3
Tecnologias Relacionadas

No contexto da TV digital, os equipamentos das emissoras e os terminais de acesso devem compartilhar de protocolos bem definidos para que o intercâmbio de informações e a provisão de serviços sejam possíveis. As seções a seguir detalham padrões e protocolos utilizados com esse propósito, o modelo de apresentação comumente adotado em sistemas de TV digital, bem como uma discussão sobre sistemas operacionais para terminais de acesso. Uma atenção é dada à biblioteca DirectFB pela sua importância neste trabalho.

3.1.
MPEG-2 Sistemas

O padrão MPEG-2 *Systems* (ISO, 2000a), ou MPEG-2 Sistemas, estabelece como um ou mais sinais de áudio e vídeo, assim como outros dados (imagens estáticas, texto etc.), devem ser combinados de forma a serem transmitidos ou armazenados apropriadamente.

O MPEG-2 Sistemas define um fluxo elementar (ES – *Elementary Stream*) como um fluxo gerado pela codificação do conteúdo de vídeo, áudio ou dados específicos. Por ser um fluxo contínuo de informação, um ES pode apresentar dificuldades em sua manipulação (i.e. transmissão, armazenamento, entre outras (ISO, 2000a)). Assim, um ES costuma ser dividido em pacotes, gerando um novo fluxo chamado de PES (*Packetized Elementar Stream*).

As especificações MPEG-2 determinam o termo programa, chamado de serviço no contexto da TV digital, como um grupo composto de um ou mais fluxos PESs, correspondendo a sinais de vídeo, áudio e dados. Lembrando que, atualmente, a maioria dos discos DVDs utilizam codificação MPEG-2 Sistemas, o conteúdo desses discos é um bom exemplo de programa a ser citado.

A partir da definição de serviço, define-se dois formatos de multiplexação de fluxos no MPEG-2 Sistemas: o Fluxo de Transporte e o Fluxo de Programa. O Fluxo de Transporte pode conter vários serviços simultaneamente. Cada serviço

pode ter uma base de tempo diferente. O Fluxo de Programa só pode conter um serviço. Independente do formato de multiplexação, todos os fluxos elementares pertencentes a um mesmo serviço utilizam a mesma base de tempo. O formato de multiplexação utilizado pelos principais sistemas de TV digital existentes é o Fluxo de Transporte.

Simplificadamente, multiplexar serviços em um Fluxo de Transporte significa organizar os pacotes dos vários fluxos PESs, pertencentes aos serviços contemplados, em um único fluxo. Para isso, é necessário inserir no Fluxo de Transporte informações para que o decodificador MPEG-2 Sistemas saiba identificar a qual fluxo PES um determinado pacote pertence e, ainda, a qual serviço um dado fluxo PES pertence.

Cada fluxo PES possui um identificador de fluxo único, denominado stream_id. Para permitir ao decodificador MPEG-2 Sistemas identificar a qual fluxo PES um determinado pacote pertence, o padrão MPEG-2 Sistemas define que cada pacote de um fluxo PES, ao ser multiplexado em um Fluxo de Transporte, deve receber o identificador do fluxo PES a que pertence.

Uma vez que os fluxos PES não possuem nenhuma indicação sobre a qual serviço estão associados, faz-se necessário um mecanismo para a determinação de quais serviços estão presentes no Fluxo de Transporte e quais os fluxos PESs que compõem cada serviço. Esse mecanismo é especificado no MPEG-2 Sistemas através de um conjunto de tabelas, ou metadados, de informação específica de programa (PSI – *Program Specific Information*), conhecidas no contexto da TV digital como SI (*Service Information*). A SI denominada PMT (*Program Map Table*) contém a lista de identificadores dos fluxos PESs que compõem um serviço. As PMTs são localizadas através da SI denominada PAT (*Program Association Table*), que contém o identificador do fluxo elementar contendo as PMTs. O processo de identificação no decodificador, de cada serviço presente em um Fluxo de Transporte, é ilustrado na Figura 7.

