado na ARIB STD-B10.

B.2.2.2 Interface *PMTElementaryStream*

A interface *PMTElementaryStream* indica o fluxo elementar do serviço (canal). Em cada serviço, a PMT está presente para descrever os fluxos elementares do serviço. Isso significa que o objeto montado com a interface indica um desses fluxos elementares. Cada objeto montado com a interface *PMTElementaryStream* é identificado pela combinação de original_network_id, transport_stream_id, *service_id* e *component_tag* (ou *elementary*_PID).

Os métodos públicos da classe *PMTElementaryStream* são:

- SBTVDLocator getSBTVDLocator ()
    - Recupera o SBTVDLocator que identifica o fluxo elementar.
- int getComponentTag ()
    - Recupera o component tag.
- *short* getElementaryPID ()
    - Recupera o elementary_PID.
- int getOriginalNetworkID ()
    - Recupera o original network ID.
- int getServiceID ()
    - Recupera o service ID.
- byte getStreamType ()
    - Recupera o stream type ID do fluxo elementar.
- int getTransportStreamID ()
    - Recupera o transport stream ID.

B.2.2.3 Interface *PMTService*

A interface *PMTService* indica o serviço específico a ser transmitido pelo fluxo de transporte. A informação é recuperada da PMT. Cada objeto montado com a interface *PMTService* é identificado pela combinação de *original_network_id*, *transport_stream_id*, e *service_id*.

Os métodos públicos da classe *PMTService* são:

- SBTVDLocator getSBTVDLocator()
    - Recupera o SBTVDLocator que identifica este serviço.
- int getOriginalNetworkID ()
    - Recupera o original network ID.
- int getPcrPid()
    - Recupera o PID do PCR.
- int getServiceID()
    - Recupera o service ID.
- int getTransportStreamID()
    - Recupera o transport stream ID.
- SIRequest retrievePMTElementaryStreams(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] somePMTDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Recupera da PMT a informação relevante aos fluxos elementares que compõem esse serviço.

B.2.2.4 Interface *PMTStreamType*

A interface *PMTStreamType* define as constantes que correspondem aos vários tipos de fluxo.

As constantes públicas estáticas da Classe *PMTStreamType* são:

- *static* byte DSMCC_DATA_CAROUSEL
    - A constante indica o tipo de fluxo do carrossel de dados definido pela ISO/IEC 13818-1.
- *static* byte INDEPENDENT_PES
    - A constante indica o tipo de fluxo de PES independente definido pela ISO/IEC 13818-1.
- *static* byte MPEG1_AUDIO
    - A constante indica o tipo de fluxo de áudio do MPEG1 definido pela ISO/IEC 13818-1.
- *static* byte MPEG1_VIDEO
    - A constante indica o tipo de fluxo de vídeo do MPEG1 definido pela ISO/IEC 13818-1.

- *static* byte MPEG2_AAC_AUDIO
    - A constante indica o tipo de fluxo de áudio do MPEG2AAC definido pela ISO/IEC 13818-1.
- *static* byte MPEG2_AUDIO
    - A constante indica o tipo de fluxo de áudio do MPEG2 definido pela ISO/IEC 13818-1.
- *static* byte MPEG2_VIDEO
    - A constante indica o tipo de fluxo de vídeo do MPEG2 definido pela ISO/IEC 13818-1.
- *static* byte MPEG4_VIDEO
    - A constante indica o tipo de fluxo de vídeo do MPEG4 definido pela ISO/IEC 13818-1.
- *static* byte MPEG4_AVC_VIDEO
    - A constante indica o tipo de fluxo de vídeo do H.264/MPEG-4 AVC definido pela ISO/IEC 13818-1.

B.2.2.5 Interface *SIBouquet*

A interface *SIBouquet* indica a subtabela da *Bouquet Association Table* (BAT) que descreve um *bouquet* específico (junto com o SITransportStreamBAT). Cada objeto que monta a interface *SIBouquet* é identificado pelo identificador bouquet_id. Esta interface estende br.org.sbtvd.si.SIInformation.

Os métodos públicos da classe *SIBouquet* são:

- int getBouquetID()
    - Recupera o bouquet ID.
- *short*[] getDescriptorTags()
    - Esse método define semântica adicional para o SIInformation#getDescriptorTags
- java.lang.String getName()
    - Este método retorna o nome do buquê a ser descrito no descritor de buquê.
- SBTVDLocator[] getSIServiceLocators()
    - Este método recupera a lista de SBTVDLocators para identificar o serviço que pertence ao serviço.
- SIRequest retrieveDescriptors(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semântica adicional ao primeiro protótipo do SIInformation#retrieveDescriptors
- SIRequest retrieveDescriptors(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semântica adicional ao segundo protótipo do SIInformation#retrieveDescriptors

- SIRequest retrieveSIBouquetTransportStreams(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método fornece informação relevante para o transport stream ao qual o buquê pertence.

B.2.2.6 Interface *SIBroadcaster*

A interface *SIBroadcaster* indica o provedor específico no serviço. As informações retornadas pelos métodos são adquiridas da BIT. Cada objeto que implementa a interface *SIBroadcaster* é identificado pelo broadcaster_id.

Os métodos públicos da classe *SIBroadcaster* são:

- int getBroadcasterID ()

    - Retorna o ID do provedor.

- boolean getBroadcastViewProperty ()

    - Retorna o valor da propriedade broadcaster's display.

- java.lang.String getName ()

    - Retorna o nome do provedor a ser descrito no descritor de provedor.

- int getOriginalNetworkID()

    - Retorna o original network ID.

- SBTVDLocator[] getSIServiceLocators()

    - Este método recupera a lista de SBTVDLocator para identificar o serviço que pertence ao provedor.

- SIRequest retrieveOriginalNetworkDescriptors(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método recupera todos os descritores que são transmitidos no primeiro laço da BIT.

- SIRequest retrieveOriginalNetworkDescriptors(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método recupera o conjunto de descritores que são transmitidos no primeiro laço da BIT.

B.2.2.7 Interface *SIEvent*

A interface *SIEvent* indica um programa específico no serviço. Cada objeto que implementa a interface SIEvent é identificado pela combinação do *original_network_id*, *transport_stream_id*, *service_id* e *event_id*. Se o valor de retorno do método for adquirido da forma simples do descritor de evento e mais de um descritor estiver presente, o algoritmo seguinte deve ser usado. No caso da linguagem retornada por javax.tv.service.SIManager#getPreferredLanguage ser usada no descritor simples de evento, o valor é retornado do descritor. Caso contrário, ele depende do estado da montagem que deve ser usado além dos descritores simples de evento disponíveis. Esta interface estende *br.org.sbtvd.si.SIInformation*.

Os métodos públicos da classe *SIEvent* são:

- SBTVDLocator getSBTVDLocator()
    - Recupera o SBTVDLocator que identifica o programa.
- java.lang.String[] getAudioComponentDescriptions()
    - Este método retorna a descrição em texto do fluxo elementar de áudio relevante ao programa.
- java.lang.String[] getComponentDescriptions()
    - Este método retorna a descrição em texto do fluxo elementar relevante ao programa.
- byte[] getContentNibbles()
    - Este método retorna o gênero do programa.
- java.lang.String[] getDataContentDescriptions()
    - Este método retorna a descrição em texto relevante ao *data broadcasting program.*
- long getDuration()
    - Este método recupera a duração do programa.
- int getEventID()
    - Este método recupera o ID do evento.
- SIExEventInformation[] getExEventInformations()
    - Este método retorna informações detalhadas relevantes ao programa.
- boolean getFreeCAMode ()
    - Este método recupera o valor de embaralhamento do programa.
- byte[] getLevel1ContentNibbles()
    - Este método retorna o primeiro passo da classificação do conteúdo do programa.
- java.lang.String getName()
    - Este método retorna o nome do programa.
- int getOriginalNetworkID()
    - Este método recupera o original network ID.
- byte getRunningStatus()
    - Este método recupera o estado de execução do programa.
- java.lang.String getSeriesName()
    - Este método retorna o nome da série relevante ao programa.

