Lucas Correia Villa Real

Uma Arquitetura para Análise de Fluxo de Dados Estruturados Aplicada ao Sistema Brasileiro de TV Digital

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

São Paulo
2009

Lucas Correia Villa Real

Uma Arquitetura para Análise de Fluxo de Dados Estruturados Aplicada ao Sistema Brasileiro de TV Digital

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia Elétrica.

Área de concentração:
Sistemas Eletrônicos

Orientador:
Prof. Dr. Marcelo Knörich Zuffo

São Paulo
2009

Ficha Catalográfica

Real, Lucas Correia Villa

Uma Arquitetura para Análise de Fluxo de Dados Estruturados Aplicada ao Sistema Brasileiro de TV Digital. – Ed. Rev. – São Paulo, 2009. 95 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1. Sistemas multimídia 2. Televisão digital 3. Análise de dados 4. Tipos abstratos de dados 5. Apresentação de informação I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos. II.t.

Dedicatória

Esse trabalho é dedicado em sua plenitude aos meus pais, por todo o apoio e carinho recebido, fundamentais para a realização dessa dissertação.

`The revolution will not be televised`

(Gil Scott-Heron)

Agradecimentos

Antes do esforço e dedicação demandados para a elaboração desta dissertação de mestrado veio uma incansável jornada de aprendizado, incentivada e apoiada incondicionalmente desde meus primeiros passos como pessoa. Agradeço e dedico essa dissertação àqueles que tornaram isto possível: meus pais. Reservo o primeiro parágrafo também à minha irmã, por quem tenho a maior admiração, e à Priscilla, que emprestou-me sua família, seu ombro e seu amor sempre que necessário.

Algumas pessoas marcaram essa trajetória em momentos de muita importância. Gostaria de agradecer aos amigos e mentores Moisés de Souza, Gerson Cavalheiro e Mark Smith por instigarem meu espírito de pesquisa. Também agradeço ao Hisham Muhammad e ao André Detsch pelos incontáveis dias e madrugadas de discussão produtiva acerca de tópicos diversos de computação e música. A música certamente ajudou.

Muito obrigado a todos do laboratório de TV digital do LSI, em especial aos colegas Rogério Nunes, Rafael Herrero, Célio Hira, Marcelo Biase, Flávio Alves, Alfredo Maruffa e Hilel Becher, que ofereceram sua experiência profissional e pessoal durante o período agitado de proposta, concepção e redação deste trabalho. Agradeço também ao meu orientador Marcelo Zuffo por sua ajuda e atenção nos momentos em que foram solicitadas.

Finalmente, gostaria de agradecer a todos os amigos e familiares que estiveram presentes e ajudaram-me de uma maneira ou outra durante esse período de dedicação exclusiva.

Resumo

Diversos sistemas computacionais transmitem informação em fluxos contínuos de dados estruturados e, por vezes, hierarquizados. Este modelo de transmissão de dados tem como uma de suas características a grande densidade de informação, o que exige de um receptor o tratamento imediato das unidades extraídas deste canal de comunicação. Muitas vezes o volume de transmissão não permite que a informação recebida seja armazenada permanentemente no receptor, o que torna a análise do conteúdo desses fluxos de dados um desafio. Este trabalho apresenta uma arquitetura para a análise de fluxo de dados estruturados aplicado à hierarquia lógica definida pelo Sistema Brasileiro de TV Digital para a transmissão de programas de televisão, validada por meio de uma implementação de referência completamente funcional.

Abstract

Various computing systems transfer data in structured data streams which also happen to be, sometimes, hierarchically organized. Such data stream model is characterized by the dense amount of information transmitted, which requires the receiver to immediately manipulate the elements extracted from that communication channel. The high rate in which data flows also makes it hard, if not impossible, for the receiver to store the desired information in its memory, which makes data flow analysis especially challenging. This work presents a novel structured data flow analysis architecture applied to the logical hierarchy defined by the Brazilian Digital TV System for the transmission of television programs, validated by means of a fully functional reference implementation.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AIT *Application Information Table* (Tabela de Informação da Aplicação)

ANSI *American National Standards Institute* (Instituto Nacional Americano de Padronização)

API *Application Programming Interface* (Interface de Programação de Aplicativos)

ARM *Advanced RISC Machines* (Máquinas RISC Avançadas)

ASCII *American Standard Code for Information Interchange* (Padrão Americano Para o Intercâmbio de Informações)

ATSC *Advanced Television Systems Committee* (Comitê de Sistemas Avançados de Televisão)

BIOP *Broadcast Inter-ORB Protocol* (Protocolo de Transmissão Inter-ORB)

CRC *Cyclic Redundancy Check* (Verificação de Redundância Cíclica)

DDB *Download Data Block* (Bloco de Dados de *Download*)

DII *Download Information Indication* (Indicação de Informação de *Download*)

DVB *Digital Video Broadcasting* (Transmissão de Vídeo Digital)

DSM-CC *Digital Storage Media Command and Control* (Comandos e Controles de Armazenamento de Mídias Digitais)

EPG *Electronic Program Guide* (Guia Eletrônico de Programação)

ES *Elementary Stream* (Fluxo Elementar)

FAT *File Allocation Table* (Tablea de Alocação de Arquivos)

FIFO *First In, First Out* (Primeiro a Entrar, Primeiro a Sair)

FUSE *File System in User Space* (Sistema de Arquivos em Espaço de Usuário)

HDTV *High Definition Television* (Televisão de Alta Definição)

HE-AAC *High-Efficiency Advanced Audio Coding* (Codificação de Áudio Avançada de Alta Eficiência)

IEC *International Electrotechnical Commission* (Comissão Eletrotécnica Internacional)

IGMP *Internet Group Management Protocol* (Protocolo de Gerenciamento de Grupos na Internet)

IP *Internet Protocol* (Protocolo de Internet)

ISDB *Integrated Services Digital Broadcasting* (Transmissão Digital de Serviços Integrados)

ISO *International Organization for Standardization* (Organização Internacional de Padronizações)

MPEG *Moving Picture Expert Group* (Grupo de Especialistas em Imagens em Movimento)

MUMPS *Massachusetts General Hospital Utility Multi-Programming System* (Sistema de multiprogramação do Hospital Geral de Massachusetts)

NIT *Network Information Table* (Tabela de Informação de Rede)

SO Sistema Operacional

PAT *Program Allocation Table* (Tabela de Alocação de Programa)

PC *Personal Computer* (Computador Pessoal)

PCR *Program Clock Reference* (Relógio de Referência do Programa)

PES *Packetized Elementary Stream* (Fluxo Elementar Empacotado)

PID *Packet Identification* (Identificador do Pacote)

POSIX *Portable Operating System Interface* (Interface de Sistema Operacional Portável)

PMT *Program Map Table* (Tabela de Mapeamento de Programa)

ProcFS *Process File System* (Sistema de Arquivos de Processos)

PSI *Program Specific Information* (Informação Específica de Programa)

RAM *Random Access Unit* (Unidade de Acesso Aleatório)

RTSP *Real Time Streaming Protocol* (Protocolo de Fluxo de Dados em Tempo Real)

SBTVD Sistema Brasileiro de TV Digital

SDT *Service Description Table* (Tabela de Descrição de Serviço)

SDTV *Simple Definition Television* (Televisão de Definição Simples)

SGML *Standard Generalized Markup Language* (Linguagem Padrão de Formatação Generalizada)

SI *Service Information* (Serviço de Informação)

SQL *Structured Query Language* (Linguagem de Consulta Estruturada)

TCP *Transfer Control Protocol* (Protocolo de Controle de Transferência)

TOT *Time Offset Table* (Tabela de Deslocamento de Horário)

TS *Transport Stream* (Fluxo de Transporte)

VFS *Virtual File System* (Sistema de Arquivos Virtual)

XML *Extensible Markup Language* (Linguagem Extensível de Formatação)

Sumário

1 Introdução

Em sistemas computacionais a transmissão de dados por meio de fluxos contínuos e estruturados é utilizada em diversas áreas da computação, como no tráfego de pacotes IP em redes de computadores, na transmissão de imagens entre um decodificador especializado e um controlador de vídeo ou ainda em aplicações financeiras, como em estatísticas da bolsa de valores.

A produção do conteúdo contido nesses fluxos ocorre muitas vezes em tempo real sendo que sua transmissão costuma se dar de forma imediata. Com isso, cabe ao receptor interpretar as estruturas de dados em tempo hábil para que elas sejam processadas e, eventualmente, descartadas de seu espaço de armazenamento finito.

Este cenário torna especialmente difícil a tarefa de aplicativos de análise de dados, visto que em muitos casos não há espaço físico para armazenar toda a informação tratada. Além disso, técnicas tradicionais de análise de programas de computador, como mensagens de rastro ou aplicativos de monitoramento e controle de outros programas, não podem ser escalados na mesma proporção em que o volume de dados recebidos aumenta. Com isso, outras metodologias se fazem necessárias para que estes dados sejam apresentados ao usuário de maneira descomplicada.

Algumas técnicas para lidar com grandes volumes de dados estruturados são sugeridas na literatura, como o uso de amostragens e do agrupamento dos elementos recebidos de acordo com algumas características estabelecidas (CORMODE, 2007). A amostragem de elementos também faz-se necessária quando não existe um critério para a seleção dos dados de entrada que serão armazenados; assim, diversos algoritmos são empregados para determinar quando um elemento do fluxo de entrada deve ser analisado e armazenado pelo receptor.

O modelo de transmissão de dados estruturados é adotado também em sistemas de TV digital, nos quais o envio dos quadros de vídeo, áudio e outros elementos é feito constantemente, de modo unidirecional, por um produtor de

conteúdo. No entanto, uma característica do protocolo de transporte adotado nesses sistemas restringe a análise do seu conteúdo a um subconjunto finito de elementos: os elementos que compõem as tabelas de informação de programas de TV digital são transmitidos repetidamente e, no cabeçalho de cada um, a sua versão é informada. Assim, o analisador do protocolo pode desconsiderar elementos cujas versões já tenham sido lidas, reduzindo o tempo de processamento e o espaço de armazenamento necessários para a sua operação.

1.1 Objetivos

Este trabalho tem como objetivo principal investigar e propor uma arquitetura para a análise de fluxo de dados estruturados, especificamente aplicada ao Sistema Brasileiro de TV Digital (SBTVD). Esta arquitetura permite que dados organizados conforme uma relação hierárquica sejam exibidos com diferentes níveis de detalhe, de acordo com a necessidade do usuário. Para garantir a portabilidade do formato no qual os dados analisados são armazenados, um modelo já consolidado que ofereça flexibilidade em múltiplas plataformas é utilizado.

A arquitetura proposta é aplicada ao modelo de transmissão de dados adotado pelo SBTVD, em vigor desde Novembro de 2007. Este estudo permitiu a elaboração de um ambiente funcional para a identificação e análise dos vários elementos estruturados transmitidos no SBTVD, que foi validado em uma plataforma PC e em um terminal de acesso de TV digital de baixo custo.

Para a elaboração deste trabalho, foram estudadas diferentes técnicas de análise de dados, como as tradicionalmente utilizadas para a inspeção de estruturas de processos em execução em um computador e técnicas especializadas em extração de informação oriunda de grandes volumes de dados. Este estudo engloba também diferentes modelos de representação de dados que permitem ao usuário um acesso direto e simples a grandes volumes de informação.

Para a aplicação da proposta ao SBTVD, foram tomados como base trabalhos de multiplexação de conteúdo em TV digital e as normas brasileiras de informações de serviço, que detalham os formatos dos elementos transmitidos no fluxo de transporte recebido pelos terminais de acesso de TV digital. O estudo de metodologias de análise se estendeu também a patentes nesta área, que descrevem com detalhes os mecanismos adotados por produtos comerciais encontrados no mercado.

1.2 Metodologia

O trabalho pode ser subdividido em seis etapas principais: (1) pesquisa sobre sistemas de análise de fluxo de dados estruturados; (2) estudo de modelos para armazenamento e representação de grandes volumes de dados estruturados; (3) proposta de uma arquitetura para a análise de fluxo deste tipo de dados; (4) aplicação desta arquitetura ao formato de dados transmitido no Sistema Brasileiro de TV Digital; (5) especificação e implementação de um analisador de fluxo de dados estruturados para o Sistema Brasileiro de TV Digital; (6) avaliação do modelo proposto e implementado.

A primeira etapa consiste em uma análise do estado-da-arte e de mecanismos que buscam resolver o problema de monitoramento de fluxo de dados estruturados. São introduzidas técnicas fundamentais usadas para resumir a informação contida nestes fluxos, como a amostragem, o agrupamento de amostras e o uso de estruturas compactas, que revelam estatísticas de determinados elementos.

Na segunda etapa são analisados modelos de armazenamento e representação de dados estruturados. Conceitos de banco de dados e de sistemas de arquivos são revisados, bem como formatos de arquivos flexíveis e padronizados que permitam o armazenamento de dados estruturados e hierarquizados.

A proposta de uma arquitetura para a análise de fluxo de dados hierárquico é então apresentada. São levados em consideração os fundamentos revisados na primeira e segunda etapas, que dão respaldo para a definição de um mecanismo portável e multiplataforma.

Na quarta etapa, a arquitetura proposta é aplicada ao protocolo de dados presente no Sistema Brasileiro de TV Digital (SBTVD). São introduzidos conceitos relevantes para a compreensão de como um terminal de acesso de TV digital determina quais são os elementos que compõem um programa enviado por uma emissora e de como essa informação é expressa na arquitetura de forma a ressaltar as informações desejáveis em uma análise.

A quinta etapa apresenta a especificação de um analisador de fluxo de dados estruturados aplicado ao SBTVD e aspectos de sua implementação em uma arquitetura PC e em um terminal de acesso de TV digital.

A sexta e última etapa é composta por uma análise da aplicação do modelo aos protocolos do SBTVD e por avaliações gerais do modelo proposto, ressaltando suas características e aspectos que permitam a criação de trabalhos que derivem deste.

1.3 Contribuições

A análise de elementos em fluxos de dados contínuos requer o uso de técnicas específicas. Diversos trabalhos neste segmento definem mecanismos de amostragem de fluxos de dados que oferecem uma visão consistente do conjunto de dados de entrada. Entretanto, mesmo se tratando de uma amostragem, um grande volume de dados ainda precisa ser agrupado de forma que possa ser avaliado pelo usuário.

Este trabalho busca contribuir com a definição de uma arquitetura simples, porém expressiva, para a análise de fluxos de dados estruturados quaisquer. A solução apresentada permite a avaliação dos elementos capturados em diferentes ambientes computacionais, possibilitando também expressá-los com diferentes níveis de detalhamento.

Em sua aplicação ao Sistema Brasileiro de TV Digital, a contribuição deste trabalho está na geração de uma ferramenta prática que possa ser utilizada em laboratório e que sirva de referência para pesquisadores da área de TV digital.

1.4 Organização do texto

Este texto está estruturado em 7 capítulos. O Capítulo 2 introduz o conceito de fluxo de dados, apresenta técnicas de amostragem de elementos contidos nesses fluxos e discute os principais modelos de exposição e representação de dados estruturados.

O Capítulo 3 apresenta conceitos de televisão digital e do formato do fluxo de dados utilizado nesse meio. São também identificados os elementos de análise desse fluxo, baseados em características encontradas em analisadores comerciais e em pesquisas aplicadas a este tópico.

No Capítulo 4, uma arquitetura genérica para a análise de fluxo de dados estruturados é proposta, compreendendo métodos para a exposição de elementos, o acesso a dados e metadados deste fluxo, a representação de dependências, a comunicação com módulos específicos da arquitetura e o acesso direto ao fluxo de transporte ou a segmentos dele.

A maneira na qual essa arquitetura se aplica ao SBTVD é explicada no Capítulo 5.

O Capítulo 6 introduz o sistema implementado para a validação dessa proposta. Esse capítulo é formado por 3 seções que consistem em considerações

de implementação, métodos para limitar o uso de recursos computacionais e na aplicação prática das propostas conceituais feitas até então.

Finalmente, o Capítulo 7 apresenta as conclusões finais deste trabalho e possíveis trabalhos futuros que possam ser desenvolvidos com base nessa proposta.

2 O Estado-da-Arte nos Mecanismos de Análise de Fluxo de Dados Estruturados

Em sistemas computacionais, é comum o uso de um modelo de transmissão de dados contínuo, denominado fluxo de dados. Nele, a quantidade de informação carregada é tão grande que se torna impraticável a um receptor armazená-la em sua plenitude. Cabe ao receptor, assim, extrair os dados na mesma taxa em que eles são transmitidos, de forma a evitar a perda de informação. Esse modelo é utilizado em diversas áreas da computação, como no tráfego de pacotes IP em uma rede de computadores, na transmissão de música por uma rádio digital, no envio de estatísticas de aplicações financeiras na bolsa de valores, entre muitos outros (CRANOR et al., 2003).

Os elementos encapsulados e transmitidos em um fluxo são geralmente estruturados, ou seja, apresentam uma organização lógica que expõe diversos atributos de cada elemento, constituindo, assim, unidades de informação. Cabe ao receptor extrair os dados de cada unidade e utilizá-los da maneira que lhe for mais adequada.

A natureza de transmissão contínua e o grande volume de dados produzido torna a tarefa de análise das unidades de informação especialmente desafiadora. Os recursos para armazenamento disponíveis em um analisador são menores do que o volume dos dados de entrada, o que requer o uso de técnicas especiais para resumir ao usuário informações relevantes quanto aos elementos encapsulados no fluxo (CORMODE, 2007).