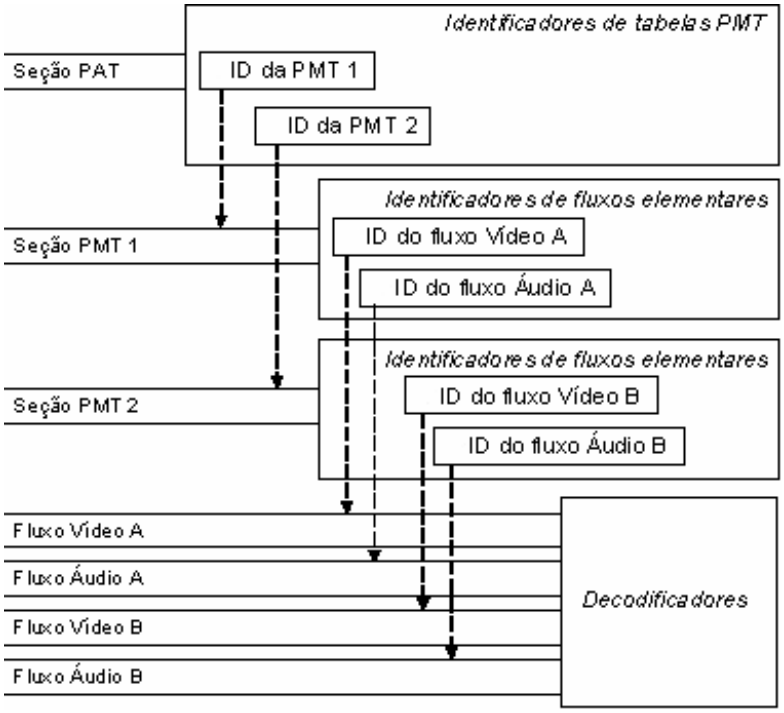

Figura 7: Relacionamento entre SIs e fluxos elementares.

O processo de identificação de cada serviço no decodificador inicia-se, quando do recebimento de um Fluxo de Transporte, pela busca da seção MPEG-2 que contém a SI PAT através de seu identificador conhecido. Segundo o padrão MPEG-2 Sistemas, as SIs PAT e PMT devem ser enviadas no Fluxo de Transporte a cada meio segundo, no máximo.

De posse das informações da SI PAT, são identificados os fluxos elementares que contêm as SIs PMT de cada serviço. Cada SI PMT fornece os identificadores de cada fluxo elementar que compõe o respectivo serviço. Os fluxos elementares referentes a cada serviço são, então, enviados para o módulo de decodificação apropriado.

Além das tabelas SIs e do conteúdo de áudio e vídeo principal, um Fluxo de Transporte MPEG-2 pode conter ainda outros dados, como arquivos, legendas, informações temporais, entre outros (ISO, 2000a).

O mecanismo de dividir cada fluxo elementar em pacotes, com o objetivo de possibilitar a multiplexação de fluxos elementares em um único Fluxo de Transporte, é ideal para fluxos elementares de áudio ou vídeo, onde cada pacote deve conter um certo número de quadros de vídeo ou amostras de áudio (ISO, 2000a). Entretanto, os fluxos de dados (incluindo as tabelas SIs) geralmente

possuem requisitos e características diferentes, que fazem com que essa divisão em pacotes torne-se mais complicada:

- Possivelmente existe mais de um item em um mesmo fluxo de dados como, por exemplo, diferentes tabelas SIs ou diferentes arquivos em uma transmissão de um sistema de arquivos. Assim, faz-se necessária uma maneira para determinar a forma com que os dados são organizados no fluxo;
- É possível existir mais de uma versão de um determinado tipo de dados. Por exemplo, as tabelas SIs ou módulos do carrossel de objetos, discutidos na próxima seção, podem ser atualizados e devem ser sobrescritos nos terminais de acesso;
- Existe a necessidade de identificar, ou mesmo oferecer suporte aos diferentes tipos de dados necessários.

Para atender a esses requisitos e características, o padrão MPEG-2 define o conceito de seções privadas, também chamadas apenas de seções, utilizadas para encapsular dados para serem multiplexados, de forma análoga aos pacotes PES.

As seções privadas MPEG-2 possuem a estrutura apresentada na Tabela 2. Elas são otimizadas para carregar dados DSM-CC, que serão discutidos a seguir, tabelas SIs, entre outros (ISO, 2000a). Para isso, possuem um cabeçalho especificado com o objetivo de informar ao decodificador como as seções estão sendo utilizadas para transportar os dados, como elas devem ser remontadas, qual o tipo de dados transportado, além de outros parâmetros para recuperação da informação (ISO, 2000a).