- int getServiceID()
    - Este método recupera o ID do serviço.
- java.lang.String get*Short*Description()
    - Este método retorna a descrição do programa.
- java.util.Date getStartTime()
    - Este método recupera a hora de início do programa.
- int getTransportStreamID()
    - Este método recupera o transport stream ID.
- byte[] getUserNibbles()
    - Este método retorna o gênero relevante ao programa.
- SIRequest retrieveSIService(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera o SIService que indica o serviço. O serviço é aquele a qual o programa indicado por SIEvent pertence.

B.2.2.8 Interface *SIInformation*

A interface *SIInformation* é uma coleção de funções comuns a *SIBouquet*, *SIBroadcaster*, *SINetwork*, *SITransportStream*, *SIService*, *PMTService*, *SIEvent*, *SITime* e *PMTElementaryStream*.

As constantes públicas estáticas da classe *SIInformation* são:

- *static short* FROM_CACHE_ONLY
    - A constante é utilizada para o parâmetro do modo de aquisição dos métodos de aquisição.
- *static short* FROM_CACHE_OR_STREAM
    - A constante é utilizada para o parâmetro do modo de aquisição dos métodos de aquisição.
- *static short* FROM_STREAM_ONLY
    - A constante é utilizada para o parâmetro do modo de aquisição dos métodos de aquisição.

Os métodos públicos da classe *SIInformation* são:

- boolean fromActual()
    - Se a informação contida no objeto que implementa essa interface for selecionada da tabela "actual" ou da tabela que nao distingue "actual" ou "not actual", "true" é retornado.
- com.sun.dtv.transport.TransportStream getDataSource()
    - Este método retorna o objeto com.sun.dtv.transport.TransportStream que foi selecionado da informação contida no objeto que implementa esta interface.

- *short*[] getDescriptorTags()
    - Este método recupera os valores das tags de todos os descritores que fazem parte da versão corrente deste objeto.
- SIDatabase getSIDatabase()
    - Este método retorna a raiz da estrutura hierárquica à qual pertence o objeto que implementa esta interface.
- java.util.Date getUpdateTime()
    - Este método retorna os últimos dia e hora de atualização desta informação inclusa no objeto que implementa esta interface.
- SIRequest retrieveDescriptors (*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método retorna todos os descritores na ordem de envio.
- SIRequest retrieveDescriptors (*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera alguns descritores.

B.2.2.9 Interface *SIIterator*

O objeto que implementa a interface SIIterator pode acessar o conteúdo através da coleção de objetos SI. A fim de manter a consistência da coleção, alguns acessos ao fluxo não são iniciados, dependendo do acesso ao conteúdo.

O método público da classe SIIterator é:

- int numberOfRemainingObjects()
    - O número de objetos mantidos no interador.

B.2.2.10 Interface *SIMonitoringListener*

A interface SImonitoringListener é implementada pela classe da aplicação, a fim de receber as mudanças de monitoramento do objeto SI.

O método público da classe *SIMonitoringListener* é:

- void postMonitoringEvent(SIMonitoringEvent anEvent)
    - Este método é chamado pela API do SI, a fim de informar o ouvinte do evento.

B.2.2.11 Interface *SIMonitoringType*

A interface *SIMonitoringType* define as constantes que correspondem a cada tipo de informação SI no *SIMonitoringEvent*.

As constantes públicas estáticas da classe *SIMonitoringType* são:

- *static* byte BOUQUET
    - Constante do objeto SIInformation que indica o buquê.

- *static* byte BROADCASTER
    - Constante do objeto SIInformation que indica o provedor.
- *static* byte NETWORK
    - Constante do objeto SIInformation que indica a rede.
- *static* byte PMT_SERVICE
    - Constante do objeto SIInformation que indica o PMT service.
- *static* byte PRESENT_FOLLOWING_EVENT
    - Constante do objeto SIInformation que indica a EIT [Present/Following].
- *static* byte SCHEDULED_EVENT
    - Constante do objeto SIInformation que indica a EIT [Schedule].
- *static* byte SERVICE
    - Constante do objeto SIInformation que indica o serviço.

B.2.2.12 Interface *SINetwork*

A interface *SINetwork* indica a subtabela da *Network Information Table* (NIT) que descreve uma rede específica (junto com a *SITransportStreamNIT*). Cada objeto que implementa a interface *SINetwork* é identificado pelo *network_id*.

Os métodos públicos da classe *SINetwork* são:

- *short*[] getDescriptorTags()
    - Este método define semânticas adicionais ao método SIInformation#getDescriptorTags.
- java.lang.String getName()
    - Este método retorna o nome da rede que é descrito no Network Name Descriptor
- int getNetworkID()
    - Recupera o network ID desta rede.
- SIRequest retrieveDescriptors(*short*retrieveMode,java.lang.ObjectappData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semânticas adicionais ao primeiro protótipo do método SIInformation#retrieveDescriptors.
- SIRequest retrieveDescriptors(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semânticas adicionais ao segundo protótipo do método SIInformation#retrieveDescriptors.
- SIRequest retrieveSITransportStreams(*short*retrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera a informação sobre o transport stream a ser transmitido através da rede.

B.2.2.13 Interface *SIRetrievalListener*

A interface *SIRetrievalListener* deve ser implementada para receber um evento SI.

O método público da classe *SIRetrievalListener* é:

- void postRetrievalEvent(SIRetrievalEvent event)
    - Este método é chamado a partir da SI API implementada de forma a notificar a conclusão do pedido do ouvinte.

B.2.2.14 Interface *SIRunningStatus*

A interface *SIRunningStatus* define a constante que corresponde ao valor do estado de execução do serviço e do evento.

As constantes públicas estáticas da classe *SIRunningStatus* são:

- *static* NOT_RUNNING
    - byte constante, tal como definido na ARIB STD-B10, indica que o estado é "not running".
- *static* PAUSING
    - byte constante, tal como definido na ARIB STD-B10, indica que o estado é executando "pausing".
- *static* RUNNING
    - byte constante, tal como definido na ARIB STD-B10, indica que o estado é "running".
- *static* STARTS_IN_A_FEW_SECONDS
    - byte constante, tal como definido na ARIB STD-B10, indica que o estado é “pronto para iniciar em alguns segundos”.
- *static* UNDEFINED
    - byte constante, tal como definido na ARIB STD-B10, indica que o estado é “indefinido”.

B.2.2.15 Interface *SIService*

A interface *SIService* indica um serviço específico que é transmitido por um dos fluxos de transporte. As informações obtidas através do método desta interface são adquiridas a partir da SDT. Cada objeto montado com a interface *SIService* é identificado pela combinação dos seguintes ID:

Original network ID, Transport stream ID, Service ID

Os métodos públicos da classe *SIService* são:

- SBTVDLocator getSBTVDLocator()
    - Este método adquire o SBTVDLocator para identificar este serviço.
- boolean getEITPresentFollowingFlag()
    - Este método retorna o valor da EIT_present_following_flag.