Este capítulo apresenta os modelos de fluxo de dados e técnicas de análise e armazenamento de grandes volumes de informação.

2.1 Modelos de Fluxo de Dados

Fluxos de dados podem ser descritos por dois modelos: fluxos de *entrada* e fluxos de *entrada e saída*. Em ambos, cada unidade do fluxo é descrita por uma tupla em que o primeiro elemento representa um identificador e o segundo um valor associado, como um peso, um contador ou um conjunto de bytes. Dados estes elementos, é possível representar um fluxo de dados como um vetor, indexado pelos identificadores de cada tupla e armazenando valores que indicam a quantidade de vezes que aquele identificador já foi encontrado desde o início do reconhecimento do conteúdo de entrada (MUTHUKRISHNAN, 2005). Um vetor ilustrativo é mostrado na Figura 2.1, em uma situação hipotética em que cada um dos 4 identificadores indexados foi encontrado 2, 3, 4 e 2 vezes, respectivamente.

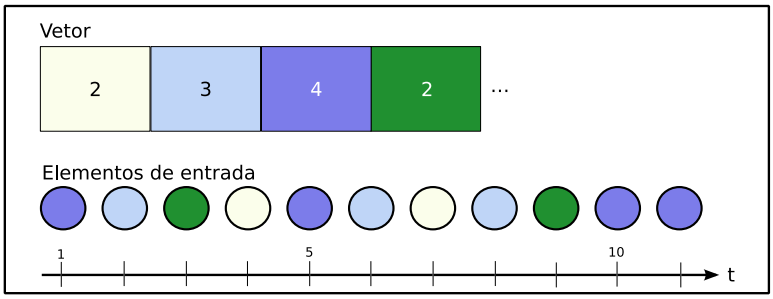

Figura 2.1: Fluxo de Dados de Entrada

O que difere um fluxo de *entrada* de um fluxo de *entrada e saída* é que neste último existe a possibilidade de se substituir elementos já adicionados ao vetor. Esta operação se dá com a subtração do valor previamente armazenado na posição indexada pelo identificador do elemento lido e a inserção de um novo elemento na mesma posição. O modelo de *entrada e saída* pode ser aplicado a um sistema em que, por exemplo, os elementos carregados no fluxo contenham um campo que defina a *versão* da informação trazida. Nesse caso, sempre que uma versão diferir daquela já armazenada no vetor e que o sistema substituir o conteúdo associado àquele elemento, uma operação de *saída* será efetuada. Esse modelo é ilustrado na Figura 2.2: todos os elementos lidos até o tempo $t = 11$ contêm um campo de versão com valor 1. Na chegada de um elemento de versão 2 no tempo $t = 12$, o conteúdo do vetor que o indexa é descartado, dando lugar à nova informação e reiniciando o contador de ocorrências privado (MUTHUKRISHNAN, 2003).

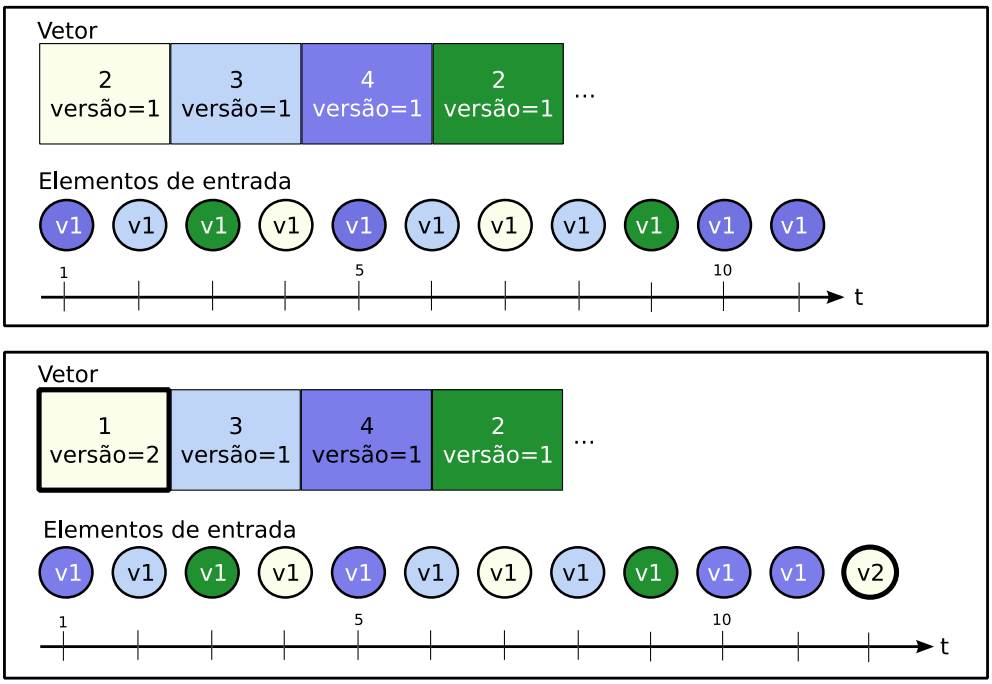

Figura 2.2: Fluxo de Dados de Entrada e Saída

2.2 Técnicas de Amostragem e de Recuperação da Informação

Para lidar com o grande volume de informação presente em um fluxo de dados, muitas vezes se faz necessário utilizar técnicas de amostragem, a fim de limitar a quantidade de memória utilizada pelos algoritmos e de reduzir a carga computacional exigida pela execução destes.

Diversas classes de algoritmos foram propostas para resolver esse problema, como os algoritmos de *Reservoir* e *Min-Wise*, que definem métodos para escolher uniformemente os elementos do fluxo de dados que serão armazenados. Outra técnica desenvolvida foi a de *Sinopse*, que consiste em armazenar um resumo incremental dos elementos lidos. As soluções encontradas por estes algoritmos e os aspectos de cada um são apresentadas nas subseções a seguir.

2.2.1 O Algoritmo Reservoir

Proposto por Vitter (1985), o algoritmo *Reservoir*, literalmente traduzido como *Reservatório*, consiste na escolha de um universo de amostras de tamanho $\geq m$, com base no qual uma amostra aleatória de tamanho m pode ser gerada. O valor de m é tipicamente escolhido de acordo com as restrições de memória e de processamento do receptor do fluxo de dados.

O primeiro passo do algoritmo consiste na inserção dos primeiros m elementos lidos no reservatório. A partir deste ponto, o restante dos elementos é processado sequencialmente, e a chance de cada um deles fazer parte do reservatório (retirando, assim, algum elemento previamente selecionado) é definida por uma função de probabilidade (HANKERSON; JOHNSON; HARRIS, 2003). A probabilidade de que o item i (sendo que $i \geq m+1$) seja escolhido é de $\frac{m}{i}$, uma razão entre o tamanho máximo do reservatório e o número de elementos encontrados até o momento.

A probabilidade de que um elemento qualquer do reservatório seja removido na eventualidade do novo candidato ter sido selecionado para inserção é a probabilidade do elemento i ser selecionado multiplicado pela probabilidade do elemento que já está no reservatório ser escolhido aleatoriamente para dar lugar ao elemento i. Assim, a probabilidade de que um elemento qualquer seja removido é de $\frac{m}{i} \times \frac{1}{m} = \frac{1}{i}$, e a probabilidade de que ele continue no conjunto de amostras é igual a $1 - \frac{1}{i}$.

Durante a recepção do fluxo de dados, diversas substituições são realizadas, alternando os elementos conforme uma probabilidade uniforme para que a amostra final seja composta por elementos escolhidos aleatoriamente. Dessa forma, a probabilidade de que um elemento i se encontre na amostragem final é definida pela probabilidade de i ser amostrado quando ele for encontrado, multiplicada pela probabilidade de que ele sobreviva até o fim, sem ser retirado do reservatório por outro elemento. Para um reservatório de tamanho $m = 1$, a Equação 2.1 mostra a probabilidade de i fazer parte da amostragem final.

$$\frac{1}{i} \times \frac{i}{(i+1)} \times \frac{(i+1)}{(i+2)} \times \cdots \times \frac{(n-2)}{(n-1)} \times \frac{(n-1)}{n} = \frac{1}{n} \tag{2.1}$$

2.2.2 O Algoritmo Min-Wise

O algoritmo *Min-Wise* oferece uma solução simples para a escolha de elementos para amostragem. Para cada item do conjunto de entrada, associa-se um valor aleatório entre 0 e 1, e então escolhem-se os itens com os menores valores para amostragem. Como cada item tem a mesma probabilidade de receber um valor aleatório mínimo, este algoritmo tem uma característica de amostragem uniforme. A Figura 2.3 ilustra uma amostragem segundo este algoritmo, em um cenário com 4 diferentes tipos de elementos.

Outra característica do Min-Wise é que, por não existir interdependência

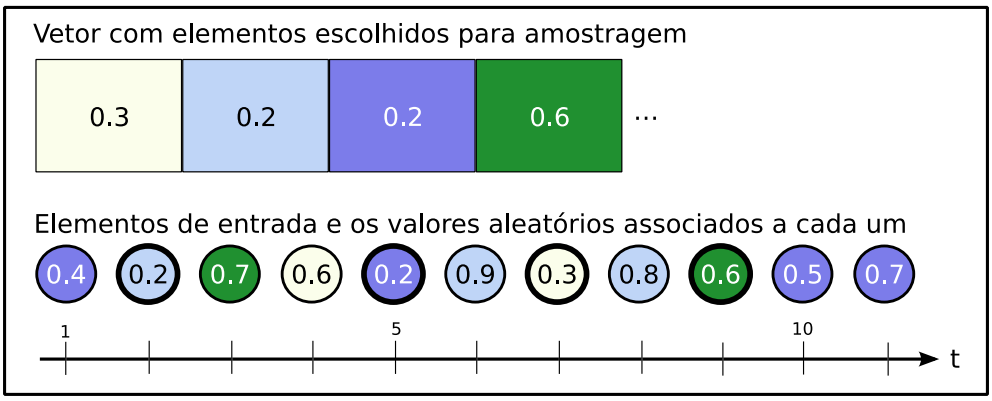

Figura 2.3: Amostragem com o algoritmo Min-Wise

entre os elementos na escolha de um candidato à amostragem, ele permite que diferentes segmentos do fluxo de entrada sejam processados por unidades independentes de processamento; cada uma destas unidades gera um resultado e, ao término, estes resultados são combinados e fornecidos para um último passo do algoritmo (BRODER et al., 2000).

2.2.3 A Técnica de Sinopses

Enquanto os algoritmos de amostragem trabalham com a aquisição de elementos aleatórios do fluxo de entrada, uma outra classe de algoritmos pode processar todos os elementos recebidos. Ainda assim, as estruturas de dados nas quais os elementos são representados constituem meramente um sumário deles, ou uma *sinopse*. Por esse motivo, praticamente nenhuma função que precise ser computada em um elemento armazenado no vetor α pode ser feita com precisão (DIMITROPOULOS et al., 2008). A Figura 2.4 apresenta este conceito por meio de uma função geradora de sinopses que consiste na multiplicação de uma matriz de projeção sobre cada elemento do vetor α.

O algoritmo de *Count-Min* é uma variante deste conceito apresentada por Cormode e Muthukrishnan (2004). Ele considera um vetor α, de dimensão n, onde cada índice armazena um contador que indica o número de vezes que um determinado elemento foi encontrado no fluxo de dados. O estado do vetor no tempo t é definido por $\alpha(t) = [\alpha_1(t), \ldots, \alpha_i(t), \ldots, \alpha_n(t)]$.

Para cada elemento $\alpha_i(t)$ em $\alpha(t)$, o seu estado inicial é $\alpha_i(0) = 0$. A sinopse de i é atualizada sempre que uma nova ocorrência deste elemento ocorrer no vetor α. Neste caso, o valor previamente armazenado é modificado conforme o valor de entrada. Definindo este conceito formalmente, a atualização de i no tempo t

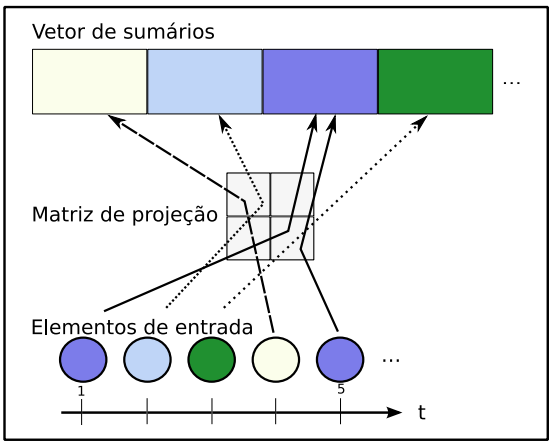

Figura 2.4: Exemplo de geração de uma sinopse

é definida pela tupla (i_t, c_t), de forma que

$$\alpha_{i_t}(t) = \alpha_{i_t}(t-1) + c_t \tag{2.2}$$

e que

$$\alpha'_i(t) = \alpha'_i(t-1), \forall i' \neq i_t. \tag{2.3}$$

A posição em α em que um elemento será inserido é definida por uma função *hash* $h(i)$ escolhida arbitrariamente. Como podem ocorrer colisões, o algoritmo *Count-Min* utiliza, ao invés de um único vetor, uma matriz de tamanho n e profundidade d, sendo que cada uma das linhas de $1..d$ é coordenada por uma função *hash* diferente, igualmente selecionada de forma arbitrária durante a inicialização do algoritmo.

Os valores de n e d são definidos conforme dois parâmetros ε e δ, que definem a precisão esperada e a probabilidade com que se deseja que esta precisão seja alcançada, respectivamente. Assim, n é calculado conforme a função teto $\lceil \frac{e}{\varepsilon} \rceil$ (onde e é a base do logaritmo natural) e d por $\lceil ln(\frac{1}{\delta}) \rceil$. A Tabela 2.1 apresenta as dimensões da tabela para algumas combinações de variáveis.

Tabela 2.1: Exemplo de valores para ε e δ e os valores resultantes de n e d.

$\boldsymbol{\varepsilon}$	$\boldsymbol{\delta}$	$\boldsymbol{n}$	$\boldsymbol{d}$	$\boldsymbol{n \times d}$
0.1	0.1	28	3	84
0.1	0.01	28	5	140
0.1	0.001	28	7	196
0.01	0.1	272	3	816
0.01	0.01	272	5	1360
0.01	0.001	272	7	1904

A inserção e as subsequentes atualizações de um elemento i na matriz são feitas em cada uma das linhas d da matriz, nas colunas indicadas pelo resultado das funções *hash* de cada uma delas. Cada um dos índices retornados pelas funções é incrementado em 1, indicando que uma nova ocorrência daquele elemento ocorreu, conforme mostra a Figura 2.5.

A operação de consulta de um elemento é feita por meio das mesmas funções *hash*. Ele é buscado em cada uma das linhas da matriz, e o menor valor encontrado é retornado como resultado da consulta.

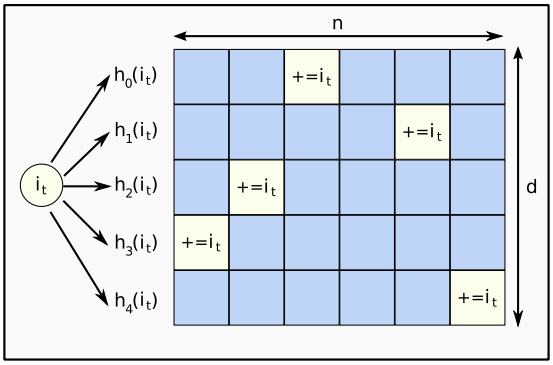

Figura 2.5: Atualização de um elemento no algoritmo de *Count-Min*

2.3 Representação de Dados Estruturados

Para que os elementos de um fluxo de dados capturados possam ser analisados pelo usuário é necessário que eles estejam, primeiramente, armazenados em estruturas de dados que permitam a sua representação efetiva. Naturalmente, a estrutura de dados escolhida está diretamente relacionada ao conteúdo de cada elemento: a representação de uma sequência de notas musicais é provavelmente diferente da sequência de pacotes TCP em uma rede de computadores. Enquanto o primeiro pode ser representado por simples elementos escalares, o último requer o uso de registros mais complexos que representem os diversos atributos que o compõem, como os endereços de origem e de destino, o tamanho do pacote, informações de integridade e a carga útil do pacote. Nesse caso, diz-se que os dados são *estruturados*.

Dependendo da relação entre os atributos de um dado estruturado, a estrutura de dados que o representa pode ainda ser *hierarquizada*, tal como em uma árvore genealógica ou em uma estrutura organizacional. O atributo de maior influência é representado no topo da hierarquia (sua *raiz*), seguido de diversos descendentes conforme os elos lógicos entre os atributos.

A Figura 2.6 mostra como uma hierarquia pode ser organizada para representar um segmento de dados encapsulado em pacotes IP e TCP em uma rede de computadores. Não casualmente a figura se assemelha a uma estrutura de dados em árvore; cada atributo pode ser intuitivamente mapeado a um nodo e a relação entre os atributos pode ser feita por meio de ligações entre os nodos e seus descendentes e ancestrais diretos.