Sintaxe

private_section() {
table_id
section_syntax_indicator
private_indicator
Reserved
private_section_length
if(section_syntax_indicator == '0') {
for(i=0; i<N; i++) {
private_data_byte
}
}
else{
table_id_extension
Reserved
version_number
current_next_indicator
section_number
last_section_number
for(i=0; i<private_section_length-9; i++) {
private_data_byte
}
CRC_32
}
}

Tabela 10: Estrutura de uma Seção Privada MPEG-2 (ISO, 2000a)

3.2. DSM-CC

DSM-CC é uma extensão do padrão MPEG-2 Sistemas, publicado em 1996 como um padrão internacional ISO (ISO, 1998b). O DSM-CC foi desenvolvido para oferecer diversos tipos de serviços multimídia, entre eles a transmissão de dados multiplexados com o conteúdo audiovisual em um Fluxo de Transporte.

As especificações do padrão DSM-CC trazem definições interessantes que são comumente aplicadas ao contexto da TV digital. Entre as principais está o mecanismo de transmissão cíclica, denominado carrossel de objetos. Considerando que um usuário pode ligar seu terminal de acesso quando bem entender, é necessário aos provedores de conteúdo garantir a entrega da informação (dados como, por exemplo, imagens, texto, aplicações, entre outros (ISO, 1998b)) nos terminais de acesso.

3.2.1. Carrossel de Objetos DSM-CC

O carrossel de objetos DSM-CC tem como objetivo prover a transmissão cíclica de objetos. O padrão DSM-CC especifica três tipos de objetos: arquivos, diretórios e eventos. Assim, o carrossel de objetos pode possuir um verdadeiro

sistema de arquivos, isto é, um conjunto de diretórios e arquivos que, por exemplo, formam uma aplicação a ser executada nos terminais de acesso.

As especificações DSM-CC determinam que os dados transmitidos através do carrossel de objetos devem ser divididos em unidades denominadas módulos. Cada módulo pode possuir mais de um arquivo desde que não ultrapasse um total de 64 Kbytes. Os arquivos que estão em um mesmo módulo podem fazer parte de diretórios diferentes. Um arquivo que possui mais de 64 Kbytes deve ser transmitido em um único módulo, pois não é permitido dividir um arquivo em mais de um módulo.

Uma vez que os objetos foram dispostos em módulos, cada módulo é então transmitido, um após o outro. Após transmitir o último módulo, a transmissão é reiniciada desde o início (i.e. o primeiro módulo transmitido). O resultado disso é um fluxo elementar que contém o sistema de arquivos transmitido de forma cíclica. Assim, se um determinado terminal de acesso não recebeu uma parte de um módulo em particular (devido a um erro na transmissão ou por ter sido iniciado após a transmissão desse módulo), basta esperar pela retransmissão desse módulo.

Em alguns casos, utilizar o carrossel de objetos dessa forma significa introduzir retardos impraticáveis para os dados enviados pelo provedor de conteúdo. Por exemplo, para um carrossel que possui tamanho total de 172 Kbytes transmitidos a uma taxa de 128 Kbps, são necessários aproximadamente 11 segundos para a transmissão de um ciclo completo. Assim, nesse exemplo, existe um retardo máximo de 11 segundos para carregar um arquivo qualquer que faz parte desse carrossel. Para amenizar esse problema, os geradores de carrossel oferecem como opção transmitir alguns módulos com maior freqüência que outros. Assim, os módulos que contêm arquivos com maior prioridade podem ser transmitidos com maior freqüência.

Para que o exemplo seja mais concreto, considere a estrutura de diretórios da Figura 8. Essa estrutura conta com um arquivo `index.htm` que deve ser exibido nos terminais de acesso. O arquivo `index.htm` possui como âncora a imagem estática `lfgs.jpg`. Uma possível seleção dessa âncora, por parte do usuário do terminal de acesso, iniciaria a exibição dos arquivos `pagu.avi` e `chorinho.wav`.

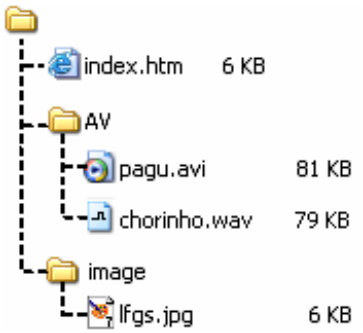

Figura 8: Estrutura de Diretórios.

Para transmitir a estrutura da Figura 8 via carrossel de objetos, primeiro é necessário dispor seus arquivos em módulos. Alocar o arquivo `index.htm` a um módulo é trivial. O próximo arquivo da estrutura (`pagu.avi`) possui mais de 64 Kbytes, não existindo a possibilidade de fazer parte do mesmo módulo de `index.htm`. Assim, `pagu.avi` é alocado em um módulo único. Analogamente, o arquivo `chorinho.wav` deve ser alocado em um terceiro módulo. Já o arquivo `lfgs.jpg` pode ser alocado no mesmo módulo que `index.htm`, uma vez que o tamanho desses dois arquivos somados é inferior a 64 Kbytes. A disposição final de arquivos em módulos é ilustrada na Figura 9.