- boolean getEITScheduleFlag()
    - Este método retorna o valor da EIT_schedule_flag.
- int getEITUserDefinedFlag()
    - Este método retorna o valor das EIT_user_defined_flags.
- boolean getFreeCAMode()
    - Este método retorna o valor do free_CA_mode.
- java.lang.String getName()
    - Este método retorna o nome que indica o serviço incluído no service descriptor.
- int getOriginalNetworkID()
    - Este método recupera o original network ID.
- java.lang.String getProviderName()
    - Este método retorna o nome do prestador de serviços incluídos no service descriptor.
- byte getRunningStatus()
    - Este método adquire o estado de execução deste serviço.
- int getServiceID()
    - Este método adquire o serviço ID.
- *short* getSIServiceType()
    - Este método adquire o tipo de serviço.
- int getTransportStreamID()
    - Este método adquire o transporte stream ID.
- SIRequest retrieveFollowingSIEvent(*short*retrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o programa seguinte de EIT [Presente / seguinte].
- SIRequest retrievePMTService(*short*retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método PMTService adquire a informações relevantes para este serviço.
- SIRequest retrievePresentSIEvent(*short* retrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, *short*[] someDescriptorTags). throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o programa presente de EIT [Presente / seguinte].

- SIRequest retrieveScheduledSIEvents(*short*retrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, *short*[] someDescriptorTags, java.util.Date startTime, java.util.Date endTime) throws br.org.sbtvd.si.SIIllegalArgumentException, br.org.sbtvd.si. SIInvalidPeriodException

    - Este método adquire as informações relevantes para o programa previsto no período designado de EIT [Calendário].

B.2.2.16 Interface *SIServiceType*

Esta API é responsável pelo acesso a informações relativas a definição de *ServiceType*.

As constantes públicas estáticas da classe *SIServiceType* são:

- *static* BOOKMARK_LIST

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "marcador lista de dados".

- *static* DATA

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "serviço de dados".

- *static* DATA_EXCLUSIVE_FOR_ACCUMULATION

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "serviço de dados exclusivos de acumulação".

- *static* DATA_FOR_ACCUMULATION_IN_ADVANCE

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "serviço de dados exclusivos para a acumulação de antecedência".

- *static* DIGITAL_AUDIO

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "áudio digital".

- *static* DIGITAL_TELEVISION

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é do tipo "TV digital".

- *static* ENGINEERING_DOWNLOAD

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "download engineering".

- *static* PROMOTION_DATA

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "promoção dados".

- *static* PROMOTION_SOUND

    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é “promoção som".

- *static* PROMOTION_VIDEO
    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "promoção vídeo".
- *static* SPECIAL_AUDIO
    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um " áudio especial".
- *static* SPECIAL_DATA
    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um " dado especial".
- *static* SPECIAL_VIDEO
    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é "um vídeo especial".
- *static* UNKNOWN
    - Constante, tipo *short*, tal como definido na ARIB STD-B10, indica que o tipo de serviço é um "desconhecido".

B.2.2.17 Interface *SITime*

A interface *SITime* provê acesso as informações da *Time and Date Table* (TDT). Se o objeto indicar a TDT, os métodos retrieveDescriptors e getDescriptorTags se comportam como é especificado quando não há descrição, pois a TDT não possui descritores.

Esta interface indica hora e data obtidas da tabela (TDT). Se o objeto indicar TDT, o comportamento dos retrieveDescriptors e das getDescriptorTags é semelhante ao caso onde não existe qualquer descritor (porque não é encontrado Descritor TDT).

O método público da classe *SITime* é:

- java.util.Date getTime()
    - Este método adquire o tempo codificado na TDT ou na TOT.

B.2.2.18 Interface *SITransportStream*

A interface *SITransportStream* é a interface base usada para indicar as informações relevantes para o fluxo de transporte.

O método que recupera o fluxo do transporte na classe *SIDatabase* e na interface *SINetwork* retorna o objeto montado com a interface SITransportStreamNIT referente à NIT. O método que recupera o fluxo de transporte na interface *SIBouquet* retorna o objeto montado com a interface *SITransportStreamBAT* referente à BAT.

Os métodos públicos da classe *SITransportStream* são:

- SBTVDLocator getSBTVDLocator()
    - Este método adquire o SBTVDLocator que identifica o fluxo de transporte.

- int getOriginalNetworkID()
    - Este método adquire o original network ID.
- int getTransportStreamID()
    - Este método adquire transporte stream ID.
- SIRequest retrieveSIServices(*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o serviço a ser transmitida pelo fluxo.

B.2.2.19 Interface *SITransportStreamBAT*

A interface *SITransportStreamBAT* indica informações do fluxo de transporte recuperadas da BAT. Todos os métodos que acessam descritores retornam informações de descritores adquiridos da BAT. O método que recupera o fluxo de transporte na *SIBouquet* retorna um objeto que implementa esta interface.

O método público da classe *SITransportStreamBAT* é:

- int getBouquetID()
    - O método adquire o ID do bouquet ao qual pertence este transporte stream.

B.2.2.20 Interface *SITransportStreamNIT*

A interface *SITransportStreamNIT* indica informações do fluxo de transporte adquiridas da NIT. Todos os métodos que acessam descritores retornam informações de descritores adquiridos da NIT. O método que recupera o fluxo de transporte no *SIDatabase* ou *SINetwork* retorna um objeto que implementa essa interface.

O método público da classe *SITransportStreamNIT* é:

- int getNetworkID()
    - Este método adquire o ID da rede a que este fluxo de transporte pertence

B.2.2.21 Classe *Descriptor*

A classe *Descriptor* indica os descritores da subtabela.

Os métodos públicos da classe *Descriptor* são:

- byte getByteAt(int index) throws java.lang.IndexOutOfBoundsException
    - Este método obtém o valor de um byte da seção de dados do descritor.
- byte[] getContent()
    - Este método adquire uma cópia da seção de dados do descrito (bytes que estão depois do byte que indica o tamanho do descritor).
- *Short* getContentLength()
    - Este método retorna o comprimento da seção de dados indicado no campo “descriptor length”.
- *short* getTag()
    - Este método adquire a tag do descritor.

B.2.2.22 Classe *SIDatabase*

A classe *SIDatabase* indica a raiz da estrutura hierárquica das informações do SI. Existe um SIDatabase para cada interface de rede. Assim, existe apenas uma *SIDatabase*, se existir apenas uma interface de rede.

As constantes públicas estáticas da classe *SIDatabase* são:

- *static* int RETRIEVE_ALL_INFORMATIONS
- *static* int RETRIEVE_CURRENT_SELECTED

Os métodos públicos da classe *SIDatabase* são:

- void addBouquetMonitoringListener(SIMonitoringListener listener, int bouquetId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações de bouquet.
- void addBroadcasterMonitoringListener(SIMonitoringListener listener, int broadcasterId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações de broadcaster.
- void addEventPresentFollowingMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações da EIT [Presente / Seguindo].
- void addEventScheduleMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId, java.util.Date startTime, java.util.Date endTime) throws br.org.sbtvd.si.SIIllegalArgumentException, br.org.sbtvd.si. SIInvalidPeriodException
    - Este método inicializa o acompanhamento das informações da EIT [schedule].
- void addNetworkMonitoringListener(SIMonitoringListener listener, int networkId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações da rede.
- void addPMTServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações da PMT relevantes para o serviço.
- void addServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicializa o acompanhamento das informações da SDT relevantes para o serviço.
- *static* SIDatabase[] getSIDatabase()
    - Este método retorna o objeto SIDatabase(para cada interface de rede).
- void removeBouquetMonitoringListener(SIMonitoringListener listener, int bouquetId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do ouvinte do evento de monitoramento das informações do bouquet.