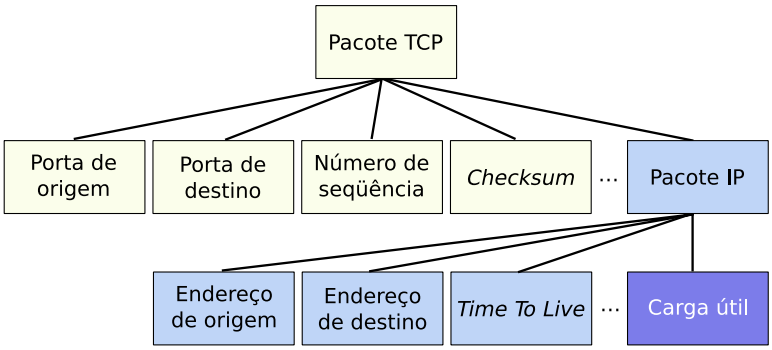

Figura 2.6: Estrutura de Dados Hierarquizada de um Pacote TCP/IP

Diversas técnicas de representação de dados estruturados hierarquizados em árvore foram propostas nos últimos anos. Algumas dessas são apresentadas a seguir, como a adotada no sistema de arquivos de suporte à linguagem de programação MUMPS, em sistemas de arquivos especiais e em documentos XML.

2.3.1 A Linguagem de Programação MUMPS

MUMPS é uma linguagem de programação interpretada desenvolvida na década de 60 no laboratório de animais do Massachussets General Hospital nos Estados Unidos e padronizada pelo instituto ANSI em 1977 (LEWKOWICZ, 1989).

Sintaticamente, MUMPS lida com um único tipo de dados, as *strings*, que podem abrigar até 255 caracteres. Semanticamente, entretanto, a linguagem permite dados de tipos numéricos e vetores multidimensionais, convertidos automaticamente para *strings* por meio de rotinas internas da linguagem.

2.3.1.1 O Banco de Dados Hierárquico em Sistema de Arquivos de MUMPS

Um dos aspectos mais interessantes da linguagem MUMPS é a sua integração nativa com um banco de dados (não relacional) hierárquico no sistema de arquivos. Este banco de dados é usado pela linguagem para armazenar persistentemente os valores de variáveis especiais denominadas *globais*, prefixadas pelo caracter ‘ ^ ’,

após a execução de um programa. Tais variáveis são visíveis e modificáveis tanto pelo programa que as criou quanto por outros programas em execução, o que faz deste o mecanismo de intercomunicação entre processos em ambientes MUMPS.

A declaração de variáveis em MUMPS admite tanto a definição de valores simples quanto hierárquicos, conforme o exemplo na Listagem 2.1:

Listagem 2.1: Declaração de uma variável global em MUMPS

```
SET ^identificador = "2112"
SET ^pacote(PMT, Versão) = "2"
SET versaoPMT = ^pacote(PMT, Versão)
```

Na primeira linha do exemplo é definida uma variável denominada `identificador`. Essa variável é armazenada no nível superior da hierarquia do sistema de arquivos integrado ao MUMPS, em um arquivo denominado `identificador`. O conteúdo desse arquivo é a *string* `2112`. A segunda linha do exemplo atribui o valor `2` à tupla $(PMT, Versão)$ da variável `pacote`. Na terceira linha, o conteúdo dessa tupla é armazenado na variável local `versaoPMT`.

A variável global `pacote` (linha 2 da Listagem 2.1) é representada no sistema de arquivos por uma hierarquia em árvore, na qual a raiz é dada por um arquivo que leva o mesmo nome da variável global, `pacote`. A raiz da árvore contém o descendente direto indexado pelo valor da variável PMT e o descendente indireto indexado pela variável $Versão$ – que receberá a string `2` atribuída no programa. Os descendentes da raiz, no entanto, não são armazenados em objetos distintos no sistema de arquivos. Cada nível na hierarquia, com exceção da raiz, é representado por uma lista de blocos encadeados composta por segmentos e ponteiros para deslocamentos dentro do arquivo que leva o nome do nodo raiz.

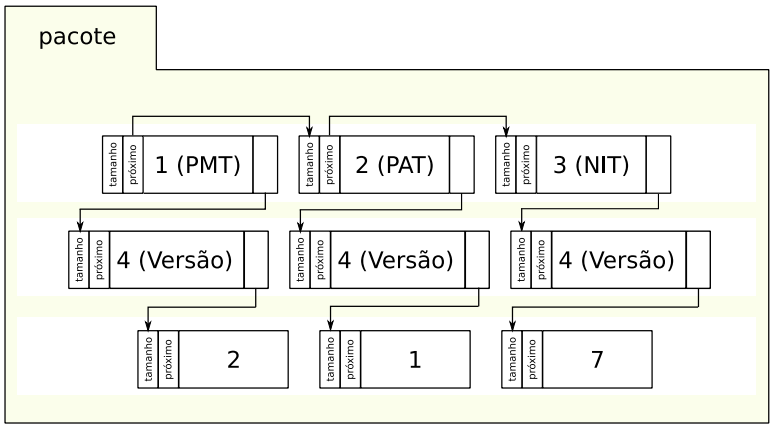

Figura 2.7: Banco de Dados Hierarquizado no Sistema de Arquivos no MUMPS

Um bloco em MUMPS pode assumir um tamanho de 128 a 512 bytes. Tais blocos são compostos por estruturas denominadas *segmentos*, que representam os diversos nodos de uma hierarquia em árvore. Um segmento é composto por um campo inteiro para indexação, uma área de dados de tamanho variável de até 255 bytes e um ponteiro para a subárvore de seu descendente, caso haja um. Os segmentos apresentam ainda um cabeçalho com dois campos, utilizados para indicar o tamanho total do segmento e apontar o deslocamento no arquivo onde o próximo segmento de mesmo nível está armazenado (WEIDERHOLD, 1987).

O armazenamento em disco de uma variável global no MUMPS é mostrado pela estrutura de dados em árvore na Figura 2.7. Nela, os segmentos utilizados para definir os nodos `PMT`, `PAT` e `NIT` são indexados pelos valores 1, 2 e 3, respectivamente, sendo todos eles armazenados em um mesmo bloco. O banco de dados de MUMPS organiza nodos de um mesmo nível em um único bloco de tamanho fixo para otimizar o acesso ao disco, evitando a movimentação excessiva das suas cabeças de leitura.

Nota-se ainda na Figura 2.7 que cada descendente dos segmentos do primeiro nível contém um único nodo, `Versão`, indexado pelo valor 4. Novamente, todos os segmentos deste nível da hierarquia são armazenados em um mesmo bloco de dados. No último nível hierárquico encontram-se finalmente os nodos folha, que contém os valores das n-uplas percorridas.

2.3.2 Sistemas de Arquivos Especiais

Sistemas de arquivos especiais permitem que dados armazenados em mídias voláteis sejam acessados por meio da interface tradicional de sistemas de arquivos (WEINBERGER, 1984). O primeiro sistema de arquivos especial criado denomina-se ProcFS e é explicado a seguir.

O sistema de arquivos *Proc* (ProcFS) surgiu da necessidade de um mecanismo eficiente para a depuração de processos em execução em sistemas operacionais UNIX. Proposto por Killian (1984), a ideia consiste em expor o mapa de memória de processos em execução para que aplicativos especializados possam acessá-los para ler e modificar os estados internos e controlar a execução dos programas. Em sua concepção original, o mapa de memória de um processo era representado por um único arquivo segmentado, cuja criação, remoção e controle de acesso eram gerenciados por um sistema de arquivos especial. Dessa forma, os dados expostos por meio dele não se encontravam fisicamente armazenados em uma mídia não-volátil, mas em memória RAM.

A concepção do ProcFS foi adotada por diversos sistemas operacionais, cada qual o estendeu para permitir a descrição de outros atributos privados do núcleo de execução. Alguns dos sistemas operacionais que implementaram o ProcFS incluem o IRIX (SGI, 1996), o Plan 9 (PIKE et al., 1990), o Solaris (MCDOUGALL; MAURO, 2007) e o Linux (BENVENUTI, 2005).

A representação dos processos em execução deixou de ser feita por meio de um único arquivo por processo a partir da versão 2.6 do Solaris, um sistema operacional desenvolvido pela Sun Microsystems. Nele, as informações referentes ao mapa de memória de um processo em execução passaram a ser estruturadas e disponibilizadas como diversos arquivos dentro de uma hierarquia de diretório específica para aquele processo. Essa hierarquia inclui também subdiretórios com arquivos de controle para cada processo leve (*thread*) criado a partir do processo principal, permitindo a inspeção detalhada de seus atributos específicos.

A flexibilidade oferecida pela exposição de metadados de processos por meio de um sistema de arquivos é explorada também na implementação do ProcFS no Plan 9, um sistema operacional desenvolvido pelo Bell Labs. Nele, é possível acessar a estrutura de diretórios ProcFS de um computador remoto e depurar, a distância, um processo em execução no outro computador (PIKE et al., 1992).

2.3.2.1 Aspectos de Implementação

Para tornar possível a exposição de metadados de processos como arquivos é necessário que o sistema operacional mantenha as informações referentes aos processos em estruturas que possam ser acessadas eficientemente. A especificação dos arquivos que irão compor um diretório virtual é geralmente feita por meio de uma interface de programação (API) que permita o cadastro de uma variável, do nome que a representará no sistema de arquivos e de um conjunto de funções que irão tratar requisições de leitura, escrita e de envios de comandos sobre o arquivo. Finalmente, faz-se necessário integrar as rotinas de criação e destruição de processos com o sistema de arquivos de forma que ele represente corretamente o estado atual dos processos em execução no computador (SINGH, 2006).

No caso do ProcFS, os arquivos exportados para a árvore virtual não expõem apenas o conteúdo de determinados atributos de um processo; eles permitem também que o processo seja controlado e depurado por um outro programa – situação esta que pode exigir a interferência de outros subsistemas do núcleo de execução. A exemplo da implementação original do ProcFS na 8^a Edição do UNIX, a dependência de outros subsistemas é bastante grande:

- Quando uma operação de controle sobre um arquivo no ProcFS é detectada pelo sistema de paginação[1], o processo que será controlado é marcado como um potencial candidato a ter seus dados movidos para a memória secundária: na eventual falta de espaço livre na memória principal, um processo em depuração tem menor prioridade na ocupação da memória em relação aos demais;

- A cada solicitação de leitura de estado de variáveis ou de incremento no passo de execução do programa, o sistema de paginação recupera os dados solicitados que podem se encontrar na memória secundária e os disponibiliza na memória principal;

- A cada operação de controle feita sobre o arquivo, o escalonador do núcleo de execução alterna o processo em depuração entre as listas de tarefas aptas a executar e de tarefas em espera para execução.

Nota-se, entretanto, que uma integração semelhante a essa também seria necessária caso um outro mecanismo que não o de arquivos fosse usado para permitir o controle de um processo por outro. Essa integração mostra, no entanto, que a exposição de estruturas de dados privadas de um sistema operacional ou de uma aplicação específica pode requerer a coordenação de diversos componentes para garantir a eficiência no uso dos recursos computacionais providos no ambiente.

2.3.3 Objetos Textuais em XML

A *eXtensible Markup Language* (XML) é uma especificação de propósito geral para a criação de linguagens de marcação, que consiste em um conjunto de símbolos e palavras que descrevem trechos de um documento. A especificação do XML foi feita pelo Consórcio W3 (W3C, 1994) em 1996 que tinha, entre os seus objetivos principais, a criação de um formato de representação de dados que fosse facilmente compreendido quando lido diretamente por pessoas e que pudesse ser adotado amplamente na Internet. Sua especificação foi feita a partir de um subconjunto da linguagem SGML (ISO, 1996), de forma que um documento XML é também um documento SGML válido.

Um documento XML é composto fisicamente por unidades denominadas *entidades*, que são representadas por uma *raiz* que pode referenciar outras entidades, e assim sucessivamente. Logicamente, os elementos de um documento XML são

[1] O sistema de paginação é responsável por movimentar segmentos de dados entre a memória principal e uma memória secundária

descritos entre dois marcadores balanceados que delimitam o seu início e fim. Cada elemento contém um tipo, é identificado por um nome e pode ter um conjunto de atributos no formato $(nome, valor)$.

A estrutura de um objeto em XML é, dessa forma, descrita por uma série de elementos, cada qual podendo descrever outros elementos em uma hierarquia em árvore. O exemplo ilustrado anteriormente na Figura 2.7 pode ser descrito em XML em uma construção similar à indicada na Listagem 2.2.

Para possibilitar a realização de consulta a nodos, valores e atributos, que podem ser representados em formatos textuais ou binários, a hierarquia em árvore definida no documento deve ser reconstruída. Essa tarefa é realizada por um *processador XML*, geralmente encapsulado na forma de uma *biblioteca* de funções para que aplicações possam compartilhar objetos segundo esse formato ou integrado a linguagens de programação especializadas (KAMINA; TAMAI, 2002).

Listagem 2.2: Declaração de um objeto em XML

```
<pacote>
    <PMT>
        <Versão>2</Versão>
    </PMT>
    <PAT>
        <Versão>1</Versão>
    </PAT>
    <NIT>
        <Versão>7</Versão>
    </NIT>
</pacote>
```

Por tornar possível a descrição de estruturas ricas e complexas, o formato XML é especialmente importante para o armazenamento de estruturas de dados em disco. A especificação do formato OpenDocument (ODF), padronizado pela organização ISO/IEC (ISO/IEC, 2006), adota o XML para esse fim, utilizando-o como base para a declaração de documentos de processamento de texto, planilhas, bancos de dados orientados a objetos, apresentações, gráficos e de equações matemáticas.

2.4 Síntese

Fluxos de dados são empregados por diversos sistemas computacionais em modelos de transmissão de dados contínuos, potencialmente infinitos. Esse tipo de transmissão torna especialmente difícil a tarefa de análise dos elementos trafegados em tais fluxos, visto que o espaço de armazenamento de um receptor é geralmente limitado. Dessa forma, a análise de fluxos de dados requer o uso de técnicas que amostrem ou sintetizem o conteúdo dos elementos nele trafegados.

A representação de elementos estruturados transmitidos nos fluxos de dados pode ser feita por meio de estruturas em árvore, e a sua exposição pode ser feita de diferentes maneiras para o usuário. Três técnicas de exposição de informações estruturadas foram apresentadas nesse capítulo – bancos de dados hierarquizados, sistemas de arquivos especiais e documentos textuais estruturados –, nas quais aspectos importantes quanto à portabilidade e acesso à informação podem ser observados.

A estrutura de diretórios no sistema de arquivos Proc permite o acesso a metadados de processos de maneira simples e bem conhecida entre usuários de computadores: a estrutura de dados é o próprio sistema de arquivos. Nenhuma aplicação especial é necessária para acessar a informação contida em um arquivo.

A organização lógica com delimitadores balanceados em arquivos textuais adotada por documentos XML permite a declaração de hierarquias complexas que podem ser lidas e criadas tanto diretamente pelos usuários quanto por processadores XML, ainda que possam gerar documentos longos e de difícil leitura; o formato de descrição de elementos em um documento XML, entretanto, permite sua portabilidade entre diferentes sistemas computacionais.

Embora os atributos de um elemento estruturado não estejam completamente expostos no banco de dados integrado à linguagem de programação MUMPS, eles podem ser consultados por meio de uma abstração da linguagem. O formato do arquivo gravado em disco, no entanto, requer o uso de uma aplicação especializada para reconstituir a hierarquia e pode ainda impedir a reconstrução dos dados caso um arquivo esteja parcialmente corrompido.

Esses aspectos serão retomados no próximo capítulo, na definição de um mecanismo de uma arquitetura para análise de fluxo de dados estruturados, que é um caso especial de fluxo de dados na qual a análise de seus dados estruturados requer o uso de técnicas de exposição de informação eficientes.

3 Sistemas de TV Digital

Este capítulo apresenta os conceitos de TV digital aplicados neste trabalho. A sua organização é feita em duas grandes partes que compreendem, primeiramente, as características fundamentais de um sistema de TV digital e, em seguida, o conceito de fluxo de transporte, que identifica a relação dos dados trafegados por meio dele com elementos estruturados hierarquizados.

3.1 TV Digital

A televisão é ainda hoje um dos meios mais importantes de comunicação. Por muito tempo, a informação foi transmitida pelas emissoras e recebida pelos telespectadores apenas por meio de canais e tecnologias analógicas, que apresentavam muitas vezes uma alta taxa de ruído e erros. Estes problemas eram facilmente observados com a exibição de imagens com baixa nitidez e uma má qualidade de áudio e de legendas (YAMADA et al., 2004).

O sistema de TV digital surgiu para corrigir estes problemas fundamentais. Todavia, ele traz não apenas uma melhoria significativa na qualidade do sinal de áudio e vídeo, mas também a possibilidade de interoperabilidade com outras tecnologias digitais e a interação com o telespectador.

Estas melhorias foram possíveis graças aos novos métodos de modulação, que definem processos nos quais o sinal original é codificado para ser enviado pelo meio físico. Tal processo de modulação permite a identificação e a correção de erros, possibilitando uma recepção livre de interferências audiovisuais.

Além disso, as tecnologias empregadas no sistema de TV digital utilizam meios de compressão de dados, de forma a aproveitar melhor o espectro de transmissão reservado para as emissoras e ainda assim apresentando um conteúdo de áudio e vídeo com maior definição. Tais meios de compressão incluem o uso de formatos como o definido na norma do MPEG-2 (ISO, 1996) e do H.264 (ITU-T, 2007), usados para a transmissão de conteúdos em definição simples (SDTV) e

alta (HDTV), respectivamente. Em um espectro de 6 MHz, utilizado hoje para a transmissão de um único canal de TV analógico, o uso desses meios de compressão permitem que uma emissora transmita até quatro programas em definição simples ou dois em alta definição (CARVALHO, 2008).

3.1.1 A Arquitetura de Transmissão e Recepção de TV Digital

Um cenário de transmissão de TV digital é composto por duas entidades: um transmissor e um receptor, representado por um terminal de acesso que permite a apresentação dos programas emitidos pelo primeiro.