Figura 9: Divisão da Estrutura de Diretórios da Figura 8 em Módulos.

O próximo passo é determinar a disposição dos módulos no carrossel. É interessante observar que se o carrossel possuir como ciclo de transmissão a seqüência de módulos: 1-2-3; um retardo de onze segundos poderia ser introduzido (para uma transmissão a 128 Kbps). Como os arquivos de áudio e vídeo dependem dos arquivos `index.htm` e `lfgs.jpg` para serem exibidos, uma disposição interessante de módulos do carrossel seria: 1-2-1-3, apresentada na Figura 10.

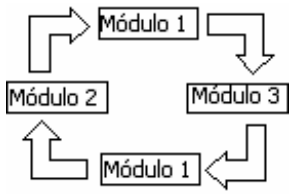

Figura 10: Disposição dos Módulos da Figura 9 no Carrossel de Objetos.

Finalmente, nesse exemplo, um objeto de evento deveria ser transmitido (nesse mesmo carrossel de objetos). Simplificadamente, o objeto de evento possui informações sobre os eventos DSM-CC que farão com que a aplicação responsável exiba o arquivo `index.htm`. Os objetos de evento, bem como os eventos DSM-CC, são explicados a seguir.

3.2.2. Eventos DSM-CC

Sincronizar o comportamento de uma determinada aplicação com o conteúdo de uma programação de TV específica é extremamente interessante, principalmente quando a aplicação em questão possui alguma relação semântica com essa programação. Por exemplo, considere como programação de TV um jogo de futebol onde existe uma aplicação, enviada anteriormente aos terminais de acesso através de carrossel de objetos, responsável por exibir o placar de outros jogos da rodada. Ao sair um gol em um desses outros jogos, é necessário que o placar seja atualizado na aplicação. Para esse tipo de sincronismo, é utilizada uma funcionalidade especificada no padrão DSM-CC, denominada eventos DSM-CC.

Um evento DSM-CC pode ser definido como uma ocorrência instantânea no tempo. Para criar um evento DSM-CC, é inserida uma estrutura no Fluxo de Transporte, denominada descritor de eventos. Cada descritor de eventos possui um identificador numérico para permitir que cada evento seja identificado de forma única no Fluxo de Transporte. Uma vez que não existe a possibilidade do provedor de conteúdo precisar exatamente onde o descritor é inserido no fluxo, cada descritor possui ainda uma referência temporal que indica em qual instante o evento deverá ocorrer. Como opção, um descritor de eventos pode informar ao sistema que o evento deve ocorrer imediatamente – esse tipo de evento é chamado de evento "*do it now*". Finalmente, o descritor de eventos possui ainda dados específicos das aplicações. No exemplo do parágrafo anterior, esses dados seriam responsáveis por informar qual time teria feito o gol, permitindo à aplicação realizar a alteração apropriada no placar.

Paralelamente à transmissão de um descritor de evento, o Fluxo de Transporte deve possuir ainda um carrossel de objetos contendo, pelo menos, um objeto de evento. Esse objeto é responsável por atribuir nomes aos eventos que podem ocorrer. Ou seja, o provedor de conteúdo envia por carrossel de objetos um

objeto de evento que possui uma tabela que relaciona nomes textuais, conhecidos pelas aplicações, aos identificadores numéricos dos eventos que serão criados pelos descritores. Um exemplo de objeto de evento é apresentado na Tabela 11. Note que cada nome textual deve ser associado a um único identificador. O nome textual presente na tabela do objeto de evento permite que cada aplicação se registre como *listener* de um tipo de evento DSM-CC. As aplicações podem ser preparadas para sempre se registrarem como *listeners* de eventos com nomes específicos. Por exemplo, a aplicação do placar discutida anteriormente é instruída para sempre se registrar em eventos com nome "Gol", "CartãoAmarelo" e "CartãoVermelho".

Nome do Evento	ID do Evento
"Gol"	04
"Cartão Amarelo"	06
"CartãoVermelho"	07

Tabela 11: Exemplo de Objeto de Evento DSM-CC.