- void removeBroadcasterMonitoringListener(SIMonitoringListener listener,int broadcasterId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do ouvinte do evento de monitoramento das informações do broadcaster.
- void removeEventPresentFollowingMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do ouvinte do evento de monitoramento das informações da EIT [presente/seguinte].
- void removeEventScheduleMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Correspondente à totalidade agendada, esse método elimina o registro do evento de monitoramento da EIT [schedule].
- void removeNetworkMonitoringListener(SIMonitoringListener listener, int networkId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do evento de monitoramento da rede.
- void removePMTServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do ouvinte de eventos de monitoramento das informações da PMT relevantes para o serviço.
- void removeServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método remove o registro do ouvinte de monitoramento das informações relevantes ao serviço.
- SIRequest retrieveActualSINetwork(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para a rede atual.
- SIRequest retrieveActualSIServices(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o serviço.
- SIRequest retrieveActualSITransportStream (*short* retrieveMode, java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags). throws br.org.sbtvd.si.SIIllegalArgumentException

Este método adquire as informações relevantes para o fluxo.

- SIRequest retrievePMTElementaryStreams(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire informações sobre o fluxo elementar da PMT relevantes para o serviço componente do fluxo deste SIDatabase.
- SIRequest retrievePMTElementaryStreams(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener,int serviceId,int componentTag, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire informações sobre o fluxo elementar da PMT relevantes para o serviço componente do fluxo deste SIDatabase.

- SIRequest retrievePMTService(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações da PMT relevantes para o serviço.
- SIRequest retrievePMTServices(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int serviceId, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações da PMT relevantes para o serviço a partir do fluxo de transporte deste SIDatabase.
- SIRequest retrieveSIBouquets(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int bouquetId, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o bouquet.
- SIRequest retrieveSIBroadcaster(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int originalNetworkId, int broadcasterId, *short* [] some DescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o broadcaster.
- SIRequest retrieveSIBroadcasters(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int originalNetworkId, *short* [] some DescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o broadcaster especificado pela originalNetworkId.
- SIRequest retrieveSINetworks(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int networkId, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para a rede.
- SIRequest retrieveSIService(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o serviço.
- SIRequest retrieveSIServices(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener,int originalNetworkId,int transportStreamId, int serviceId, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire as informações relevantes para o serviço.
- SIRequest retrieveSITimeFromTDT(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire da Time Date Table (TDT) as informações sobre tempo.
- SIRequest retrieveSITimeFromTOT(*short* retrieveMode,java.lang.Object appData, SIRetrievalListener listener, *short*[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquire da Time Offset Table (TOT) as informações sobre tempo.

B.2.2.23 Classe *SIExEventInformation*

A interface *SIExEventInformation* indica os itens da descrição e o nome dos detalhes de um programa específico. A informação é adquirida dos descritores de evento no formato extenso.

Os métodos públicos da classe *SIExEventInformation* são:

- java.lang.String getDescription()
    - Este método adquire a descrição do item.
- java.lang.String getName()
    - Este método adquire o nome do item.

B.2.2.24 Classe *SILackOfResourcesEvent*

O evento da classe *SILackOfResourcesEvent* é notificado quando os recursos necessários para adquirir os dados solicitados em uma requisição não estão disponíveis no SI. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent

O método público da classe *SILackOfResourcesEvent* é:

- SILackOfResourcesEvent(java.lang.Object appData, SIRequest request)
    - Este é o método construtor deste evento.

B.2.2.25 Classe *SIMonitoringEvent*

O objeto da classe *SIMonitoringEvent* é transmitido para que o objeto ouvinte notifique a aplicação da mudança da informação monitorada. Esta classe estende java.util.EventObject.

Os métodos públicos da classe *SIMonitoringEvent* são:

- SIMonitoringEvent(SIDatabase source,byte objectType,int networkId,int bouquetId, int originalNetworkId, int transportStreamId, int broadcasterId, int serviceId, java.util.Date startTime, java.util.Date endTime)
    - Constructor of event object
- int getBouquetID()
    - Este método retorna o bouquet ID
- int getBroadcasterID()
    - Este método retorna o broadcaster ID do broadcaster.
- java.util.Date getEndTime()
    - Este método retorna o fim da programação quando a informação dos eventos é modificada.
- int getNetworkID()
    - Este método retorna a rede ID da rede.
- int getOriginalNetworkID()
    - Este método retorna o ID rede original do objeto de SIInformation.

- int getServiceID()
    - Este método retorna o ID do serviço do objeto de SIInformation.
- byte getSIInformationType()
    - Este método adquire o tipo do SIInformation na mudança da informação.
- java.lang.Object getSource()
    - Este método adquire a instância do SIDatabase a ser enviada ao evento.
- java.util.Date getStartTime()
    - Este método retorna o início da programação quando a informação dos eventos é modificada.
- int getTransportStreamID()
    - Este método retorna o transport stream ID do objeto SIInformation.

B.2.2.26 Classe *SINotInCacheEvent*

Quando o pedido da aquisição do SI na modalidade de FROM_CACHE_ONLY for executado e os dados pedidos não existirem na *cache*, este evento é notificado como uma resposta. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent.

O método público da classe *SINotInCacheEvent* é:

SINotInCacheEvent(java.lang.Object appData, SIRequest request)

- Construtor do evento

B.2.2.27 Classe *SIObjectNotInTableEvent*

O evento da classe *SIObjectNotInTableEvent* é notificado quando a tabela SI que tem a informação sobre a localização do objeto requisitado é recuperada mas não contém o referido objeto. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent.

O método público da classe *SIObjectNotInTableEvent* é:

SIObjectNotInTableEvent(java.lang.Object appData, SIRequest request)

- Construtor do evento.

B.2.2.28 Classe *SIRequest*

A instância do objeto da classe *SIRequest* indica o pedido da aquisição da aplicação. A aplicação pode cancelar o pedido usando este objeto.

Os métodos públicos da classe *SIRequest* são:

- boolean cancelRequest()
    - Este método cancela o pedido da aquisição.
- boolean isAvailableInCache()
    - Este método retorna a disponibilidade da informação se estiver retornando do fluxo ou da *cache*.

B.2.2.29 Classe *SIRequestCancelledEvent*

O evento da classe *SIRequestCancelledEvent* é lançado como uma resposta quando uma requisição é cancelada através do método de SIRequest.cancelRequest. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent.

O método público da classe *SIRequestCancelledEvent* é:

SIRequestCancelledEvent(java.lang.Object appData, SIRequest request)

- Constructor

B.2.2.30 Classe *SIRetrievalEvent*

A classe *SIRetrievalEvent* é a classe básica para o evento de conclusão do pedido da aquisição do SI. Somente um evento é retornado para um pedido da aquisição do SI. Esta classe estende java.util.EventObject.

Os métodos públicos da classe *SIRetrievalEvent* são:

- SIRetrievalEvent(java.lang.Object appData, SIRequest request)
  - Construtor do evento
- java.lang.Object getAppData()
  - Este método retorna os dados da aplicação passados ao método de aquisição.
- java.lang.Object getSource()
  - Este método retorna um objeto SIRequest do evento de origem.

B.2.2.31 Classe *SISuccessfulRetrieveEvent*

O evento da classe *SISuccessfulRetrieveEvent* é enviado como uma resposta quando o pedido é terminado normalmente. O resultado pode ser adquirido usando o método getResult. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent.

Os métodos públicos da classe *SISuccessfulRetrieveEvent* são:

- SISuccessfulRetrieveEvent(java.lang.Object appData, SIRequest request, SIIterator result)
  - Construtor.
- SIIterator getResult()
  - Este método retorna o objeto de SIIterator que inclui os dados pedidos.