Cabe ao transmissor codificar os sinais de áudio e vídeo produzidos em estúdio com os algoritmos de compressão estipulados pelo sistema de TV digital vigente. Desse processo resultam *fluxos elementares* de áudio e vídeo devidamente combinados de forma que sua reprodução pelo receptor seja síncrona. Podem ainda ser somados a esses fluxos elementos como legendas, programas interativos e metadados diversos, igualmente sincronizados (caso necessário) com um relógio de sistema. Esses fluxos são então multiplexados em pacotes de tamanho fixo, propriamente identificados por meio de tabelas de informação de serviço, que indicam ao receptor como recompor esses elementos. O processo de multiplexação resulta em um *fluxo de transporte* que é modulado e transmitido por meio de radiodifusão para os diversos receptores (PEREIRA et al., 2005).

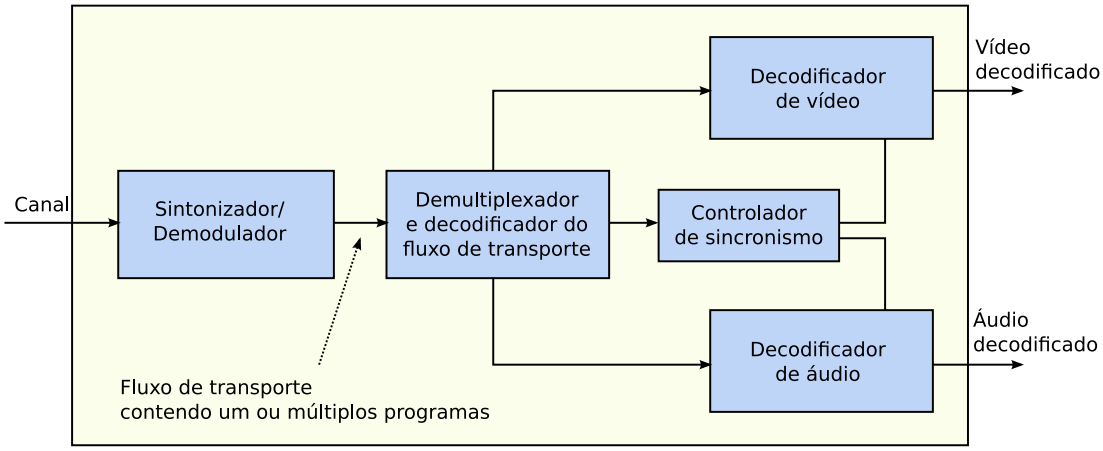

Figura 3.1: Arquitetura de recepção e decodificação sincronizada de TV digital

Na arquitetura do receptor, o sinal transmitido é recebido por um sintonizador-demodulador, também chamado *Front End*. Esse componente é responsável pela sintonia, redução de ruídos, demodulação e recomposição do sinal recebido na

antena, que é entregue para o sistema digital de demultiplexação na forma de um fluxo de transporte (BEDICKS JR, 2008). Conforme a indicação das tabelas de informação de serviço, um aplicativo residente no terminal de acesso determina os fluxos de áudio e vídeo que serão enviados aos decodificadores e, com base nas estampas de tempo presentes no fluxo, apresentados ao usuário. Essa arquitetura é contemplada na Figura 3.1.

3.1.2 O Fluxo de Transporte do MPEG-2

O fluxo de transporte é um protocolo de transmissão de dados definido pela norma ITU-T H.220.0 para o padrão MPEG-2 (ISO, 1996). Projetado para permitir o transporte de conteúdo de vídeo, áudio e dados sincronizados, o fluxo de transporte do MPEG-2 foi adotado como meio de encapsulamento de dados nos padrões de TV digital ISDB (ARIB, 2008), DVB (ETSI, 1997), ATSC (ATSC, 2007) e SBTVD (ABNT, 2008a), vigentes no Japão, Europa, América do Norte e Brasil, respectivamente.

A composição do fluxo de transporte é feita por pacotes consecutivos de 188 bytes denominados TS (do inglês *Transport Stream*), cada qual abrigando um segmento dos fluxos elementares de áudio, vídeo, dados ou de tabelas de informação de serviço – estas últimas definidas pelo MPEG-2 e estendidas pelo padrão de TV digital específico. Para indicar o seu conteúdo, um cabeçalho ocupa os primeiros 4 bytes de cada pacote, que inclui um byte de sincronismo (0x47), o número identificador do pacote (PID), sinalização de erro e sua prioridade, entre outros. O cabeçalho do pacote pode ainda ser sucedido por um campo de tamanho variável denominado campo de adaptação que, caso seja transmitido, ocupa parte dos 188 bytes reservados à carga útil (CHERNOCK et al., 2001). Essa estrutura é mostrada na Figura 3.2.

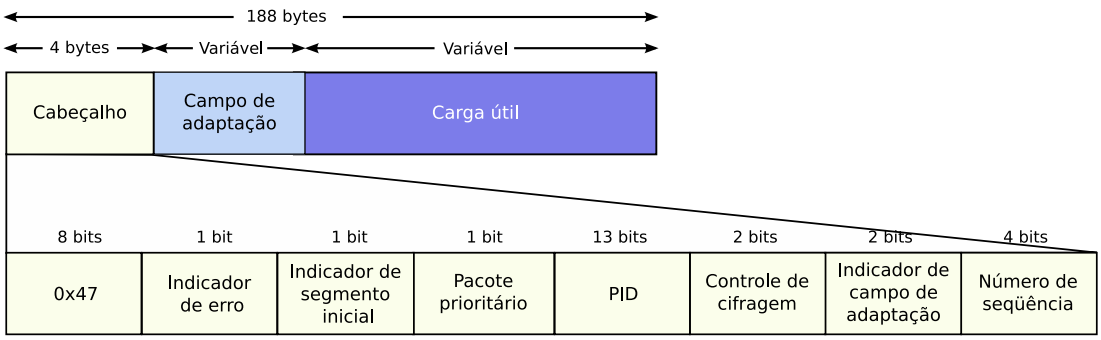

Figura 3.2: Estrutura de um pacote de fluxo de dados do MPEG-2. (Os campos de bits são representados com o bit mais significativo à direita.)

3.1.2.1 Tabelas de Informação de Serviço

As tabelas de informação de serviço definidas no MPEG-2 são obtidas a partir do número identificador dos pacotes (PID) e processadas pelo *software* residente no terminal de acesso para recuperar o nome da emissora e a programação contida no fluxo de transporte. A transmissão dessas tabelas ocorre de maneira cíclica, em uma taxa determinada pela norma de TV digital. Dessa forma, um terminal de acesso recém ligado não requer muito tempo de espera para identificar o conteúdo e iniciar a apresentação da programação.

Entre as tabelas de informação definidas no MPEG-2, três são de uso mandatório no Sistema Brasileiro de TV Digital (ABNT, 2007a):

1. Tabela de Informação de Rede (NIT): contém o nome da emissora, o código de área que define a região na qual a transmissão se destina e os dígitos no controle remoto que devem acessar o seu serviço principal, entre outros;
2. Tabela de Mapeamento de Programas (PMT): indica o PID dos pacotes que carregam os fluxos elementares de áudio e vídeo que compõem um *programa*, além do PID dos pacotes que descrevem o tempo de decodificação e apresentação destes. Um programa pode ser composto por mais de um canal de áudio e vídeo, possibilitando sua reprodução em diversas línguas e ângulos. Um fluxo de transporte que transmita dois programas apresenta duas tabelas PMT para descrever os PIDs que compõem cada um deles;
3. Tabela de Associação de Programas (PAT): indica o PID das tabelas PMT que contém programas habilitados para reprodução e o PID da tabela de informação de rede (NIT). A tabela PAT tem um número de identificação fixo (0x00), o que torna possível a reconstrução de toda a árvore de programação a partir de um pacote desse tipo.

O relacionamento entre essas tabelas pode ser observado na Figura 3.3. Nela, a tabela PAT indica que o fluxo de transporte é composto por uma tabela NIT, de PID 0x10, e por dois programas, de PIDs 0x21 e 0x22, cada qual descrevendo os identificadores dos pacotes de áudio, vídeo e de sincronismo (PCR) que o compõem. O programa de PID 0x22, embora presente no fluxo de dados, não foi habilitado pelo transmissor para sua exibição – logo não deve ser processado, tampouco considerado para decodificação, pelo terminal de acesso.

Nota-se também na Figura 3.3 a existência de dois campos denominados `table_id` e `version`, ambos presentes na carga útil de cada pacote. O primeiro

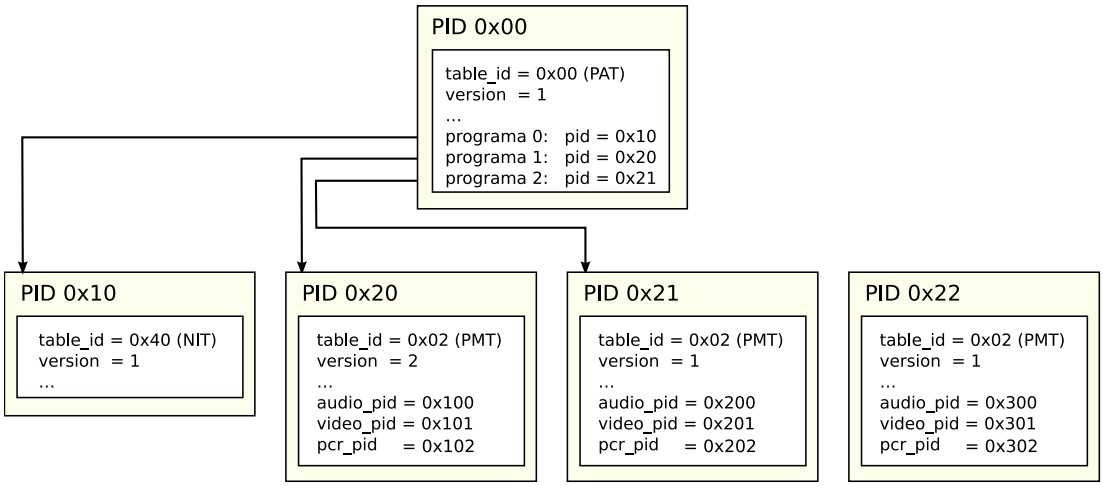

Figura 3.3: Relação entre as tabelas de informação de serviço NIT, PMT e PAT

é usado para identicar o tipo de tabela trazida no pacote. Esse campo se faz necessário tendo em vista que apenas alguns tipos de tabela têm seu PID fixado pela norma. O segundo campo indica a versão da tabela transmitida. Ele é incrementado sempre que houver uma atualização no conteúdo de uma tabela, situação essa que irá requerer uma nova interpretação de seu conteúdo por parte do terminal de acesso. Um exemplo típico de atualização de versão ocorre na troca de programação de uma emissora, que passa a transmitir uma nova composição de elementos de áudio e vídeo com PIDs diferentes do programa anterior.

Além das tabelas PMT, PAT e NIT, diversas outras tabelas de informação de serviço são compreendidas no SBTVD. Em comum, todas contêm um cabeçalho com membros que indicam a versão da tabela, seu tamanho e seu identificador, entre outros. Algumas dessas tabelas apresentam, também, um conjunto opcional ou mandatório de descritores, que trazem informações adicionais como a codificação do áudio e vídeo, restrições parentais, logotipo da emissora, relógio de referência, taxa de bits e zona de fuso horário, essenciais para que um terminal de acesso programe corretamente seus decodificadores e respeite as classificações indicativas da programação.

3.1.2.2 Encapsulamento de Fluxos Elementares de Áudio e Vídeo

No SBTVD, a formatação dos fluxos elementares de áudio e vídeo que compõem um programa é feita de acordo com as especificações de codificação do H.264 e do HE-AAC v2, respectivamente. Como estes fluxos apresentam uma dependência temporal entre si, é necessário que sua transmissão traga informações adicionais

que permitam identificar os tempos de decodificação e apresentação, de forma que sua reprodução seja sincronizada. Assim, o transporte dos pacotes que trazem segmentos de áudio e vídeo não é feito apenas pelos pacotes TS de 188 bytes; um outro protocolo denominado PES (*Packetized Elementary Streams*) é adotado e usado para encapsular esse conteúdo antes que ele seja mapeado aos pacotes TS do fluxo de transporte.

Em pacotes PES também incluem-se informações relativas ao relógio do transmissor. Essa informação é utilizada pelo receptor para que a decodificação dos fluxos de áudio e vídeo seja feita com sincronia. Embora segmentado em pacotes de tamanho fixo, a extensão de um pacote PES completo é variável, correspondendo, tipicamente, ao tamanho de um quadro de áudio ou de vídeo. Naturalmente, a carga útil de pacotes de áudio e vídeo não apresenta um campo de versão, pois todos os pacotes transmitidos devem ser decodificados pelo receptor.

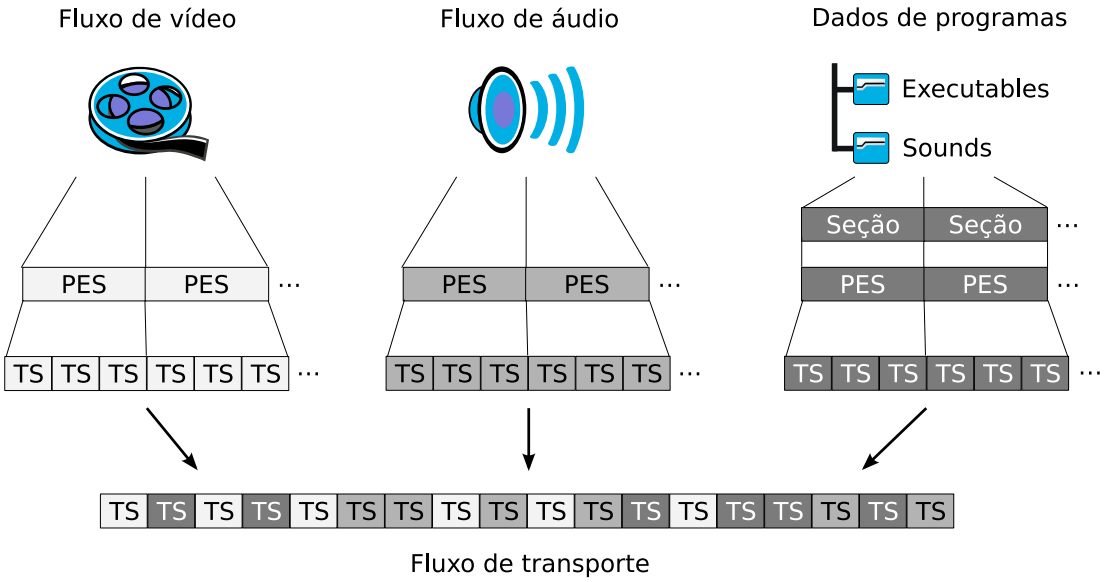

Figura 3.4: Encapsulamento de vídeo, áudio e dados no fluxo de transporte

O processo de conversão de fluxos elementares para pacotes TS é mostrado na Figura 3.4. O posicionamento de cada pacote TS no fluxo de transporte é decidido por um algoritmo de escalonamento, que considera as taxas de transmissão de tabelas mínimas exigidas pela norma de TV digital para definir a ordem temporal de cada pacote (GIRÃO et al., 2005). A figura mostra também como é feita a tradução de dados de programas em pacotes TS; maiores detalhes são apresentados no tópico a seguir.

3.1.2.3 Tabelas de Transmissão de Dados

A transmissão de elementos de dados que não requerem sincronia com outros fluxos é feita por meio de uma família de protocolos denominada DSM-CC, do inglês *Digital Storage Media Command and Control* (Comandos e Controles de Armazenamento de Mídias Digitais) (ISO, 1996). Definido pela ISO/IEC e parte da família de padrões do MPEG, o DSM-CC foi originalmente concebido para suportar a entrega de vídeo sob demanda em redes com um canal de comunicação bidirecional, mas acabou sendo estendido para atender a redes unidirecionais. O DSM-CC é utilizado no contexto de TV digital para a transmissão cíclica de dados, tal como atualizações de *software* enviadas pelos fabricantes de terminais de acesso pelo fluxo de transporte ou aplicações interativas que são interpretadas por um *Middleware*. Este tipo de transmissão recebe o nome de *carrossel* (ABNT, 2007b).

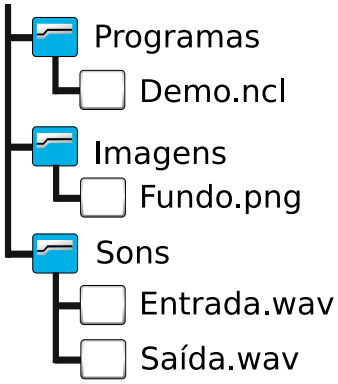

Figura 3.5: Exemplo de estrutura de diretórios transmitida em um carrossel de objetos

Dois tipos de carrosséis são definidos pelo protocolo: carrosséis de dados e de objetos. No primeiro, o conteúdo dos dados transmitidos não é especificado; o receptor precisa ter conhecimento de como tratá-los. É por meio dele que são enviadas as atualizações de *software* dos terminais de acesso, cujos dados só dizem respeito a uma família específica de aparelhos. Os carrosséis de objetos, por outro lado, contêm dados identificáveis, como imagens, arquivos de texto, música, ou de aplicações, transmitidos juntamente com uma estrutura de diretórios, que indica a organização lógica dos objetos transmitidos, como no exemplo mostrado na Figura 3.5.