Para tornar o exemplo mais concreto, é considerada a Figura 11. Um jogo entre Portuguesa e Marília é transmitido enquanto a aplicação exibe o placar, identificado pelo número 1 na figura, de outros jogos da rodada. Paralelamente, um carrossel de objetos contendo o objeto de evento apresentado na Tabela 11 é transmitido. No exemplo, a aplicação do placar se registra apenas para receber os eventos identificados com os valores numéricos 4, 6 ou 7, por possuírem os nomes "Gol", "Cartão Amarelo" e "CartãoVermelho", respectivamente. Ao ocorrer um gol em um dos outros jogos, o provedor de conteúdo envia o descritor de eventos, identificado por "a" na Figura 11, contendo as informações necessárias. Ao receber um evento cujo identificador numérico possui o valor 4, a aplicação reconhece que deve computar um gol de acordo com os dados presentes no descritor do evento e atualiza o placar (número 2 na Figura 11).

Analogamente, quando a aplicação recebe o evento, descrito em "b" na Figura 11, ela sabe que não deve computar um gol, mas sim informar que um cartão vermelho foi mostrado ao jogador "Nilmar" do time "Corinthians". A aplicação exibe assim o placar identificado pelo número 3 na Figura 11.

Conforme as especificações do padrão DSM-CC, descritores com mesmo identificador podem ser enviados quantas vezes forem necessários. Assim, ao

receber um segundo descritor cujo ID possui o valor numérico 4, a aplicação sabe que outro gol deve ser computado (descritor "c" e número 4 na Figura 11).

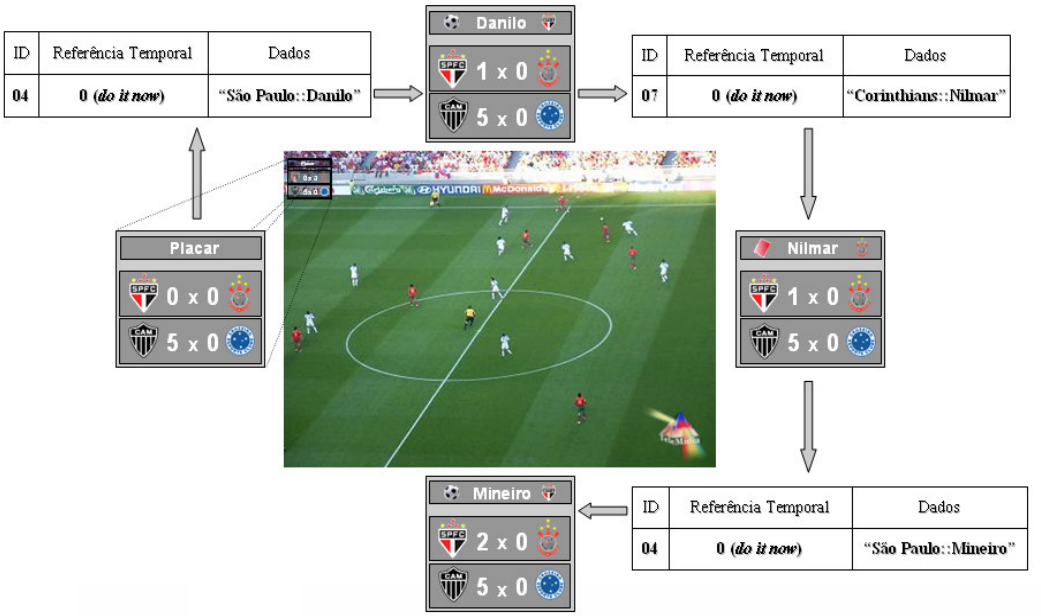

Figura 11: Exemplo de Sincronismo através de Eventos DSM-CC.

Através do exemplo discutido, pode-se observar que as aplicações para TV digital, na maioria das vezes, não seriam funcionais sem a possibilidade de manipular a apresentação de interfaces gráficas. O modelo de apresentação adotado na maioria dos sistemas de TV digital será discutido na próxima seção.

3.3. Modelo de Apresentação

Um modelo de apresentação de TV digital é comumente dividido em três camadas, ilustradas na Figura 12. A camada de fundo normalmente é capaz de exibir apenas uma cor, sendo que, em alguns casos, pode exibir uma imagem estática. Acima da camada de fundo, vem a camada de vídeo com capacidade de exibir o vídeo que está sendo decodificado por hardware (decodificadoras MPEG-2). A maioria dos terminais de acesso possui decodificadoras com recursos limitados, o que significa poucas opções de resolução como, por exemplo: a possibilidade do vídeo ser exibido ou em tela cheia ou em um quarto, ou um conjunto limitado de coordenadas para exibição desse vídeo.