B.2.2.32 Classe *SITableNotFoundEvent*

O evento da classe *SITableFoundEvent* é enviado como uma resposta quando a tabela SI que deve conter a informação requerida não foi encontrada. Uma das razões pode ser o fato dela não estar sendo transmitida no fluxo conectado ao SI database. Esta classe estende a classe br.org.sbtvd.si.SIRetrievalEvent.

O método público da classe *SITableNotFoundEvent* é:

SITableNotFoundEvent(java.lang.Object appData, SIRequest request)

- Construtor

B.2.2.33 Classe *SITableUpdatedEvent*

O evento da classe *SITableUpdateEvent* é lançado como uma resposta quando a tabela, que transmite a informação sobre o objeto alvo da requisição do SI é atualizada e a informação do descritor em conformidade com objeto antigo não está disponível. Neste caso, a aplicação deve inicialmente atualizar o objeto de SIInformation. Então a informação sobre o descritor deve ser pedida outra vez. Esta classe estende br.org.sbtvd.si.SIRetrievalEvent.

O método público da classe *SITableUpdatedEvent* é:

- SITableUpdatedEvent(java.lang.Object appData, SIRequest request)
    - Construtor Padrão.

B.2.2.34 Classe *SIUtil*

A classe *SIUtil* inclui função de utilidade relevante ao SI.

O método público da classe *SIUtil* é:

- *static* java.lang.String convertSIStringToJavaString(byte[] sbtvdSIText, int offset, int length) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método converte o string de texto codificado para o objeto string de Java baseado na ARIB STD-B10:2008 Parte 2, Apêndice A. Esse método herdado de Class java.lang.Object.

B.2.2.35 Classe *SIException()*

A classe *SIException*() é a base da hierarquia de exceções da SI. Esta classe estende java.lang.Exception

O método público da classe *SIException*() é:

- SIException()
    - Construtor padrão de exceção.
- SIException(String message)
    - Construtor com parâmetro (String) para indicar o motivo da exceção.

B.2.2.36 Classe *SIIllegalArgumentException()*

A classe *SIIllegalArgumentException*() é lançada quando mais de um argumento impróprio é passado (por exemplo, valores numéricos fora do espaço). Esta classe estende br.sbtvd.si.SIException

O método público da classe *SIIllegalArgumentException*() é:

- SIIllegalArgumentException() ()
    - Construtor padrão de exceção.
- SIIllegalArgumentException (String message)
    - Construtor com parâmetro (*String*) para indicar o motivo da exceção

B.2.2.37 Classe *SIInvalidPeriodException*

A classe *SIInvalidPeriodException* acontece quando a extensão de tempo especificada é imprópria, por exemplo, o tempo de inicialização está mais atrasado. Esta classe estende br.org.sbtvd.si.SIException

Os métodos públicos da classe *SIInvalidPeriodException* são:

- SIInvalidPeriodException()
    - Construtor padrão de exceção.
- SIInvalidPeriodException(java.lang.String reason)

Construtor de exceção que tem uma razão para ser especificado.

Anexo C
(normativo)

Especificação API de extensão para sintonia – Pacote *br.org.sbtvd.net.tuning*

C.1 Classe *ChannelManager*

A classe *ChannelManager* especifica um objeto responsável pela atividade de *zapping* (troca) de canais sintonizáveis através da interface de rede (terrestre, cabo, satélite, IPTV) existente no receptor de televisão digital.

Os métodos públicos são os seguintes:

- *static* ChannelManager getInstance ()
    - Retorna um objeto (instância) *ChannelManager*.
- int getNumberofChannels ()
    - Retorna o número de canais encontrados em todas as interfaces de rede disponíveis no sistema.
- Channel [] getChannels ()
    - Retorna o número de canais encontrados em todas as interfaces de rede disponíveis no sistema.
- void tuneChannel (Channel ch, com.sun.dtv.tuner.TunerListener lis) throws *com.sun.dtv.tuner.TuningException*
    - Sintoniza assincronamente o canal fornecido pelo parâmetro inteiro *num*. Este método lança uma exceção do tipo *com.sun.dtv.tuner.TuningException* no caso de falha na sintonia.
- void tuneNextChannel (com.sun.dtv.tuner.TunerListener lis) throws *com.sun.dtv.tuner.TuningException*
    - Sintoniza assincronamente o próximo canal da tabela *TransportStream* gerada durante a varredura. Este método lança uma exceção do tipo *com.sun.dtv.tuner.TuningException* no caso de falha na sintonia.
- void tunePreviousChannel (*com.sun.dtv.tuner.TunerListener* lis) throws *com.sun.dtv.tuner.TuningException*
    - Sintoniza assincronamente o canal anterior da tabela *TransportStream* gerada durante a varredura. Este método lança uma exceção do tipo *com.sun.dtv.tuner.TuningException* no caso de falha na sintonia.

C.2 Classe *Channel*

A classe *Channel* representa um objeto que contém os dados dos canais detectados durante a varredura. A partir das informações dessa classe, é possível, por exemplo, sintonizar um canal através do seu número virtual (*remote control key id*).

Os métodos públicos são os seguintes:

- com.sun.dtv.transport.TransportStream getTransportStream ()
  - Retorna o objeto que contém o canal
- String getNetworkName ()
  - Retorna a descrição da rede na qual o canal se encontra no momento.
- String getTransportStreamName ()
  - Retorna a descrição do fluxo de transporte no qual o canal se encontra no momento.
- int getRemoteControlKeyId()
  - Retorna o número virtual do canal

Anexo D
(normativo)

Especificação API de ponte NCL

D.1 Considerações gerais

Os pacotes br.org.sbtvd.bridge e br.org.sbtvd.bridge.ncl contêm o conjunto de classes disponíveis para a ponte entre os aplicativos escritos nas linguagens NCL e Java, em ambiente Ginga. As funções disponíveis nas classes que são descritas em D.2.1 podem ser utililzadas para o desenvolvimento de aplicações Ginga-J, incluindo aplicações Ginga-NCL, bem como o desenvolvimento de aplicações Ginga-NCL incluindo Xlets Java.

No primeiro caso, a API de ponte NCL Ginga-J torna possível a apresentação e manipulação de um documento NCL em uma aplicação Java, através da classe NCLPlayer, mantendo o documento original preservado durante todo o processo de exibição. As classes que agregam tais funcionalidades são disponibilizadas pelo pacote br.org.sbtvd.bridge.

No segundo caso, a API de ponte viabiliza que um documento NCL também seja capaz de incluir Xlets Ginga-J como um de seus nós de mídia (elemento <media>). Um elemento <media> contendo um código Java pode definir âncoras (através de elementos <area>), e atributos (através de elementos <property>). As transições aplicadas ao Xlet invocarão os métodos da interface *javax.microedition.xlet.Xlet* (ver PBP 1.1:2008), que representam as transições de sua máquina de estados. As transições aplicadas aos nós e âncoras devem gerar eventos da classe NCLEvent, que encapsulam a transição e o identificador do nó ou âncora em questão. Em D.2.2 são apresentadas as classes que são responsáveis pela comunicação do Ginga-NCL com o ambiente Ginga-J, quando um documento NCL inclui uma aplicação Ginga-J. Tais classes compõem o pacote br.org.sbtvd.bridge.ncl.

Informações complementares podem ser obtidas na ABNT NBR 15606-2:2007, 10.3.4.3 e 11.2.

D.2 API de ponte NCL

D.2.1 Pacote br.org.sbtvd.bridge

D.2.1.1 Classe NCLPlayer

A classe NCLPlayer é uma classe que representa um exibidor para um documento NCL, sendo um componente gráfico que estende *java.awt.Component*. O tratamento dos eventos de entrada (de teclas, por exemplo) será feito pelo exibidor NCL (os eventos são repassados pelo ambiente Ginga-J para o Ginga-NCL) enquanto o NCLPlayer possuir o foco de interação, dentre os componentes gráficos utilizados no Xlet em questão.