A formatação dos dados no carrossel de objetos é descrita por mensagens que seguem um protocolo denominado BIOP. As mensagens BIOP consistem em um cabeçalho, que contém informações como o tamanho da mensagem e a versão do protocolo na qual ela está codificada, e um subcabeçalho, que descreve parâmetros

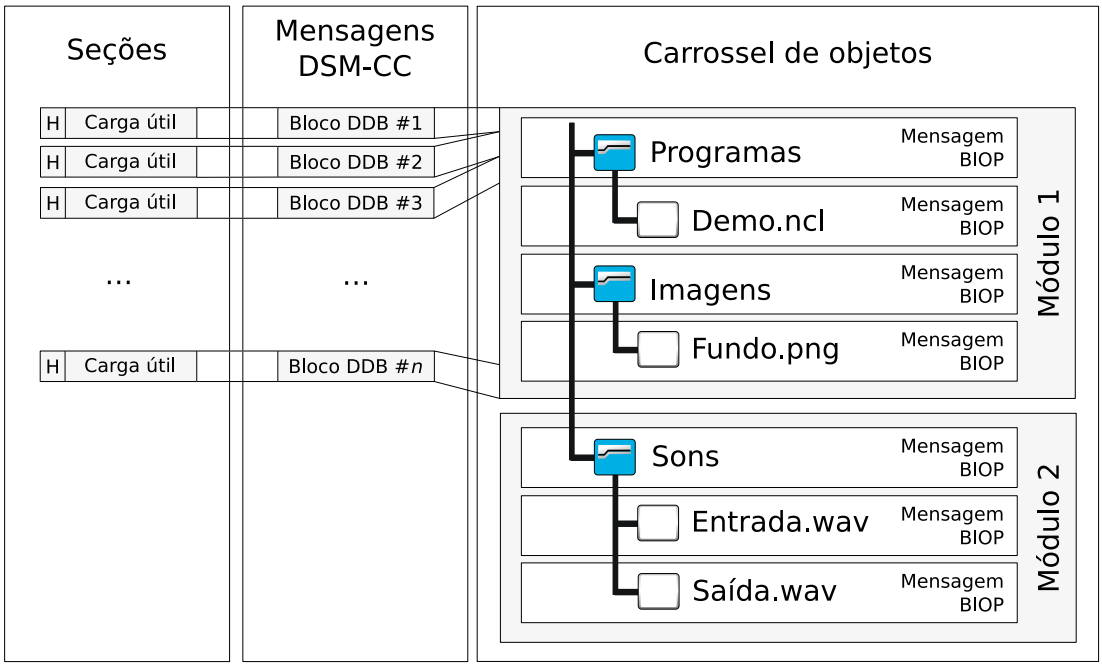

Figura 3.6: Composição de uma mensagem BIOP

específicos do tipo de objeto na mensagem. Em um objeto que representa um diretório, por exemplo, o subcabeçalho contém os nomes dos elementos contidos naquele diretório.

Uma mensagem BIOP é composta por módulos de tamanho máximo de 64 KiB. Cada um desses módulos divide-se subsequentemente em outros blocos menores, denominados *blocos de dados*, que são encapsulados em *mensagens de download* de acordo com um protocolo denominado DDB (*Download Data Block*, ou Bloco de Dados de *Download*). Cada mensagem DDB é encapsulada, por sua vez, em unidades denominadas *seções*, que são finalmente distribuídas em pacotes do fluxo de transporte de 188 bytes (Figura 3.6). Para identificar os blocos que compõem um determinado conteúdo, o DSM-CC envia mensagens denominadas DII (do inglês *Download Information Indication*, ou Indicação de Informação de *Download*), que descrevem os blocos válidos de maneira similar à forma como a tabela PAT indica a composição dos programas válidos de uma emissora (INFANTE; NASIOPOULOS, 2008).

3.2 Elementos de Análise em um Fluxo de Transporte de TV Digital

A informação contida nas tabelas de serviço e em seus descritores é essencial para a correta apresentação dos programas transmitidos pela emissora. Logo, é de suma importância validar seu conteúdo em um processo de análise do fluxo de transporte. A listagem a seguir indica os elementos de análise mais comuns em produtos comerciais, por serem algumas das principais fontes de erro no fluxo de transporte, bem como funcionalidades adicionais que se mostram bastante úteis no auxílio à identificação de problemas:

- Identificação das tabelas de informação de serviço (SI) do MPEG-2. A interpretação dos campos que compõem as tabelas SI permite a identificação de parâmetros incorretos ou não suportados por uma implementação específica;
- Identificação das tabelas de informação de serviço estendidas e de descritores definidos pelo DVB, ISDB, ATSC ou SBTVD (os valores que alguns descritores podem assumir podem não ser válidos em todos esses padrões). As tabelas e descritores mandatórios e opcionais especificados por uma norma de transmissão devem seguir uma formatação que, caso esteja fora da especificação, pode levar à exibição de informações incorretas para o usuário ou mesmo impedir a decodificação de fluxos elementares de áudio e vídeo;
- Decodificação e exibição de dados do Guia de Programação Eletrônico (EPG). Essa informação permite que se possa validar os resumos e tempo de duração da programação enviados pela emissora;
- Decodificação e exibição de dados de legenda e *closed caption*. O protocolo de legenda especifica uma série de informações quanto a disposição do texto e rolagem automática que devem ser respeitados por um terminal de acesso para que esse recurso não apresente problemas visuais indesejáveis. Alguns analisadores permitem a decodificação desse protocolo para auxiliar na depuração de problemas associados a esse recurso;
- Decodificação e exibição de carrosséis de dados e objetos. A atualização do *software* residente no terminal de acesso e a execução de conteúdos interativos requer a recepção e armazenamento dos objetos transferidos pela pilha de protocolos do DSM-CC. O acesso ao conteúdo de cada arquivo e di-

retório transmitidos auxilia na identificação de erros que podem bloquear a utilização de um terminal de acesso ou interferir no seu bom funcionamento;

- Separação e gravação individual dos fluxos elementares de áudio e vídeo. Alguns aplicativos são especializados na análise de fluxos elementares, compreendendo diversos protocolos de decodificação. A extração e a gravação dos fluxos elementares de áudio e vídeo em disco permitem que eles sejam analisados separadamente;

- Decodificação dos fluxos de áudio e vídeo presentes no fluxo de transporte analisado. Esse recurso permite que um usuário tenha acesso e possa reproduzir individualmente um fluxo de áudio ou vídeo e comparar a qualidade de reprodução àquela observada em um terminal de acesso;

- Identificação da taxa de transmissão das tabelas de informação de serviço (SI). As normas de TV digital especificam a taxa de transmissão mínima de algumas tabelas SI. O atraso na recepção de uma tabela SI pode impedir a identificação de um serviço durante a varredura de frequências ou desabilitá-lo durante uma troca de canais. A validação das taxas de transmissão das tabelas SI é, dessa forma, bastante importante para a resolução desse tipo de problema;

- Validação do CRC presente em cada pacote. Interferências físicas, o mau funcionamento de um *software* controlador de dispositivo ou mesmo a transmissão de dados em uma taxa acima daquela suportada pelo padrão de TV digital podem levar a falhas de integridade de pacotes do fluxo de transporte. A verificação do CRC de pacotes recebidos pode, assim, indicar a presença de algum problema físico ou lógico no lado do transmissor ou receptor;

- Listagem de erros presentes em cada PID. Todo o uso incorreto de parâmetros ou erro detectados em um PID são agrupados e listados para que o usuário tenha um resumo dos defeitos de cada elemento analisado.

Alguns produtos permitem também detectar a inconformidade de um fluxo de transporte por meio da análise cruzada de tabelas. Matthew (2004a) descreve um mecanismo baseado em uma lista, populada a partir dos programas e identificadores listados na tabela PAT. A seguir, o fluxo de transporte é varrido em busca das tabelas PMT presentes naquela lista. Caso alguma tabela PMT não seja encontrada e nenhuma nova versão da tabela PAT seja recebida para informar que

aquela PMT não se encontra efetivamente no fluxo de transporte, uma condição de erro é gerada. (MATTHEW, 2004b) também descreve uma metodologia similar, utilizada pelos sistemas analisadores de fluxo de transporte da Tektronix, para constatar se todos os programas identificados na tabela de descrição de serviços (SDT) foram transmitidos no fluxo de transporte [1].

Esses elementos de análise são encontrados em produtos comerciais como (ZANALYZER, 2008; DEKTEC, 2006) e (ELECARD, 2006), que os apresentam ao usuário em interfaces criadas especialmente para essa finalidade. A Figura 3.7 mostra a composição típica de um aplicativo de análise de fluxo de dados: uma listagem exibe as tabelas reconhecidas, que podem ser expandidas em diferentes níveis hierárquicos para mostrar a informação com mais detalhes. Nesta figura também nota-se a representação da informação por meio de gráficos, usados para auxiliar o usuário a compreender determinados eventos no fluxo de transporte. Para a análise de erros, o aplicativo pode indicar valores incorretos ou inesperados por meio do realce de cores.

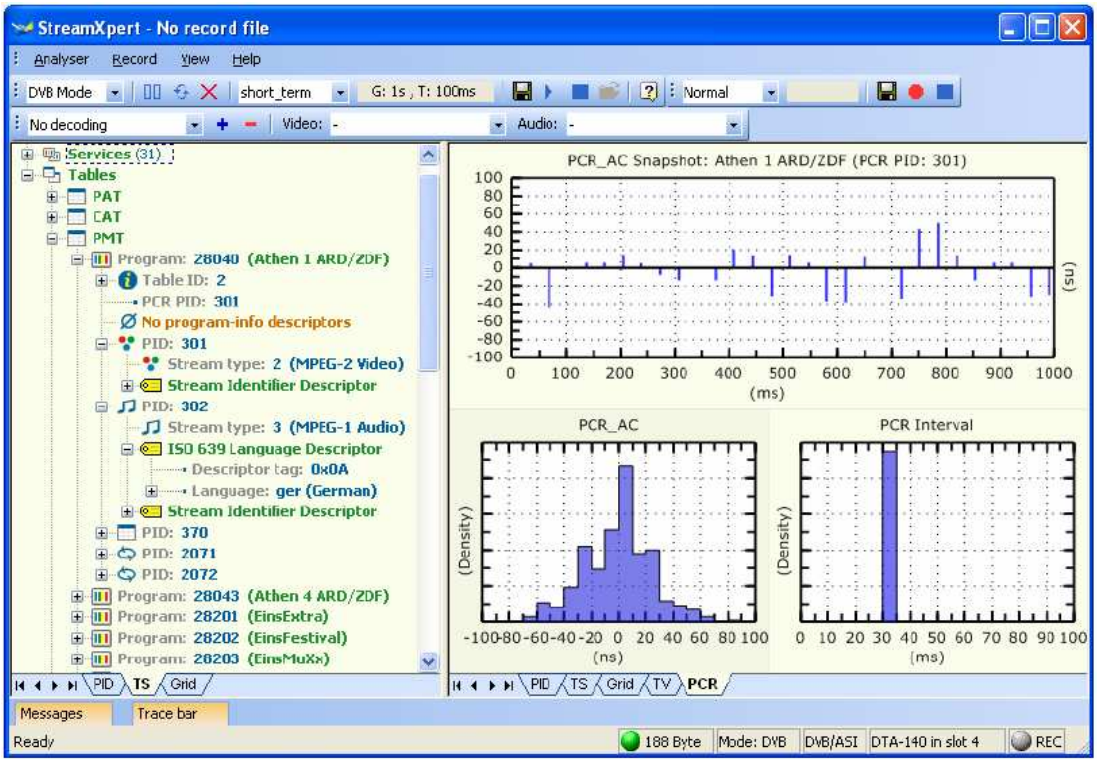

Figura 3.7: Aplicativo StreamXpert, da Dektec

Um outro aplicativo de análise popular entre engenheiros de TV digital é o DVBsnoop (SCHERG, 2001), um *software* livre desenvolvido para a análise de

[1] Esses pedidos de patente não foram aceitos pelo Escritório de Patentes Europeu (OFFICE, 2004).

fluxos de dados codificados conforme o padrão europeu (DVB) que utiliza uma interface em modo texto para a exposição dos resultados capturados. Assim como o o StreamXpert, o DVBsnoop apresenta um recurso bastante desejável que não se encontra em muitas soluções proprietárias: a captura e análise do fluxo de transporte em tempo real.

3.3 Síntese

Este capítulo apresentou os principais elementos da arquitetura de transmissão de TV digital e o modelo de transmissão de informação em elementos estruturados. O fluxo de transporte do MPEG-2 utilizado por sistemas de transmissão de TV digital representa um caso especial de fluxo de dados. Dois tipos de elementos trafegam nesse fluxo: tabelas de informação de serviços, que são tratadas pelo receptor caso a versão da tabela não seja a mesma daquela previamente recebida, e fluxos elementares de áudio, vídeo e dados que, em razão de sua vazão, não podem ser armazenados plenamente na memória do analisador.

Os aspectos apresentados neste capítulo são importantes para a compreensão de como a proposta da arquitetura de análise de fluxo de dados, apresentada no próximo capítulo, se aplica ao Sistema Brasileiro de TV Digital (Capítulo 5).

4 Uma Arquitetura para Análise de Fluxo de Dados Estruturados

A arquitetura proposta para a análise de um fluxo de dados compreende 5 componentes principais: (*i*) a interface de captura de dados, por onde a leitura de elementos é realizada; (*ii*) um módulo destinado à escolha dos elementos que serão processados; (*iii*) uma camada responsável pelo processamento e geração de metadados dos elementos escolhidos; (*iv*) uma estrutura para o armazenamento dos resultados e (*v*) uma interface que disponibilize os resultados produzidos ao usuário (Figura 4.1).

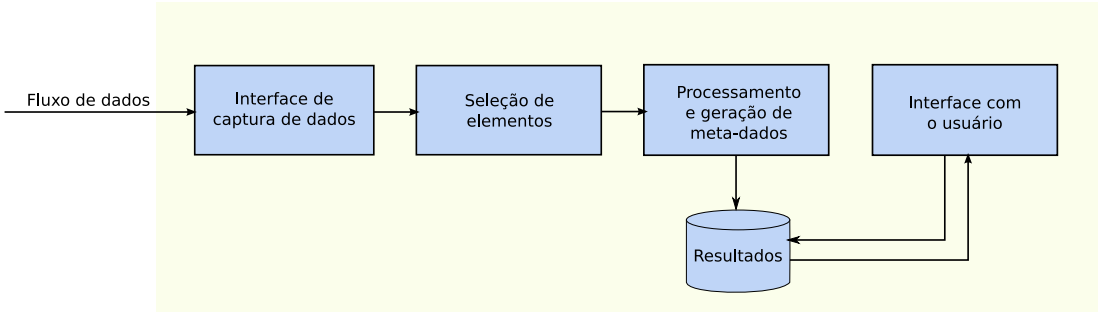

Figura 4.1: A arquitetura de um analisador de fluxo de dados proposta

Embora o tratamento dos elementos lidos dependa de características intrínsecas a eles, a Seção 2.2 mostrou que diversos algoritmos estatísticos podem ser aplicados para definir quais elementos procedentes do fluxo de dados serão processados. Da mesma forma, é possível definir um conjunto de regras genéricas que, aplicadas a elementos estruturados de um fluxo de dados, permitam a sua análise de maneira consistente e sistemática.

Este capítulo apresenta a contribuição principal desta dissertação, que é uma arquitetura para a análise de fluxo de dados estruturados. Sua composição é feita da seguinte maneira: primeiramente são apresentadas as justificativas que

levaram à proposta do mecanismo de análise; em seguida, são apresentadas as características que tornam viável a implementação deste mecanismo e, por fim, são descritas as vantagens de sua adoção frente a outros modelos de armazenamento e exposição de elementos estruturados amostrados.

4.1 Requisitos do Modelo de Análise de Fluxo de Dados Estruturados

A seguir são apresentados os requisitos desejáveis para a definição de um modelo que permita expor os vários atributos de um conjunto de dados estruturados. Parte destes requisitos foi determinada com base em experiências pessoais na análise de fluxos de dados em TV digital e de pacotes em redes de computadores. São eles:

1. Integração com aplicações já existentes; este requisito é importante pois existem à disposição inúmeras ferramentas genéricas (por exemplo, processadores de texto) e específicas (como aplicativos de análise matemática) já integradas à maioria dos sistemas operacionais que podem ser usados para a manipulação e análise de dados.

2. Uso de estruturas de dados legíveis e compreensíveis por usuários; o advento da internet e a evolução de linguagens de marcação incrementaram o debate acerca da importância da legibilidade e intuitividade das linguagens declarativas e procedurais. Considerando a possibilidade dos dados estruturados serem muito complexos, este requisito torna-se desejável.

3. Independência de ambientes computacionais; a cada dia aumenta a diversidade de microprocessadores, sistemas operacionais e linguagens de programação. Portanto, um sistema de análise abrangente deve ser capaz de operar numa diversidade grande de ambientes computacionais.

4. Facilidade de localização de elementos, atributos e metadados; considerando que as estruturas de dados são oferecidas na forma de fluxos e os dados estruturados podem ser complexos, a agilidade de localização de elementos, atributos e metadados torna-se desejável para uma análise rápida e eficiente.

5. Exposição das informações capturadas sem requerer aplicações especializadas; o objetivo deste requisito é evitar a necessidade de ferramentas sofisticadas requerendo esforços adicionais de desenvolvimento.

6. Possibilitar a navegação na estrutura hierárquica com diferentes níveis de detalhamento; como a expansão de todos os níveis hierárquicos dos elementos estruturados pode resultar em um grande volume de dados, torna-se desejável que cada nível possa ser expandido individualmente.