Acima da camada de vídeo, vem a camada mais importante para as aplicações de TV digital, a camada gráfica. Todas as operações gráficas das aplicações são realizadas nessa camada. A camada gráfica pode possuir uma

resolução diferente das camadas de fundo e de vídeo, bem como apresentar um formato de *pixels* diferente. Geralmente, o formato dos *pixels* da camada de vídeo é retangular, enquanto que o formato dos *pixels* da camada gráfica possui uma base quadrada.

Figura 12: Modelo de Apresentação da TV Digital.

O modelo de apresentação é considerado como uma das partes mais complexas pelos desenvolvedores de middleware (Morris & Chaigneau, 2005). Ao mesmo tempo que as camadas devem ser desenvolvidas de forma independente, pois existem diferentes componentes nos terminais de acesso responsáveis pela geração de cada uma delas, é interessante também que haja algum tipo de interação entre elas. Assim, a forma específica com que uma camada é desenvolvida pode impor limitações às outras. Por exemplo, sempre que a camada de vídeo possuir o foco, a camada gráfica pode ficar impossibilitada de receber eventos do usuário.

Os principais middlewares existentes, como MHP e DASE, utilizam os componentes da ferramenta de janelas abstratas AWT (*Abstract Window Toolkit*), seguindo as especificações HAVi (*Home Audio/Video Interoperability*) (HAVi, 1999) para controlar a imagem, ou a cor, do plano de fundo. Além disso, os componentes HAVi são utilizados na camada gráfica para renderizar as interfaces gráficas das aplicações e exibir vídeos secundários (Morris & Chaigneau, 2005). O vídeo decodificado por hardware é exibido diretamente na camada de vídeo.

Os componentes HAVi/AWT, entretanto, dependem de uma máquina virtual Java. Isso significa, conforme discutido no Capítulo 1, perda no

desempenho e custos sobre royalties e propriedade intelectual. Para alcançar melhor desempenho e utilizar o mínimo de recursos, este trabalho considera a biblioteca DirectFB (Hundt, 2004). Resumidamente, DirectFB oferece uma abstração de dispositivos, através do sistema operacional. Assim, antes de discorrer sobre a biblioteca DirectFB, os sistemas operacionais serão discutidos na próxima seção.

3.4. Sistemas Operacionais para Terminais de Acesso

Os sistemas operacionais voltados para computadores em geral oferecem um conjunto de chamadas de sistema e bibliotecas de programação que possibilitam a criação de processos (representação interna de programas em um sistema), o gerenciamento de memória, o acesso a sistemas de arquivos e a comunicação básica entre processos (Tanenbaum, 1992). Simplificadamente, pode-se dizer que sistemas operacionais gerenciam o ambiente de execução de programas, definindo abstrações para tornar mais amigável e singular o acesso a estruturas de controle de mais baixo nível, relacionadas diretamente com o hardware. Foram concebidos no intuito de tornar mais simples a construção de aplicações, além de permitir a concorrência de execução entre tais aplicações, permitindo assim um compartilhamento no uso dos recursos.

Por sua vez, terminais de acesso de TV digital possuem hardware especializado para a exibição de conteúdo televisivo e forte limitação na quantidade de recursos computacionais. Em TV digital, vários outros requisitos vêm à tona, como aqueles impostos pela recepção de dados por difusão, sincronismo, interação por controle remoto etc. Além desses, os recursos controlados por um sistema operacional para terminais de acesso incluem comunicação em rede (por meio de um canal de retorno), gerenciamento de energia, dispositivos de som e dispositivos gráficos.

Nota-se que alguns desses recursos podem ser críticos para a execução de uma aplicação multimídia distribuída, como assim o são para vários tipos de programas interativos, que podem ser disponibilizados no contexto de um sistema de TV digital. Um gerenciamento adequado, respeitando os requisitos de desempenho sobre esses recursos críticos (e.g. CPU e canal de retorno), é fundamental em qualquer perfil de terminal de acesso.

De qualquer forma, todas as funcionalidades, até aqui descritas, presentes em um sistema operacional, devem estar acessíveis para os demais componentes de software, por meio de uma API. Essa API consiste na interface oferecida pelos sistemas operacionais de terminais de acesso, com o objetivo de prover as funções para a comunicação entre o software em execução e o hardware, assim como as demais abstrações comumente definidas internamente em um sistema operacional convencional. Uma alternativa para oferecer uma abstração para a construção e manipulação do modelo de apresentação sobre o sistema operacional Linux[6] é dada pela biblioteca DirectFB. Na verdade, além de oferecer uma abstração ao modelo de apresentação, a biblioteca DirectFB oferece ainda abstrações sobre outros dispositivos (controle remoto, teclado, mouse etc.). DirectFB é o assunto da próxima seção.