As constantes públicas estáticas da classe NCLPlayer são:

- *static* int PLAYING
    - Identificação para documento em execução.
- *static* int PAUSED
    - Identificação para documento em pausado.
- *static* int STOPPED
    - Identificação para documento em parado.

Os métodos públicos da classe NCLPlayer são:

- NCLPlayer(java.net.URL documentURL)
    - Método construtor para NCLPlayer, que recebe como parâmetro uma instância da classe java.net.URL como localizador do documento.
- void addNCLPlayerEventListener(NCLPlayerEventListener listener, long nclPlayerEventPlayerMask)
    - Este método registra um NCLPlayerEventListener para receber todos os NCLPlayerEvents distribuídos pela máquina associada ao nó que se vincula ao valor *long eventMask* fornecido.
- void removeNCLPlayerEventListener(NCLPlayerEventListener listener)
    - Remove um NCLPlayerEventListener da classe de recepção dos NCLPlayerEvents distribuídos.
- NCLPlayerEventListener[] getNCLPlayerEventListeners()
    - Retorna uma lista de todos os NCLPlayerEventListeners registrados neste *NCLPlayer*. Os objetos ouvintes adicionados várias vezes aparecem apenas uma vez na lista retornada.
- NCLPlayerEventListener[] getNCLPlayerEventListeners(long nclPlayerEventMask)
    - Retorna uma lista de todos os NCLPlayerEventListeners registrados neste *NCLPlayer* que ouvem a todos os tipos de eventos indicados no valor *long eventMask*. Os objetos ouvintes adicionados várias vezes aparecem apenas uma vez na lista retornada.
- java.net.URL getDocumentURL()
    - Retorna um objeto da classe *java.net.URL* que é o localizador do documento NCL sendo manipulado pelo objeto *NCLPlayer* em questão.
- java.lang.String getPropertyValue(java.lang.String propertyId)
    - Retorna uma instância de *String* com o valor da propriedade definida pelo parâmetro *String propertyId*.
- int getStatus()
    - Retorna um valor inteiro representando o estado do objeto *NCLPlayer* (*PLAYING* – em execução, *PAUSED* – pausado, *STOPPED* – parado).
- void setDocument(java.net.URL documentURL)
    - Define o documento NCL a ser manipulado pelo *NCLPlayer*, recebendo um objeto da classe *java.net.URL* como identificador do documento. O novo estado de execução do *NCLPlayer* deverá ser definido como parado (*STOPPED*).
- boolean startDocument(java.lang.String interfaceId)
    - Começa a reprodução de um documento NCL começando a apresentação a partir da interface de documento especificada pela instância de *String interfaceId*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean stopDocument()
    - Pára a apresentação de um documento NCL. Todos os eventos do documento que estão ocorrendo devem obrigatoriamente ser parados. Retorna *true* em caso de sucesso, e *false* em caso contrário.

- boolean pauseDocument()
    - Pausa a apresentação de um documento NCL. Todos os eventos do documento que estão ocorrendo devem obrigatoriamente ser pausados. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean resumeDocument()
    - Retoma a apresentação de um documento NCL. Todos os eventos de documentos que foram previamente pausados pelo método *pauseDocument()* devem obrigatoriamente ser retomados. Retorna *true* em caso de sucesso e *false* em caso contrário.
- NCLEdit getNCLEdit()
    - Retorna uma instância da classe NCLEdit, que oferece funcionalidades de edição do documento NCL em tempo de exibição.

D.2.1.2 Classe NCLPlayerEvent

A classe NCLPlayerEvent estende java.util.EventObject e representa o evento raiz para todos os eventos NCL gerados pela aplicação NCL encapsulada por uma instância da classe em questão. As máscaras de evento definidas nesta classe são utilizadas para especificar quais tipos de eventos um NCLPlayerEventListener deve ouvir. Informações adicionais podem ser obtidas na ABNT NBR 15606-2:2007, 10.3.4.3.

As constantes públicas estáticas da classe NCLPlayerEvent são:

- *public static final int* PRESENTATION_START = 1;
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o início da reprodução (*start*) de um nó ou âncora.
- *public static final int* PRESENTATION_STOP = 2;
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o término da reprodução (*stop*) de um nó ou âncora.
- *public static final int* PRESENTATION_ABORT = 4
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o abortamento da reprodução (*abort*) de um nó ou âncora.
- *public static final int* PRESENTATION_PAUSE = 8;
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi a pausa da reprodução (*pause*) de um nó ou âncora.
- *public static final int* PRESENTATION_RESUME = 16;
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi a retomada da reprodução (*resume*) de um nó ou âncora.
- *public static final int* ATTRIBUTION_SET = 32;
    - Máscara de evento para eventos de atribuição (tipo *attribution*) cuja ação (campo *action*) foi a definição (*set*) de um parâmetro de um nó ou âncora.
- protected int id
    - Identificação do evento.

Os métodos públicos da classe NCLPlayerEvent são:

- NCLPlayerEvent(Object source, int id, String value)
    - Método construtor para NCLPlayerEvent, que recebe como parâmetro uma referência (*source)* do objeto que originou o evento, um inteiro (*id)* que identifica o evento e um identificador (value) do nó ou âncora relacionado ao evento.
- int getID()
    - Retorna o tipo de evento.
- String getValue()
    - Retorna o identificador do nó ou âncora relacionado ao evento.

D.2.1.3 Interface *NCLPlayerEventListener*

A interface *Listener* deve ser implementada por quem deseja receber notificação de eventos distribuídos para objetos que são elementos da classe *NCLEvent*. Estende a interface java.util.EventListener.

A aplicação que se interessa em monitorar os eventos NCL de um *NCLPlayer* implementa esta interface registrando-se com o *NCLPlayer* através do método *NCLPlayer.addNCLPlayerEventListener()*. Quando um evento é distribuído no *NCLPlayer*, o método *eventDispatched* desse objeto é executado.

O método público da interface *NCLPlayerEventListener* é:

- void NCLPlayerEventDispatched(NCLPlayerEvent event)
    - Método executado quando um evento é distribuído no *NCLPlayer*.

D.2.1.4 Classe *NCLGingaSettingsNode*

A classe *NCLGingaSettingsNode* é a classe que representa um nó NCL cujos atributos são variáveis globais definidas pelo autor do documento ou variáveis de ambiente que podem ser manipuladas pelo processamento do documento NCL. A lista completa dessas variáveis de ambiente é apresentada na ABNT NBR 15606-2.

Os métodos públicos da classe *NCLGingaSettingsNode* são:

- NCLGingaSettingsNode(java.lang.String nodeId)
    - Método construtor para NCLGingaSettingsNode, que recebe como parâmetro uma String identificadora única.
- String getValue(String value)
    - Retorna o valor da variável de acordo com a descrição da variável de ambiente passada como String. A lista completa de variáveis de ambiente disponíveis encontra-se na ABNT NBR 15606-2:2007, Tabela 12.

D.2.1.5 Classe *NCLEdit*

A classe *NCLEdit* oferece métodos para edição de um documento NCL, que encapsulado em um objeto da classe *NCLPlayer* instancia objetos da classe NCLEdit (método *getNCLEdit*) associados. Comandos de edição provenientes da instância de NCLEdit alteram somente a apresentação do documento NCL (representada pelo objeto NCLPlayer) – o documento original é preservado durante todo o processo de edição, conforme especificado para comandos de edição NCL em ABNT NBR 15606-2:2007.