7. Permitir a representação de dependências entre elementos; em alguns casos os fluxos de transporte podem transportar elementos que contenham metadados de outros elementos. Nesse caso a representação de dependências entre elementos torna-se uma característica importante de ser analisada e exposta.

8. Elaboração de uma plataforma aberta a partir desses requisitos que possa ser usada efetivamente pela comunidade.

Vários desses requisitos podem ser atendidos por uma representação em arquivos textuais, tal como documentos XML. O grande volume de dados gerado por uma análise, entretanto, pode tornar difícil a leitura de um documento XML, considerando que cada elemento pode ser composto por subárvores de profundidade variada, que podem exceder os limites do campo visual. Essa limitação poderia ser contornada com o desenvolvimento de uma aplicação que permita o filtro de seu conteúdo e o colapso e a expansão de subárvores que não sejam de interesse do usuário. Essa solução, entretanto, impede a satisfação do requisito de número 5.

Tecnologias de armazenamento e consulta de informações baseadas em banco de dados permitem que o acesso a grandes volumes de elementos, subárvores e atributos específicos seja feita com grande eficiência e abstração. Para isso, o formato dos dados em disco é geralmente otimizado para que o banco de dados tenha um baixo tempo de resposta às solicitações feitas pelo usuário. Esse formato, entretanto, não é projetado para sua compreensão visual, o que requer o uso de aplicativos especializados para a extração dos dados armazenados. Essa característica impede que os requisitos de número 2 e 5 sejam satisfeitos.

Muito embora o desempenho no acesso a uma estrutura de diretórios dependa dos algoritmos e das estruturas de dados empregados no sistema de arquivos, a exposição de elementos estruturados hierarquizados por meio dessa abstração é aderente aos requisitos não atendidos pelas soluções anteriores, pois o suporte a operações sobre objetos em um sistema de arquivos está presente em qualquer aplicação e sistema operacional moderno. Além disso, ela permite, por meio do colapso e expansão de diretórios, que cada nível da hierarquia seja exposto ou

ocultado da interface pelo próprio usuário, sem a necessidade do desenvolvimento de aplicações especiais para esse fim. Por último, a representação de dependências entre elementos e a exposição de atributos e metadados pode ser feita com o uso de recursos padronizados, permitindo a portabilidade na exposição de elementos segundo essa abstração.

A ubiquidade de sistemas de arquivos no universo de plataformas computacionais e as possibilidades de representação de dados por meio de seus objetos e funcionalidades tradicionais justifica, assim, a escolha de um mecanismo para armazenamento e exposição de elementos estruturados hierarquizados baseado nesse conceito.

4.2 Definição do Modelo de Exposição de Elementos Estruturados em Sistemas de Arquivos

A estrutura de um elemento não vazio é constituída por um ou mais atributos e zero ou mais metadados. Cada atributo pode conter uma quantidade diversa de informação a respeito do elemento – inclusive subdivididas em um outro conjunto de atributos. Definimos que cada conjunto de atributos é mapeado para um diretório e que cada atributo que não seja subdividido em outros atributos é mapeado para um arquivo, cujo conteúdo é aquele contido na carga útil do atributo.

Metadados podem conter informações a respeito de um atributo específico ou de um conjunto deles. Definimos que cada metadado é representado por um par ordenado $(nome, valor)$, e que essa tupla é mapeada para um atributo do arquivo ou diretório que representa os dados.

Por último, definimos que a relação de dependência entre dois elementos α e β é representada por um par ordenado (γ, β), onde γ é um arquivo especial pertencente ao diretório α que *aponta* para o diretório ou arquivo β.

Essas regras são suficientes para representar hierarquias complexas, que podem ser acessadas com o uso de ferramentas tradicionais de ambientes computacionais, como as interfaces de abertura e leitura de arquivos e diretórios, especificadas e recomendadas pelos padrões internacionais de interoperabilidade entre sistemas operacionais POSIX.

As subseções a seguir especificam com detalhes como esses e outros conceitos

de sistemas de arquivos se relacionam com a análise de elementos estruturados.

4.2.1 Canais de Comunicação Unidirecionais Entre Processos

Além da funcionalidade básica de representação de elementos distintos capturados em um fluxo de dados, o acesso direto em tempo real ao conteúdo do fluxo beneficia diversas classes de problemas. A exemplo, os elementos analisados em um fluxo de vídeo H.264 podem conter informações quanto à configuração dos parâmetros da imagem, ao coeficiente de quantização, entre outros (ABNT, 2008d). Entretanto, pode ser interessante para o usuário reproduzir a mídia do fluxo de vídeo em paralelo ao processo de captura e processamento dos elementos específicos, sem que a integridade dos dados encaminhados ao componente de processamento seja comprometida.

Essa funcionalidade é suportada na arquitetura de análise proposta por um canal de comunicação unidirecional $\overrightarrow{ab}$ entre o processo produtor a e um único processo consumidor b, por onde cada elemento de entrada passa. A restrição a um único processo consumidor é feita para que o modelo de recepção de dados, no qual uma interface de leitura recebe apenas um fluxo de dados, seja representado corretamente. Do contrário, uma implementação pode ter seu desempenho prejudicado devido à realização de cópias redundantes dos elementos de entrada a cada um dos consumidores.

Um tipo de arquivo especial denominado FIFO (do inglês *First-In,First-Out*) é definido pelo padrão POSIX para prover serviços de comunicação entre processos por meio de arquivos. Um processo produtor abre o arquivo em modo de escrita e um outro processo o abre em modo de leitura. No entanto, diferentemente de uma operação sobre dados em disco, a área de armazenamento do sistema de arquivos não é utilizada; a troca de dados por meio de um FIFO ocorre com uma transferência direta de dados da origem para o destino.

As propriedades de FIFOs permitem, dessa forma, a preservação da semântica definida para um canal de comunicação $\overrightarrow{ab}$, justificando sua recomendação para a implementação do recurso de um canal de comunicação unidirecional entre processos em uma arquitetura de análise de elementos estruturados.

4.2.2 Mecanismos de Atributos

A informação contida em um elemento estruturado pode ser descrita com o uso de metadados – pares ordenados do tipo $(nome, valor)$. Esses pares, quando atribuídos a um elemento, permitem que algumas informações sejam obtidas sem a necessidade da interpretação completa dos seus dados. Por exemplo, um elemento que represente uma imagem digital pode conter os pares *(Resolução,1920x1080)*, *(Local,São Paulo)*, *(Evento,São Silvestre)*, facilitando a classificação dessa imagem pelo usuário.

A geração de metadados pode ser feita de duas maneiras: pelo fornecimento explícito do usuário ou pela própria arquitetura de análise de fluxo de dados, caso haja informação suficiente para sua geração automática. A arquitetura proposta neste trabalho define a seguinte política para a atribuição de metadados nos arquivos que representam os elementos e seus atributos:

1. Geração automática: é feita sempre que alguma informação relevante estiver disponível no fluxo de dados, ou que algum erro lógico tenha sido identificado no elemento lido. Tais informações são descritas por um nome prefixado por um radical determinado, reservado para metadados gerados automaticamente. Exemplo: `(‘‘system.error’’, ‘‘Sintaxe inválida’’)`.

2. Atribuição pelo usuário: é feita pelo usuário para o envio de comandos arbitrários aos diferentes componentes da arquitetura representados por arquivos FIFO. Tais comandos são prefixados por um radical determinado, diferente daqueles usados na geração automática de metadados. Exemplo: `(‘‘user.buffer_size’’, ‘‘4096’’)`.

Quatro operações sobre atributos são definidas na arquitetura: criação, modificação, leitura e exclusão. A política de modificação e exclusão segue os mesmos princípios adotados para a criação: metadados gerados automaticamente pelo sistema ou pelo usuário podem apenas ser modificados ou excluídos pelo mesmo.

Uma classe de funções estabelecidas pela norma POSIX 1003.1e (IEEE, 2001) permite a manipulação de metadados em arquivos. A interface de *atributos estendidos* definida no padrão foi inicialmente concebida para permitir a implementação de listas de controle de acesso a arquivos, porém ela é abrangente o suficiente para permitir o armazenamento de quaisquer pares de dados.

Uma discussão sobre o suporte a atributos estendidos em sistemas de arquivos tradicionais, que não foram projetados para suportá-los, pode ser encontrada no

Anexo A.

4.2.3 Ligações Simbólicas no Sistema de Arquivos

Com frequência, os elementos obtidos de um fluxo de transporte passam por algum processo de classificação, seja ele com a finalidade de agrupar elementos que tenham características similares ou de indicar aqueles que apresentem correlação com outros. Essa necessidade traz, como consequência, o armazenamento de tais elementos (arquivos e subdiretórios) dentro de um ou mais diretórios, o que pode implicar um aumento significativo na quantidade de espaço de armazenamento requerido.

Para evitar cópias redundantes de arquivos e subárvores, é definido o último componente do sistema de arquivos para análise de elementos estruturados, a *ligação simbólica*. Uma ligação simbólica é um arquivo especial α que mapeia um outro arquivo ou diretório β. Ao acessar um objeto $\alpha \mapsto \beta$, o sistema de arquivos deve recuperar a informação do alvo β e redirecionar a operação para ele. Além disso, operações especiais sobre o arquivo α devem permitir a identificação do seu alvo (sem que para isso seja necessário ler o conteúdo do arquivo alvo) e a criação de uma ligação simbólica para um alvo inexistente.

O padrão POSIX suporta esse tipo de declaração por meio de arquivos especiais validados em tempo de execução. O conteúdo do objeto alvo não é replicado no arquivo de ligação simbólica; este consiste apenas na informação do seu alvo. Com esse recurso, o modelo hierárquico deixa de oferecer uma única visão fixa e monolítica da estrutura em árvore, possibilitando o agrupamento de elementos e a descrição de dependência de dados com um impacto mínimo no espaço de armazenamento requerido [1].

4.3 Síntese

Este capítulo apresentou uma proposta de uma arquitetura para a análise de fluxo de dados estruturados com base em um sistema de arquivos. Essa proposta foi realizada a partir da definição de diversos requisitos desejáveis, os quais permitiram delinear o modelo de representação de dados e os recursos que esse modelo deveria apresentar.

A implementação de um analisador de fluxo de dados segundo a definição

[1] O espaço requerido é variável e depende da implementação de ligações simbólicas em cada sistema operacional

dessa arquitetura pode ser feita com o uso de ferramentas e interfaces padronizadas. Com isso, a exposição de elementos estruturados hierarquizados pode ser feita com grande portabilidade, visto que os recursos básicos exigidos estão presentes em qualquer plataforma que respeite padrões internacionais de interoperabilidade entre sistemas operacionais.

Essa mesma arquitetura oferece ainda um mecanismo flexível para a comunicação e o envio de comandos a componentes diversos que a compõem, de forma que suas funcionalidades básicas sejam estendidas, de forma a atender a outros requisitos desejáveis. Assim sendo, essa proposta supera os objetivos iniciais almejados, tornando-se genérica o suficiente para permitir o mapeamento de uma ampla classe de problemas.

O próximo capítulo apresenta como essa arquitetura pode ser aplicada para permitir a análise de componentes transmitidos em um fluxo de transporte de um sistema multimídia, mais especificamente do Sistema Brasileiro de TV Digital (SBTVD). A seguir, a validação da implementação de um sistema de arquivos para a análise do fluxo de transporte do SBTVD é apresentada, consolidando essa proposta.

5 Aplicação da Arquitetura Proposta no Sistema Brasileiro de TV Digital

Este capítulo apresenta como a arquitetura de análise de elementos estruturados proposta pode ser aplicada a um sistema multimídia, mais especificamente ao Sistema Brasileiro de TV Digital (SBTVD). A escolha desse tema ocorreu em função do envolvimento do autor deste trabalho com a recomendação, definição e implementação do sistema no país.

Desde as primeiras fases de implantação do SBTVD no Brasil, que ocorreu a partir de dezembro de 2005, a análise do fluxo de dados emitido pelos produtores de conteúdo se mostrou essencial para a validação dos elementos transmitidos e a correta implementação de terminais de acesso de TV digital. A rápida identificação da origem de condições anômalas na decodificação de elementos do fluxo de dados continua sendo determinante para que a transição do sistema de transmissão analógico para digital ocorra dentro dos prazos estabelecidos pelo Governo. Dessa forma, torna-se essencial a existência de mecanismos eficientes de análise que possam ser utilizados pelos diversos participantes desse fórum.

Este capítulo apresenta primeiramente a técnica empregada para a amostragem dos elementos que serão expostos ao usuário, seguida por uma proposta para a disposição desses elementos em uma hierarquia de diretórios segundo a arquitetura definida no capítulo anterior.

O capítulo é finalizado com as conclusões gerais da aplicação da arquitetura proposta no fluxo de transporte do SBTVD.

5.1 Um Sistema de Arquivos para Análise do Fluxo de Transporte

O Capítulo 3 mostrou que as tabelas transmitidas no fluxo de transporte seguem uma organização estruturada e hierarquizada, o que permite seu mapeamento direto a uma árvore composta por arquivos e diretórios em um sistema de arquivos. A transmissão cíclica de tabelas e o fluxo contínuo de elementos de áudio, vídeo e dados, no entanto, tende a exigir um espaço de armazenamento infinito para que toda a informação seja mantida na memória do receptor. Dessa forma, definimos algumas restrições quanto ao tipo de dado que ficará permanentemente acessível no sistema de arquivos:

1. O armazenamento de tabelas de informação de serviço é prioritário em relação a outros elementos do fluxo de transporte;
2. A estrutura de dados deve expor diferentes versões de uma tabela em subárvores distintas. Para otimizar o consumo de memória exigido para manter duas ou mais versões de uma mesma tabela, uma lógica similar à do algoritmo *Count-Min* (CORMODE; MUTHUKRISHNAN, 2004) é empregada: apenas os atributos que diferem daqueles já armazenados em versões anteriores daquela tabela são registrados;
3. Fluxos elementares de áudio e vídeo não são amostrados, porém o acesso ao seu conteúdo deve ser possível;
4. A estrutura de arquivos e diretórios transferidos por meio dos carrosséis de dados e objetos do DSM-CC pode demandar um espaço de armazenamento maior do que aquele encontrado na memória RAM. Para permitir que os pacotes de dados recebidos possam ser analisados, uma partição de dados não-volátil é usada para a sua gravação.

A tradução de elementos presentes no fluxo de transporte para objetos no sistema de arquivos é feita conforme um conjunto de regras:

1. Tabelas do MPEG-2 e do SBTVD são armazenadas na raiz do sistema de arquivos na forma de diretórios que recebem o nome da tabela em questão. Uma listagem do diretório raiz indica todas as tabelas reconhecidas e analisadas até então;

2. Um subdiretório dentro dos diretórios de cada tabela indica a versão recebida e abriga todos os atributos presentes naquela tabela. Descritores presentes em cada tabela são armazenados em subdiretórios específicos no mesmo nível hierárquico que abriga os seus atributos;

3. Atributos simples são armazenados na forma de arquivos. Atributos estruturados são armazenados em um diretório que recebe o nome do atributo e que contém dois ou mais arquivos ou subdiretórios.

4. Pacotes PES são armazenados na raiz do sistema de arquivos na forma de diretórios que recebem como nome o número PID do pacote em questão.

5. Objetos recebidos nos carrosséis de dados são armazenados em uma subárvore conforme a hierarquia especificada nas mensagens BIOP dos pacotes DSM-CC. Como eles são transmitidos em pacotes PES, a subárvore é criada dentro de um diretório criado conforme a regra anterior;

6. Fluxos elementares de áudio e vídeo, presentes em pacotes PES, são representados por meio de canais unidirecionais FIFO, que têm seu conteúdo produzido sob demanda, dependendo da existência de um processo consumidor.

A apresentação de metadados e os erros e inconformidades encontrados a partir da análise de cada tabela são dispostos por meio do mecanismo de atributos estendidos. O uso desse e outros objetos de sistema de arquivos é explicado a seguir.

5.1.1 O Uso de arquivos FIFO

Arquivos FIFO são usados para permitir o acesso direto aos fluxos elementares de áudio e vídeo. A arquitetura de análise pode prover acesso direto tanto aos pacotes PES, que contêm um cabeçalho com informações de tempo de decodificação e apresentação de cada quadro, quanto aos pacotes elementares (ES), que compõem a carga útil dos pacotes PES. O acesso a ambos é interessante do ponto de vista de análise, pois muitas plataformas de terminal de acesso apresentam recursos para a reprodução de fluxos ou arquivos monomídia.

A escolha de arquivos segundo a semântica de FIFOs permite o acesso direto aos fluxos elementares, sem a necessidade de que os mesmos sejam armazenados em disco ou mantidos em memória pelo sistema de arquivos. Ao mesmo tempo, um arquivo FIFO permite a captura de fluxos individuais de áudio e

vídeo para fornecê-los a aplicativos especializados na análise dos seus protocolos de codificação. Essa arquitetura também possibilita a integração de extensões ao sistema de arquivos: um analisador de H.264 poderia ser implementado como um processo consumidor da FIFO que representa um fluxo elementar de vídeo.

Outros elementos transferidos por meio de pacotes PES são os dados de legenda e *closed caption*. Eles podem ser igualmente acessados por arquivos FIFO ou processados separadamente por um aplicativo capaz de interpretar os protocolos utilizados para o transporte desse tipo de informação.