3.5. DirectFB

DirectFB é uma biblioteca leve, disponível como código aberto, que oferece aceleração gráfica através de um dispositivo do sistema operacional Linux denominado *FrameBuffer Device* (Uytterhoeven, 2001). O *FrameBuffer* é um recurso nativo do *kernel* do Linux que foi originalmente desenvolvido para oferecer a visualização de imagens gráficas enquanto o sistema estivesse operando em modo texto. Assim, DirectFB permite a manipulação de interfaces gráficas sem a necessidade da existência de um servidor X (XFree, 1994).

Além da aceleração gráfica, o tamanho reduzido é uma outra importante vantagem no uso de DirectFB: uma instalação completa, com os *drivers* para aceleração gráfica e fontes, pode ocupar menos de 15 MB, enquanto uma instalação completa do Xfree86 4.2 (XFree, 1994) ocupa geralmente mais de 100 MB. Além disso, DirectFB é compatível com mais de 90% das placas de vídeo convencionais (Hundt, 2004), oferecendo ainda uma abstração para dispositivos de entrada.

[6] Linux foi escolhido por ser um sistema operacional gratuito, de código aberto e amplamente utilizado em terminais de acesso para TV digital . Além disso, Linux foi definido como o sistema operacional do modelo de referência proposto para o SBTVD, o qual consiste em um dos principais focos definidos para este trabalho.

O DirectFB foi desenvolvido através de interfaces. No contexto do DirectFB, uma interface é uma estrutura da linguagem C que contém ponteiros para funções ou para outras interfaces. O diagrama de interfaces do DirectFB é apresentado na Figura 13.

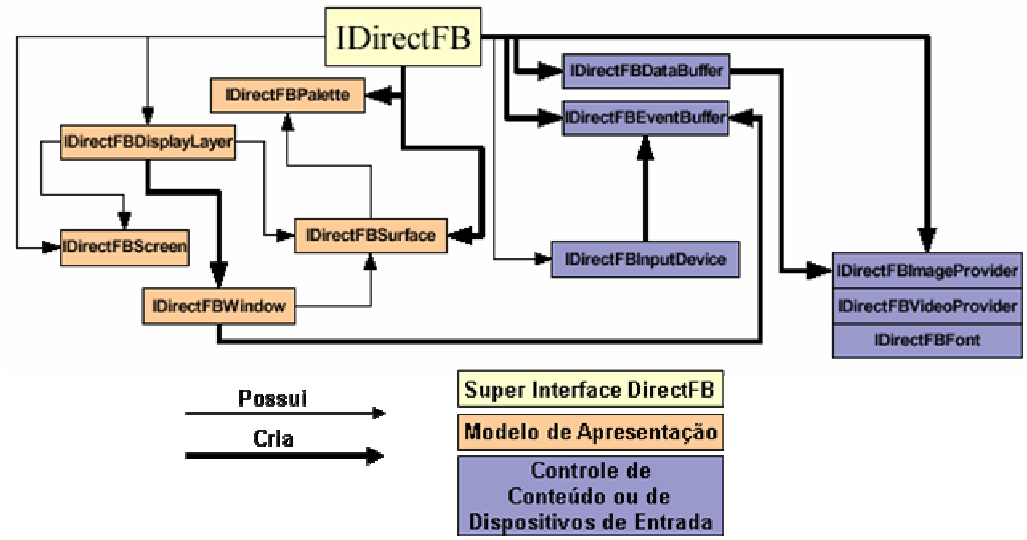

Figura 13: Diagrama de Interfaces do DirectFB

Na Figura 13, as interfaces em laranja (*IDirectFBLayer*, *IDirectFBScreen*, *IDirectFBWindow*, *IDirectFBSurface* e *IDirectFBPalette*) possuem funções para criação e manipulação do modelo de apresentação. Já as interfaces em azul (*IDirectFBInputDevice*, *IDirectFBDataBuffer*, *IDirectFBEventBuffer*, *IDirectFBImageProvider*, *IDirectFBVideoProvider* e *IDirectFBFont*) possuem funções para controlar dispositivos de entrada (*IDirectFBInputDevice*, *IDirectFBDataBuffer* e *IDirectFBEventBuffer*) ou para controlar o conteúdo das mídias exibidas no modelo de apresentação (*IDirectFBImageProvider*, *IDirectFBVideoProvider* e *IDirectFBFont*). Todas as interfaces podem ser obtidas através da super interface em amarelo *IDirectFB*, a única a ser obtida através de uma função global (*DirectFBCreate()*).