Os métodos públicos da classe *NCLEdit* são:

- boolean addRegion(java.lang.String regionBaseId, java.lang.String regionId, java.lang.String regionStr)
    - Adiciona um elemento <region> no documento NCL, como membro da *region base* identificada pela String *regionBaseId*, como filho do elemento identificado pela String *regionId* e com sua definição na String *regionStr*. Retorna *true* em caso de sucesso, e *false* caso contrário.
- boolean removeRegion(java.lang.String regionId)
    - Remove o elemento <region> identificado pela String regionId do documento NCL. Retorna *true* em caso de sucesso e *false* caso contrário.
- boolean addRegionBase(java.lang.String regionBaseStr)
    - Adiciona o elemento <regionBase> descrito na String *regionBaseStr* ao elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* caso contrário.
- boolean removeRegionBase(java.lang.String regionBaseId)
    - Remove o elemento <regionBase> identificado pela String *regionBaseId* do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addRule(java.lang.String ruleStr)
    - Adiciona um elemento <rule> descrito na String *ruleStr* como integrante do elemento <ruleBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeRule(java.lang.String ruleId)
    - Remove o elemento <rule> identificado na String *ruleId* do elemento <ruleBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addRuleBase(java.lang.String ruleBaseStr)
    - Adiciona um elemento <ruleBase> descrito na String *ruleBaseStr* como integrante do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* caso contrário.
- boolean removeRuleBase(java.lang.String ruleBaseId)
    - Remove o elemento <ruleBase> identificado na String *ruleBaseId* do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addConnector(java.lang.String connectorStr)
    - Adiciona um elemento <connector> descrito na String *connectorStr* como integrante do elemento <connectorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeConnector(java.lang.String connectorId)
    - Remove o elemento <connector> identificado na String *connectorId* do elemento <connectorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.

- boolean addConnectorBase(java.lang.String connectorBaseStr)
    - Adiciona um elemento <connectorBase> descrito na String *connectorBaseStr* como integrante do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeConnectorBase(java.lang.String connectorBaseId)
    - Remove o elemento <connectorBase> identificado na String *connectorBaseId* do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addDescriptor(java.lang.String descriptorStr)
    - Adiciona um elemento <descriptor> descrito na String *descriptorStr* como integrante do elemento <descriptorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* caso contrário.
- boolean removeDescriptor(java.lang.String descriptorId)
    - Remove o elemento <descriptor> identificado na String *descriptorId* do elemento <descriptorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addDescriptorSwitch(java.lang.String descriptorSwitchStr)
    - Adiciona um elemento <descriptorSwitch> descrito na String *descriptorSwitchStr* como integrante do elemento <descriptorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeDescriptorSwitch(java.lang.String descriptorSwitchId)
    - Remove o elemento <descriptorSwitch> identificado na String *descriptorSwitchId* do elemento <descriptorBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addDescriptorBase(java.lang.String descriptorBaseStr)
    - Adiciona um elemento <descriptorBase> descrito na String *descriptorBaseStr* como integrante do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeDescriptorBase(java.lang.String descriptorBaseId)
    - Remove o elemento <descriptorBase> identificado na String *descriptorBaseId* do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addTransition(java.lang.String transitionStr)
    - Adiciona um elemento <transition> descrito na String *transitionStr* como integrante do elemento <transitionBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeTransition(java.lang.String transitionId)
    - Remove o elemento <transition> identificado na String *transitionId* do elemento <transitionBase> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addTransitionBase(java.lang.String transitionBaseStr)
    - Adiciona um elemento <transitionBase> descrito na String *transitionBaseStr* como integrante do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeTransitionBase(java.lang.String transitionBaseId)
    - Remove o elemento <transitionBase> identificado na String *transitionBaseId* do elemento <head> do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.

- boolean addImportBase(java.lang.String docBaseId, java.lang.String importBaseStr)
    - Adiciona a um elemento base NCL identificado na String *docBaseId* (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>) a definição do elemento <importBase> contida na String *importBaseStr* no documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeImportBase(java.lang.String docBaseId, java.lang.String importBaseId)
    - Remove o elemento <importBase> identificado na String *importBaseId* de um elemento base NCL identificado na String *docBaseId* (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> ou <connectorBase>) do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addImportedDocumentBase(java.lang.String importedDocumentBaseStr)
    - Adiciona ao elemento <head> do documento NCL a definição do elemento <importedDocumentBase> contida na String *importedDocumentBaseStr*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeImportedDocumentBase(java.lang.String importedDocumentBaseId)
    - Remove do elemento <head> do documento NCL o elemento <importedDocumentBase> identificado pela String i*mportedDocumentBaseId*.
- boolean addImportNCL (java.lang.String importNCLStr)
    - Adiciona ao elemento <importedDocumentBase> do documento NCL a definição do elemento <importNCL> contida na String *importNCLStr*. Retorna *true* em caso de sucesso e *false* em caso contrário. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeImportNCL(java.lang.String importNCLId)
    - Remove do elemento <importedDocumentBase> do documento NCL o elemento <importNCL> identificado pela String i*mportNCLId*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addNode(java.lang.String compositeId, java.lang.String nodeStr)
    - Adiciona a um nó de composição NCL identificado na String *compositeId* (<body>, <context> ou <switch>) a definição de um nó NCL (<media>, <context> ou <switch>) contida na String *nodeStr*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeNode(java.lang.String compositeId, java.lang.String nodeId)
    - Remove de um nó de composição NCL identificado na String *compositeId* (<body>, <context> ou <switch>) a definição de um nó NCL (<media>, <context> ou <switch>) identificada na String *nodeId*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean addInterface(java.lang.String nodeId, java.lang.String interfaceStr)
    - Adiciona uma interface do NCL (elemento <port>, <area>, <property> ou <switchPort>) descrita na String *interfaceStr* a um nó (elemento <media>, <body>, <context> ou <switch>) identificado pela String *nodeId* do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeInterface(java.lang.String nodeId, java.lang.String interfaceId)
    - Remove uma interface do NCL (elemento <port>, <area>, <property> ou <switchPort>) identificado na String *interfaceId* de um nó (elemento <media>, <body>, <context> ou <switch>) identificado pela String *nodeId* do documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.

- boolean addLink(java.lang.String compositeId, java.lang.String linkStr)
    - Adiciona a um nó de composição NCL identificado na String *compositeId* (<body>, <context> ou <switch>) a definição de um elemento <link> NCL contida na String *linkStr*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean removeLink(java.lang.String compositeId, java.lang.String linkId)
    - Remove de um nó de composição NCL identificado na String *compositeId* (<body>, <context> ou <switch>) a definição de um elemento <link> NCL identificada na String *linkId*. Retorna *true* em caso de sucesso e *false* em caso contrário.
- boolean setPropertyValue(java.lang.String propertyId, java.land.*String* value)
    - Atribui o valor da String *value* a uma propriedade identificada por *propertyId*. O valor da instância de *String propertyId* deve obrigatoriamente identificar um atributo *name* de um elemento <property> ou um atributo id de elemento <switchPort>. O <property> ou <switchPort> deve obrigatoriamente pertencer a um nó (elemento <body>, <context>, <switch> ou <media>) de um documento NCL. Retorna *true* em caso de sucesso e *false* em caso contrário.

D.2.2 Pacote br.org.sbtvd.bridge.ncl

D.2.2.1 Classe *NodeManager*

A classe estática *NodeManager* possui os métodos para registro de NCLEventListeners de forma que um Xlet Ginga-J associado a um nó de mídia NCL (elemento <media>) possa receber eventos (encapsulados em instâncias da classe NCLEvent) do ambiente Ginga-NCL. Informações adicionais podem ser obtidas na ABNT NBR 15606-2:2007, 10.3.4.3 e 11.2.