5.1.2 A Aplicação de Atributos Estendidos

Atributos estendidos são pares ordenados do tipo $(nome, valor)$ associados a um arquivo ou diretório. Seu uso na análise de elementos capturados do fluxo de transporte é feito de 3 maneiras diferentes. Na primeira, tuplas são geradas automaticamente pelo sistema de análise para indicar a ocorrência de erros ou dados informativos relevantes.

A nomenclatura "`system.error.<erro>`" é adotada em atributos indicativos de erro; um problema de CRC, por exemplo, é indicado por um atributo estendido de nome "`system.error.crc`", associado ao diretório principal que representa a tabela. O conteúdo presente no campo *valor* pode assumir uma frase indicativa como *Lido 0x2112, esperado 0x1221*.

Dados informativos são agregados ao arquivo ou diretório segundo o padrão "`system.info.<info>`". A indicação da taxa de transmissão de uma tabela segue essa sintaxe, em que o último campo assume o valor `bitrate`. Nota-se que a lista de nomes válidos deve ser de conhecimento do usuário para que ele possa automatizar a extração de informação a partir da leitura dos atributos de cada arquivo ou diretório.

Também com o intuito de facilitar a extração automática de informações de arquivos, o segundo uso dado a atributos estendidos consiste na indicação da formatação dos dados presentes em cada arquivo. O nome reservado "`system.format`" é definido para tal, podendo assumir valores como "`binary data`", "`number`", `string`", significando que um conteúdo binário, numérico ou textual é encontrado no arquivo. Outras construções são também admitidas, como "`string [number]`" e "`number [<new_line>number]`", indicando a presença de um texto legível seguido de um número e de um ou mais números separados por quebras de linha, respectivamente. Esses atributos são gerados automaticamente pelo

sistema de análise e não podem ser modificados pelo usuário.

O último uso definido para atributos estendidos na arquitetura de análise de fluxos de transporte do SBTVD é para especificar o tamanho dos *buffers* internos dos arquivos FIFO que dão acesso aos fluxos elementares de áudio e vídeo. A finalidade desse atributo é permitir a redução, em tempo de execução, do consumo de memória do computador analisador ou o aumento de quadros mantidos em memória para possibilitar a reprodução ininterrupta do seu conteúdo. O atributo reservado para essa finalidade, "`user.buffer_size`", é especificado pelo usuário.

5.1.3 A Organização de Versões e Dependências com Ligações Simbólicas

Ligações simbólicas são utilizadas na arquitetura de análise para indicar qual, entre uma ou mais versões de uma tabela já recebida, denota a última versão transmitida no fluxo de transporte. Uma construção similar é usada para indicar os fluxos elementares principais e secundários de áudio e vídeo em programas compostos por múltiplas línguas e câmeras. Essa indicação é especialmente útil para identificar o motivo pelo qual um programa é decodificado em espanhol quando o áudio em português é esperado, por exemplo. A ligação simbólica criada para a versão atual de uma tabela permite também o acesso a elementos específicos por meio de caminhos invariáveis; a última versão de um determinado arquivo pode ser sempre encontrada pela leitura de um caminho conhecido como `/PMT/Current/table_id`, que tem a ligação simbólica `Current` apontando para um subdiretório dentro de `/PMT`. A decisão de utilizar ligações simbólicas dessa maneira vem da proposta de Muhammad e Detsch (2002), que propõe uma estrutura de diretórios coerente para o armazenamento de programas instalados em sistemas operacionais Linux.

Dependências entre tabelas são também representadas por meio desses objetos no sistema de arquivos. A relação entre as tabelas PAT e PMT apresentada anteriormente na Figura 3.3 pode ser descrita com o uso dessa abstração; assumindo a existência das subárvores `/PMT/0x20/2/` e `/PMT/0x21/1/` que representam as tabelas PMT trazidas nos PIDs 0x20 e 0x21 e da estrutura de diretório `/PAT/1/Programs/`, a conexão entre as tabelas é indicada por duas ligações simbólicas `/PAT/1/Programs/1` $\mapsto$ `/PMT/0x20/2` e `/PAT/1/Programs/2` $\mapsto$ `/PMT/0x21/1`. Uma propriedade especial do uso de ligações simbólicas nesse caso é a simplicidade com que um fluxo de transporte fora de conformidade com o padrão é exposto para o usuário: caso uma das tabelas anunciada pela PAT

não tenha sido transmitida, por exemplo, a ligação simbólica apontará para um diretório inexistente, resultando em um elo quebrado – situação facilmente identificável por ferramentas padronizadas de listagem de diretórios como o comando `ls` especificado pelo POSIX 1003.2 [1].

Uma última aplicação dada às ligações simbólicas é o agrupamento de elementos que se encaixem em uma mesma categoria. Por exemplo, todos os pacotes que trazem fluxos elementares de áudio, vídeo ou legendas podem ser listados no diretório `/FluxosElementares`, que irá conter arquivos especiais apontando para o local de armazenamento de cada um deles, sem que o usuário precise navegar na hierarquia de diretórios para encontrá-los.

5.2 Síntese

Este capítulo apresentou como o mecanismo de análise de fluxo de dados proposto pode ser aplicado às tabelas encapsuladas no fluxo de transporte do Sistema Brasileiro de TV Digital, um superconjunto do fluxo de transporte do MPEG-2. Com a proposta, se mostrou ser possível representar os principais elementos de análise presentes em produtos comerciais e desejáveis na resolução de problemas de fluxos de transporte por meio de uma estrutura de diretórios conceitualmente simples.

Mesmo com um caráter de transmissão potencialmente infinito, o estabelecimento de algumas regras permite a análise das tabelas essenciais em um fluxo de transporte, ao mesmo tempo em que o acesso aos pacotes de áudio, vídeo e legendas pode ser feito para acompanhar cada programa recebido. Os mesmos objetos no sistema de arquivos usados para essa finalidade permitem a extração de fluxos independentes para sua análise posterior, recurso este executado com uma operação bastante tradicional de cópia de arquivos.

Diversas funcionalidades são fornecidas por meio de atributos estendidos, como a indicação de erros e dados informativos e a configuração do tamanho de memória reservada para os *buffers* de fluxos elementares. A flexibilidade fornecida por esse mecanismo permite ainda que outros metadados sejam agregados às tabelas, bastando para isso tornar pública a lista de nomes reservados usados pela implementação.

Além da finalidade de representar interdependências entre tabelas e de fornecer uma visão consistente do sistema de arquivos, ligações simbólicas possibili-

[1] Atualmente incorporado ao POSIX 1003.1

tam a quebra do paradigma tradicional de um modelo hierárquico que representa apenas uma visão fixa e monolítica da estrutura em árvore. Essa característica, somada à naturalidade com que a proposta se integra a outras estruturas de dados hierárquicas (como as transmitidas nos carrosséis de dados), valida a escolha de uma abstração de arquivos na definição de uma arquitetura para a análise do fluxo de transporte do SBTVD.

6 A Implementação de Referência DemuxFS

Este capítulo apresenta a características da implementação de referência de um sistema de arquivos para a análise do fluxo de dados do Sistema Brasileiro de TV Digital, denominada DemuxFS. Sua concepção é feita de acordo com as especificações consolidadas no capítulo anterior.

As seções a seguir são divididas da seguinte forma. A primeira apresenta as considerações quanto à escolha das arquiteturas de *hardware* e *software*. A segunda parte avalia os recursos computacionais exigidos por essa implementação, considerando as características de fluxos de dados mostradas no Capítulo 2 e dos elementos que compõem o fluxo de transporte do MPEG-2. Em seguida, a maneira segundo a qual os recursos definidos no Capítulo 4 são empregados no DemuxFS é demonstrada. Encerrando este capítulo, encontram-se as conclusões obtidas com a implementação e o uso do DemuxFS pelo grupo de pesquisa em TV digital do Laboratório de Sistemas Integráveis da USP.

6.1 Considerações de Implementação

As plataformas de *hardware* e *software* compatíveis com a implementação do sistema de arquivos foram determinadas com base em dois critérios: portabilidade, para permitir sua execução na maior quantidade de plataformas com o menor número de modificações no código possível, e integração com um terminal de acesso de TV digital de baixo custo, de forma a possibilitar a análise de fluxos de transporte a partir do próprio aparelho usado para assistir ao conteúdo de radiodifusão.

Um sistema de arquivos tradicional é intrinsicamente ligado às interfaces de programação (APIs) internas do sistema operacional (SO) no qual ele é desenvolvido, tais como funções de alocação de memória, proteção de acesso a regiões críticas, metodologias de declaração e registro de métodos, entre outros. Ele

também é diretamente dependente do formato de estruturas de dados do SO, como aquelas que representam instâncias de arquivos abertos e de *cache* de dados. Assim, mesmo que as interfaces de comunicação de aplicações do usuário com o sistema de arquivos sigam as especificações do padrão POSIX, sua implementação torna-se completamente dependente do sistema operacional no qual foi desenvolvido, demandando a reescrita de uma grande parte do código quando este é portado para outro sistema operacional.

A implementação de um sistema de arquivos portável requer, assim, que sua concepção seja feita em *espaço de usuário* [1], de onde o acesso aos recursos do sistema operacional pode ser feito por meio de interfaces padronizadas. Essa funcionalidade é encontrada na arquitetura de *software* proposta pelo FUSE (SZEREDI, 2009), do inglês *Filesystem in Userspace*, adotada na concepção do DemuxFS. O FUSE é destacado e apresentado com detalhes na subseção a seguir.

Como base para a implementação do DemuxFS, duas arquiteturas de *hardware* foram escolhidas. A primeira, devido a sua abrangência, é a plataforma PC. Nela, a entrada do fluxo de transporte é feita por meio de arquivos que contenham segmentos de programas de TV digital, capturados de transmissões reais ou gerados em laboratório. A segunda arquitetura consiste em uma plataforma ARM XScale para eletrônicos de consumo no segmento de TV digital. Essa arquitetura, projetada e desenvolvida pelo Laboratório de Sistemas Integráveis da USP, é apresentada na subseção seguinte à do FUSE.

Embora diversos trabalhos relacionados a TV digital tenham sido desenvolvidos em conjunto com outros pesquisadores do Laboratório de Sistemas Integráveis da USP, a implementação do DemuxFS foi feita separada e individualmente até o depósito desta dissertação.

6.1.1 Sistemas de Arquivos em Espaço de Usuário

O FUSE (*Filesystem in Userspace*) provê um mecanismo para a implementação de sistemas de arquivos em espaço de usuário em ambientes POSIX. Inicialmente concebido no Linux, ele encontra-se hoje integrado a sistemas operacionais como

[1] O termo *espaço de usuário* denota um modo de execução no qual processos têm acesso restrito a recursos computacionais, como instruções privilegiadas e leitura/ escrita a regiões de memória quaisquer; apenas o núcleo do sistema operacional, que executa em *espaço de sistema*, acessa-os livremente. É com base nessa separação que sistemas operacionais implementam mecanismos de proteção de acesso à memória, de segurança e de virtualização (SILBERSCHATZ; GALVIN; GAGNE, 2004).

o FreeBSD, NetBSD, Mac OS X, GNU/Hurd e Open Solaris [2].

A arquitetura do FUSE é constituída por dois módulos: uma biblioteca de funções, denominada *libfuse*, por meio da qual um sistema de arquivos implementa as semânticas das operações sobre seus objetos (arquivos e diretórios), e um módulo específico para o sistema operacional, implementado no núcleo executivo, que fica responsável pela comunicação entre o sistema de arquivos e as aplicações que requisitam dados armazenados nele. Em sua implementação no núcleo do Linux, o canal de comunicação entre a *libfuse* e o gerenciador do núcleo executivo é o dispositivo virtual `/dev/fuse`. Quando aberto, esse dispositivo retorna um descritor pelo qual o sistema de arquivos pode registrar seus serviços e enviar e receber comandos, segundo um protocolo específico. Uma vez registrado, os serviços exportados pelo sistema de arquivos tornam-se disponíveis para as aplicações por meio de um ponto de montagem definido pelo administrador na árvore de diretórios do computador onde o processo do sistema de arquivos executa.

O fluxo de uma operação no FUSE pode ser acompanhado na Figura 6.1. Nela, uma aplicação solicita a abertura de um arquivo `/mnt/arquivo.txt` por meio da função POSIX `open`. Essa função é fornecida pela biblioteca básica do sistema (Glibc), que abstrai a interface de abertura de arquivo do núcleo executivo utilizado, que nesse exemplo é a função `syscall` de número 5. A execução dessa função gera uma interrupção de *software*, levando à troca do contexto de execução do processo de modo usuário para modo sistema. Ao passar a executar em modo privilegiado, uma camada denominada VFS (*Virtual File System*) determina qual sistema de arquivos responde por requisições feitas ao diretório `/mnt`, e então redireciona a solicitação ao sistema de arquivos correspondente, que trata o comando e retorna o resultado para a aplicação solicitante (WEINBERGER, 1984).

Ao oposto de sistemas de arquivos implementados no núcleo executivo, como o 9P e o Procfs mostrados na Figura 6.1, os serviços de um sistema de arquivos FUSE não se encontram efetivamente no espaço de sistema. Cabe ao módulo FUSE, então, redirecionar a solicitação, mais uma vez, ao processo em espaço de usuário que implementa o serviço (identificado na figura pelo processo P2), que irá finalmente tratá-la e retornar o resultado para a aplicação (identificado pelo processo P1).

O impacto associado ao uso do FUSE em relação a uma implementação nativa

[2] As dificuldades técnicas e desafios associados à sua implementação em plataformas Microsoft Windows são abordadas em (DRISCOLL; BEAVERS; TOKUDA, 2008).

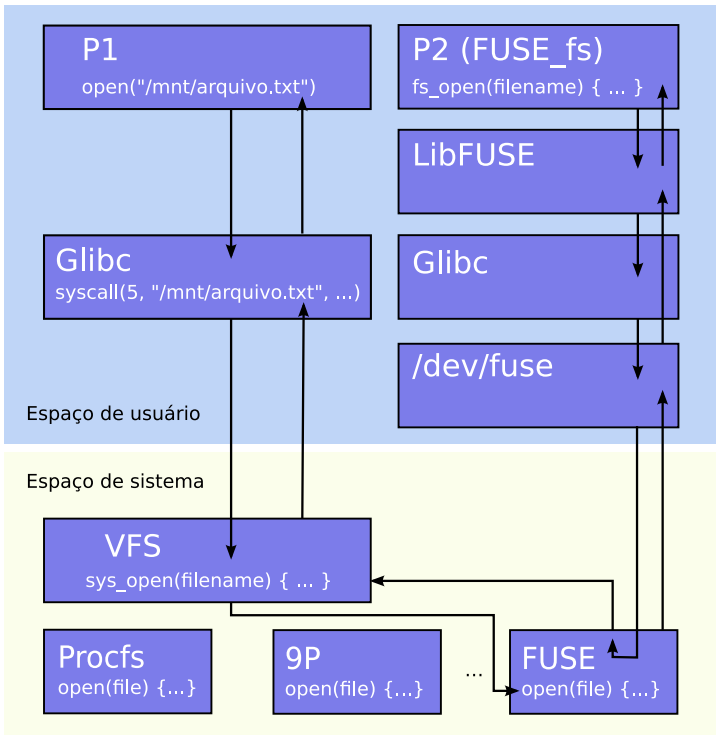

Figura 6.1: Fluxo de uma chamada de sistema a um sistema de arquivos FUSE

no núcleo executivo está nas trocas de contexto e cópia de dados extras que ocorrem para cada chamada de sistema durante a comunicação das duas entidades pelo canal `/dev/fuse`. A latência da chamada de sistema pode ser dramaticamente reduzida, em algumas implementações do FUSE, com o uso de segmentos de memória compartilhados, que evitam a cópia redundante de dados. Assim, nessas plataformas, o impacto no uso de sistemas de arquivos implementados em espaço de usuário se reduz às trocas de contexto, de menor relevância para projetos que não requerem tempos de resposta previamente determinados, como é o caso do DemuxFS.

6.1.2 A Plataforma de TV Digital Tall

A Tall é uma plataforma para terminais de acesso de TV Digital. Desenvolvida no Laboratório de Sistemas Integráveis (LSI) da USP e baseada na arquitetura do processador de mídias Intel CE2110, a Tall é composta por um processador ARM XScale de 1 GHz e dois módulos de memória RAM de 128 MiB cada. A periferia de componentes inclui decodificadores de vídeo H.264 e MPEG-2, um acelerador gráfico para operações 2D e 3D e controladores SATA, USB 1.1, USB 2.0 e Smart Card, entre outros. Um sintonizador/demodulador digital capaz de interpretar o sinal do SBTVD é também integrado à plataforma, que pode ser vista na foto da Figura 6.2.

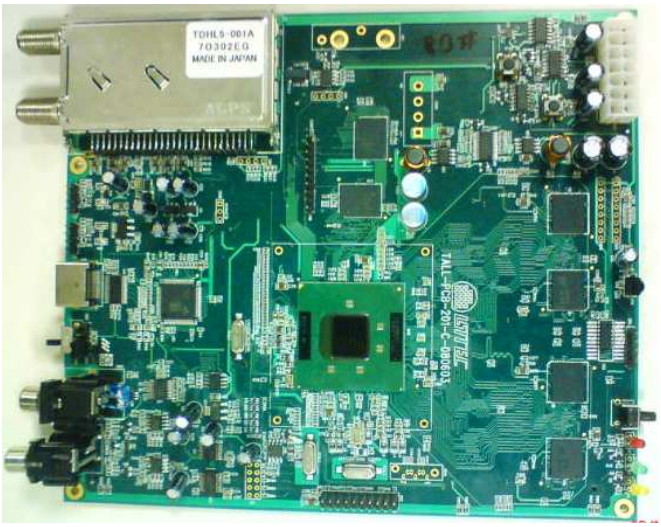

Figura 6.2: A plataforma de TV digital Tall

Tanto o projeto de *hardware* quanto o desenvolvimento do *software* básico da plataforma Tall foram feitos pelo grupo de pesquisadores do LSI. Sua pilha de *software* é composta pelo núcleo do Linux e pelas bibliotecas Glibc e GStreamer, que fornecem funcionalidades de programação na linguagem C e de aplicações multimídia, respectivamente. A interface de leitura do fluxo de transporte é feita a partir da saída do sinal demodulado do *tuner*, que é enviado para um dispositivo demultiplexador programável; é por meio dele que uma aplicação residente solicita a leitura de tabelas de informação de serviço e encaminha os pacotes de fluxos elementares de áudio e vídeo aos decodificadores dedicados.