A Figura 14 ilustra como as interfaces definidas em DirectFB criam um modelo de apresentação para TV digital, conforme discutido na Seção 2.3. As camadas do modelo de apresentação são representadas pela interface *IDirectFBDisplayLayer*. Cada *IDirectDisplayFBLayer* possui informações para configurar seus recursos. Por exemplo, a interface que representa a camada de fundo, do modelo de apresentação, possui apenas atributos para configurar uma cor de fundo ou, no máximo (dependendo dos recursos da plataforma), uma

imagem de fundo. A interface que representa a camada de vídeo possui atributos para apresentar uma janela capaz de renderizar um vídeo. A interface que representa a camada gráfica, por sua vez, pode (dependendo da plataforma) ter atributos para apresentar janelas capazes de renderizar textos, imagens e vídeos.

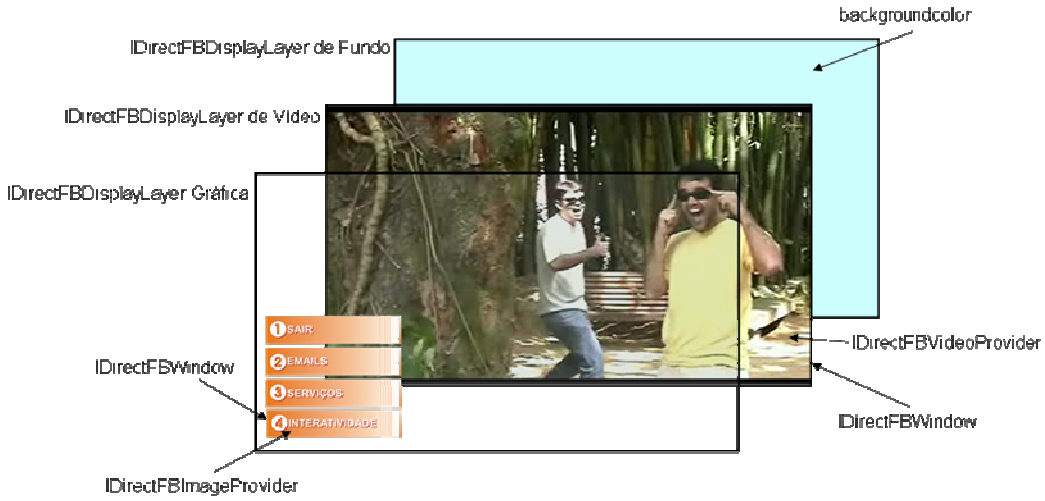

Figura 14: Modelo de Apresentação sob a Ótica do DirectFB

As janelas, criadas através da interface *IDirectFBWindow*, possuem uma superfície (*IDirectFBSurface*) capaz de renderizar o conteúdo de mídias decodificadas. Para decodificação, é necessário associar à superfície uma interface que trate o tipo específico de mídia. As interfaces disponíveis no DirectFB para essa finalidade são: *IDirectFBImageProvider*, *IDirectFBVideoProvider* e *IDirectFBFont*. As interfaces *IDirectFBImageProvider* e *IDirectFBVideoProvider* controlam a decodificação de imagens estáticas e vídeos, respectivamente, através de bibliotecas específicas. A interface *IDirectFBFont* permite exibir dados textuais.

Informações sobre a capacidade de uma superfície, como por exemplo, resoluções e cores disponíveis, se possui capacidade de renderizar vídeos, entre outras (Hundt, 2004), podem ser obtidas através da interface *IDirectFBPalette*. De forma análoga, a interface *IDirectFBScreen* provê informações sobre as camadas de uma plataforma, tais como, número de camadas suportadas pela plataforma, capacidades de apresentação de cada camada, entre outras (Hundt, 2004). Finalmente, cada janela é capaz de receber eventos (*IDirectFBEventBuffer*) através de dispositivos de entrada. A interface *IDirectFBInputDevice* oferece uma abstração desses dispositivos.

É importante ressaltar que toda renderização gráfica realizada pelas interfaces do DirectFB utilizam o dispositivo *FrameBuffer*, fazendo com que a manipulação de interfaces gráficas seja uma tarefa com baixo custo de processamento, quando comparado ao de outras bibliotecas que não utilizam esse dispositivo como, por exemplo, *XFree* e AWT.