Os métodos públicos da classe *NodeManager* são:

- *static* void addNCLEventListener(NCLEventListener listener)
    - Este método registra um NCLEventListener para receber todas as instâncias de NCLEvent distribuídas pelo ambiente Ginga-NCL para o Xlet associado a um nó de mídia (elemento <media>) em uma aplicação Ginga-NCL.
- *static* void removeNCLEventListener(NCLEventListener listener)
    - Remove um NCLEventListener previamente registrado.
- *static* NCLEventListener[] getNCLEventListeners()
    - Retorna uma lista de todas as instâncias de NCLEventListener registradas junto ao NodeManager. Os objetos ouvintes adicionados várias vezes aparecem apenas uma vez na lista retornada.

D.2.2.2 Classe *NCLEvent*

A classe *NCLEvent* estende java.util.EventObject e representa a classe de evento raiz para todos os eventos gerados pelo formatador NCL que manipula um documento incluindo um Xlet Ginga-J. As máscaras de evento definidas nesta classe são utilizadas para especificar quais tipos de eventos um *NCLEventListener* deve ouvir. Informações adicionais podem ser obtidas na ABNT NBR 15606-2:2007, 10.3.4.3 e 11.2.

As constantes públicas estáticas da classe *NCLEvent* são:

- *static* int PRESENTATION_START
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o início da reprodução (*start*) de uma âncora definida para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.

- *static* int PRESENTATION_STOP
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o término da reprodução (*stop*) de uma âncora definida para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.
- *static* int PRESENTATION_ABORT
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi o abortamento da reprodução (*abort*) de uma âncora definida para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.
- *static* int PRESENTATION_PAUSE
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi a pausa da reprodução (*pause*) de uma âncora definida para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.
- *static* int PRESENTATION_RESUME
    - Máscara de evento para eventos de apresentação (tipo *presentation*) cuja ação (campo *action*) foi a retomada da reprodução (*resume*) de uma âncora definida para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.
- *static* int ATTRIBUTION_SET
    - Máscara de evento para eventos de atribuição (tipo *attribution*) cuja ação (campo *action*) foi a definição (*set*) de um parâmetro definido para o nó de mídia (elemento <*media*>) que inclui o Xlet Ginga-J.
- protected int id
    - Identificação do evento.

Os métodos públicos da classe *NCLEvent* são:

- NCLEvent(Object source, int id, String value)

  o Método construtor para NCLEvent, que recebe como parâmetro uma referência (source) do objeto que originou o evento, um inteiro (id) que identifica o evento e um identificador (value) do nó ou âncora relacionado ao evento.
- int getID()
    - Retorna o tipo de evento.
- String getValue()
    - Retorna o identificador do nó ou âncora relacionado ao evento.

D.2.2.3 Interface *NCLEventListener*

A interface *Listener* deve ser implementada por quem deseja receber notificação de eventos distribuídos pelo formatador NCL manipulando um documento NCL que inclui um Xlet Ginga-J, sendo objetos que são elementos da classe *NCLEvent*. Estende a interface java.util.EventListener.

A aplicação que deseja monitorar os eventos gerados pelo formatador NCL deve implementar esta interface.

O método público da interface *NCLEventListener* é:

- void NCLPlayerEventDispatched(NCLEvent event)
    - Método executado quando um evento NCLEvent é gerado pelo formatador NCL

Anexo E
(normativo)

Especificação API de suporte a planos gráficos – Pacote br.org.sbtvd.ui

E.1 Classe *ColorCoding*

A classe *ColorCoding* abriga constantes para enumerar os diferentes modelos de codificação possíveis para cada plano. Os valores possíveis correspondem àqueles retornados em com.sun.dtv.ui.Plane.getColorCodingModel().

As constantes públicas estáticas presentes nesta classe são:

- *public static* final int ARGB8888 = 1
    - indica que o modelo de cores no plano é ARGB8888
- *public static* final int YUV442 = 2
    - indica que o modelo de cores no plano é YUV442.
- *public static* final int YUV444 = 3
    - indica que o modelo de cores no plano é YUV444.
- *public static* final int ONE_BPP = 4
    - indica que o modelo de cores no plano é um bit por pixel

E.2 Classe *StillPicture*

A classe *StillPicture* estende a classe com.sun.dtv.lwuit.Component. É o meio pelo qual imagens JPEG são adicionadas ao plano de imagens estáticas. Esta classe deve ser declarada com o modificador final.

Os construtores públicos são os seguintes:

- StillPicture(String path)
    - Constrói um objeto StillPicture. O caminho passado no parâmetro path deve corresponder à localização de uma imagem JPEG no sistema de arquivos da aplicação.

Instâncias desta classe não suportam as funcionalidades de foco nem de animação.

Os seguintes métodos herdados do com.sun.dtv.lwuit.Component não podem ser invocados direitamente pelas aplicações:

- void paint(Graphics g)
- void paintBackgrounds(Graphics g)
- void paintComponent(Graphics g)
- void paintComponent(Graphics g, boolean background)

E.3 Classe *SwitchArea*

A classe *SwitchArea* estende a classe com.sun.dtv.lwuit.Component. Esta classe deve ser declarada com o modificador final.

É um componente que define uma área retangular do plano de seleção vídeo/imagem. Cada área retangular adicionada por meio do método com.sun.dtv.ui.DTVContainer#addComponent() corresponde a uma área em que o plano de imagens estáticas aparecerá sobre o plano de vídeo ou vice-versa dependendo da cor do estilo (*com.sun.dtv.lwuit.plaf.Style*) do componente.

Instâncias desta classe não suportam as funcionalidades de foco nem de animação.

Os seguintes métodos herdados do com.sun.dtv.lwuit.Component não podem ser invocados direitamente pelas aplicações:

- void paint(Graphics g)
- void paintBackgrounds(Graphics g)
- void paintComponent(Graphics g)
- void paintComponent(Graphics g, boolean background)

Por meio do *com.sun.dtv.lwuit.plaf.Style* associado a cada componente pode ser definido se o vídeo será exibido por cima do *Still Picture Plane* ou vice-versa. As instâncias de com.sun.dtv.lwuit.plaf.Style associadas a este componente só podem conter cores sólidas (java.awt.Color). A cor preta, java.awt.Color. BLACK, representa que o vídeo deve ser exibido por cima do Still Picture Plane. Com o uso de qualquer outra cor, o conteúdo do Still Picture Plane será exibido na frente do vídeo.

Bibliografia

[1] SOUZA FILHO, Guido Lemos de; LEITE, Luiz Eduardo Cunha; BATISTA, Carlos Eduardo Coelho Freire. *Ginga-J: The Procedural Middleware for the Brazilian Digital TV System.* In: _____ Journal of the Brazilian Computer Society. No. 4, Vol. 13. p.47-56. ISSN: 0104-6500. Porto Alegre, RS, 2007.

[2] SOARES, Luiz Fernando Gomes; RODRIGUES, Rogério Ferreira; MORENO, Márcio Ferreira. *Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System*. In: _____ Journal of the Brazilian Computer Society. No. 4, Vol. 13. p.37-46. ISSN: 0104-6500. Porto Alegre, RS, 2007.

[3] Sun Microsystems, Java Digital Television (DTV) API:2008, < http://java.sun.com/javame/technology/javatv/index.jsp>

[4] Sun Microsystems, Java TV API:2007, <http://java.sun.com/products/javatv/>

[5] Sun Microsystems, Java Media Framework API (JMF), <http://java.sun.com/products/java-media/jmf/index.jsp>