Uma aplicação completa de TV digital foi desenvolvida para a plataforma Tall pelo LSI. Entre os recursos suportados estão a troca básica de canais, a visualização de legendas e do guia de programação enviado pelas emissoras, a atualização do *software* residente por carrosséis de dados do DSM-CC e a sua integração com o *middleware*. Essa aplicação integra-se à plataforma Tall de modo a oferecer uma solução completa de TV digital no Brasil.

A participação pessoal na produção dessa plataforma ocorreu em diversas etapas do projeto, que incluem a definição e a implementação dos protocolos de comunicação entre o *middleware* e a aplicação de reprodução de mídias, o desenvolvimento de *drivers* de dispositivos e a infraestrutura geral dos programas residentes, responsáveis pela decodificação e exibição de mídias. Essa afinidade com o projeto tornou a escolha da Tall bastante natural para a validação da proposta do tema desta dissertação.

6.2 Requisitos Computacionais

O DemuxFS é um sistema de arquivos implementado em espaço de usuário, cujos arquivos e diretórios são mantidos em uma área de armazenamento volátil. Como consequência, a utilização da memória RAM deve ser otimizada de forma a maximizar a quantidade de elementos que possam ser amostrados do fluxo de transporte.

Os elementos armazenados em memória não-volátil pelo DemuxFS são as tabelas de informação de serviço (SI) descritas no fluxo de transporte do MPEG-2. O Capítulo 5 mostrou que a transmissão das tabelas SI é feita de forma cíclica, de modo que um terminal de acesso de TV digital possa identificar o conteúdo transmitido no fluxo de dados e iniciar a decodificação do programa sem que o usuário precise aguardar por um período muito longo. Com exceção da tabela de descrição de data e hora, o formato do cabeçalho presente nas tabelas SI é comum a todas. Nesse cabeçalho, um campo em especial determina se o elemento deve ou não ser amostrado: a sua *versão*.

Caso uma tabela de versão V já tenha sido amostrada em um instante de tempo anterior ao atual, não se faz necessário que ela seja processada e agregada ao sistema de arquivos mais uma vez. Dessa forma, caso o fluxo de transporte não traga alterações nas versões das tabelas já amostradas, não haverá mais demanda para a alocação de memória por parte do DemuxFS. A tabela de descrição de data e hora, que não apresenta um campo de versão, é processada sempre que recebida e a memória usada pela sua amostra anterior é reaproveitada e sobrescrita.

Para não comprometer a disponibilidade de memória RAM, 3 diferentes políticas podem ser adotadas na chegada de novas versões de tabelas já amostradas, escolhidas pelo usuário durante a carga do DemuxFS. São elas:

- Ávida: armazena todas as versões recebidas, cada qual em um subdiretório exclusivo que leva o nome da versão presente no cabeçalho do pacote. Caso o emissor gere novas versões em uma taxa excessivamente alta a ponto de repetir versões previamente amostradas, a amostra coletada anteriormente é removida e substituída pela nova. Um indicador no sistema de arquivos denominado `Current`, implementado sob a forma de um *link simbólico*, aponta para a nova versão de modo que o usuário que analisa o conteúdo saiba qual é a versão da tabela em vigor. Essa política é conceitualmente similar à definição de um *buffer* circular (CORMEN et al., 2001);
- Moderada: idem à anterior, mas até N amostras diferentes de cada tabela

são mantidas em memória;

- Reservada: mantém apenas a última versão V recebida da tabela em memória. A versão $V - 1$ é retirada da árvore do sistema de arquivos e a memória ocupada por ela é liberada completamente assim que todas as referências feitas a seus arquivos e subdiretórios forem fechadas [3]. Para garantir uma visão consistente do sistema de arquivos, mesmo sob diferentes políticas de gerenciamento de memória, o indicador `Current` é atualizado para apontar para a nova e única versão amostrada dessa tabela.

As restrições de uso de memória são também aplicadas aos arquivos FIFO que representam os fluxos elementares de áudio, vídeo e outros dados transmitidos em pacotes PES. Por meio das chamadas de sistema de configuração de atributos estendidos, o usuário pode minimizar o uso de memória RAM ou maximizar o desempenho de um aplicativo de reprodução de mídias, de acordo com a sua necessidade.

6.3 A Arquitetura de Análise de Fluxo de Dados do DemuxFS

O DemuxFS é um sistema de arquivos real implementado sobre a infraestrutura do FUSE. Todos os objetos criados e exportados pelo DemuxFS tornam-se visíveis na estrutura de diretórios e podem ser acessados por qualquer programa e usuário.

A arquitetura básica do DemuxFS é apresentada na Figura 6.3, dividida em 5 módulos diferentes: entrada do fluxo de transporte, extração, validação e processamento de pacotes e representação na árvore de diretórios. A arquitetura admite a entrada do fluxo de transporte por duas possíveis origens: arquivos e por um sintonizador (*tuner*), este último disponível apenas na plataforma Tall. Os elementos de entrada são separados em pacotes de 188 bytes e encaminhados ao módulo de validação, que determina se o pacote é passível de análise ou não: pacotes que não apresentam carga útil ou que não contemplam o byte de sincronismo 0x47 são descartados. Enquanto o primeiro caso é admitido pela especificação do fluxo de transporte do MPEG-2, o último consiste em um erro, que é reportado pelo DemuxFS em um arquivo de registro de eventos. Após sua validação, o pacote é encaminhado para o módulo de processamento, de onde seu conteúdo será extraído e exposto por meio do sistema de arquivos.

[3] Esse comportamento é especificado na norma POSIX

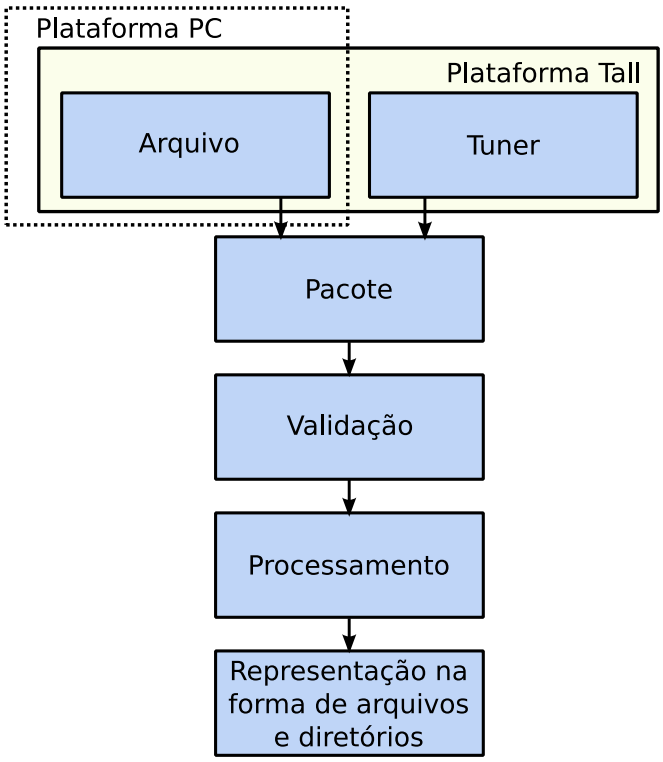

Figura 6.3: Diagrama de blocos do DemuxFS

O tratamento dado aos pacotes de informação de serviço (SI) e de fluxos de áudio, vídeo e dados é mostrado na Figura 6.4. Cada pacote recebido é armazenado em uma lista de *buffers* indexada pelo seu número identificador (PID). Apenas quando todos os pacotes de uma tabela SI estiverem no *buffer* seu conteúdo é extraído e exposto no sistema de arquivos. O mesmo é válido para hierarquias de objetos transferidas por meio de mensagens BIOP do DSM-CC: a cada bloco inserido na lista daquele PID a tabela DII é verificada para identificar se todo o conteúdo já foi recebido. Quando todos os dados já tiverem sido armazenados no *buffer* os arquivos e diretórios transferidos pelo carrossel de objetos são armazenados no disco rígido e ligações simbólicas são criadas de dentro da árvore do DemuxFS para o ponto onde eles foram gravados.

O fluxo de pacotes PES de vídeo e áudio e dados (com exceção dos carrosséis de dados e objetos) é mais simples; o destino desses pacotes é invariavelmente um arquivo FIFO. Uma única exceção ocorre no tratamento de pacotes PES de vídeo, que podem conter uma identificação de tamanho zero. Esse caso é especificado na norma do SBTVD (ABNT, 2008b), significando que o pacote não tem um tamanho definido. Logo, pacotes de vídeo recebidos dessa maneira não necessitam ser acumulados em um *buffer* antes de serem encaminhados ao arquivo FIFO correspondente.

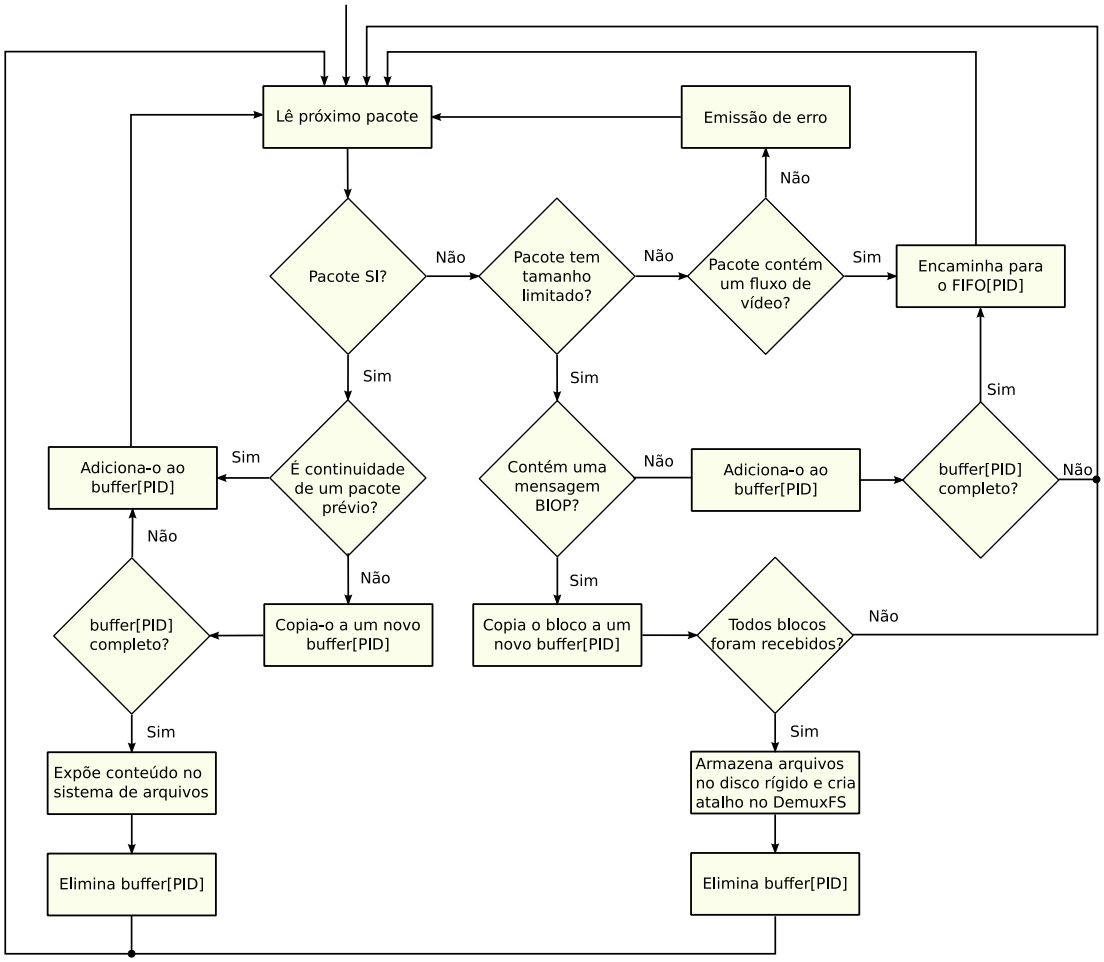

Figura 6.4: O fluxo de um pacote no DemuxFS

O modo de operação dos arquivos FIFO é descrito com mais detalhes na subseção a seguir.

6.3.1 A semântica e o uso dos arquivos FIFO

A semântica das operações nos arquivos FIFO é definida pela implementação do DemuxFS e reflete no desempenho do sistema de arquivos. Primeiramente, apenas um processo consumidor é admitido. A tentativa de abertura de um FIFO que já está em uso por outro processo é negada, e um erro indicativo de recurso ocupado é retornado ao usuário. Essa restrição garante que quando um descritor de arquivo válido for retornado em uma operação de abertura de um FIFO o processo terá acesso a todos os pacotes transferidos por esse canal. Como medida de otimização, caso o FIFO associado a um pacote PES não tenha sido aberto por um processo, o pacote é descartado imediatamente; nenhum acúmulo é feito no *buffer* daquele PID e nenhum dado é repassado para o FIFO.

O atributo estendido `user.buffer_size` proposto no capítulo anterior é aplicado aos arquivos desse tipo. A configuração desse parâmetro implica a modificação do tamanho da memória de armazenamento interna de um arquivo FIFO específico, podendo refletir na percepção do usuário que reproduz um fluxo elementar de áudio ou vídeo a partir desse arquivo.

Os arquivos FIFO representam dois tipos de elemento no DemuxFS: pacotes PES, presentes no fluxo de transporte, e pacotes ES, que consistem na carga útil dos pacotes PES. O desencapsulamento dos pacotes PES é feito internamente pelo DemuxFS para permitir a reprodução de fluxos monomídia (que não são sincronizados com uma segunda mídia) e para satisfazer decodificadores que não interpretem o cabeçalho de pacotes PES, como ocorre no *hardware* especializado da plataforma Tall. A manutenção interna dos arquivos FIFO de pacotes PES e ES é feita separadamente para cada um deles. Assim, é possível que um processo consuma os pacotes ES, enquanto outro processo consuma os dados trafegados no FIFO do PES, embora a restrição de uma única abertura por FIFO ainda se aplique.

6.3.2 Prévias do Fluxo de Vídeo

Em diversas situações de análise de fluxo de transporte, a visualização de uma única imagem do vídeo é suficiente para que o usuário identifique o programa transmitido ou verifique alguma condição de anomalia. Imagens estáticas capturadas do vídeo são também apreciadas na geração de relatórios de problemas encontrados em uma determinada emissora. A captura de uma única imagem, no entanto, demanda os mesmos recursos de uma decodificação de vídeo, como a reserva exclusiva de um decodificador em *hardware*.

Esse recurso é incorporado no DemuxFS com o auxílio de um decodificador de vídeo H.264 em *software* provido pelo projeto FFmpeg (BELLARD, 2000). Esse decodificador integra-se ao sistema de arquivos por meio dos arquivos FIFO de pacotes ES e acrescenta mais um objeto à subárvore do diretório, denominado `snapshot.ppm` [4].

A abertura do arquivo de prévia de vídeo requer a leitura do conteúdo transmitido no arquivo FIFO dos pacotes ES. Assim, quando o sistema de arquivos recebe uma solicitação de abertura do arquivo `snapshot.ppm`, o contador de re-

[4] O PPM é um formato de representação de *pixels* de codificação extremamente simples, no qual cada ponto da imagem é representado por um valor numérico em texto (ASCII). Mais informações sobre esse formato podem ser encontradas em (KAY; LEVINE, 1995)

ferência do arquivo FIFO associado é incrementado, impedindo sua abertura por um processo externo até que todos os quadros necessários para a decodificação de uma imagem completa tenham sido lidos. Caso o arquivo FIFO já esteja em uso por um processo no momento da abertura do arquivo `snapshot.ppm`, um erro informando que o recurso está ocupado é retornado para a aplicação solicitante.

6.3.3 Organização da Estrutura de Diretórios

A análise do fluxo de transporte do SBTVD é feita no DemuxFS por meio da listagem e leitura de diretórios e arquivos e de seus atributos estendidos, dispostos hierarquicamente conforme a estrutura lógica de cada tabela. O primeiro nível ($n = 0$) é composto unicamente por diretórios, cada qual representando uma tabela de informação de serviço, um fluxo elementar ou uma subárvore transmitida pelos carrosséis de dados e objetos. Duas exceções são feitas para os diretórios `/Report` e `/Streams`, que não abrigam diretamente o conteúdo de uma tabela específica. No primeiro se encontram relatórios de erros e avisos emitidos pelo DemuxFS, que possibilitam a visualização dos eventos conforme um ordenamento temporal, enquanto no segundo são encontradas ligações simbólicas para os diretórios do DemuxFS que abrigam fluxos elementares de áudio, vídeo e dados. Esses diretórios são expandidos na Figura 6.5 [5].

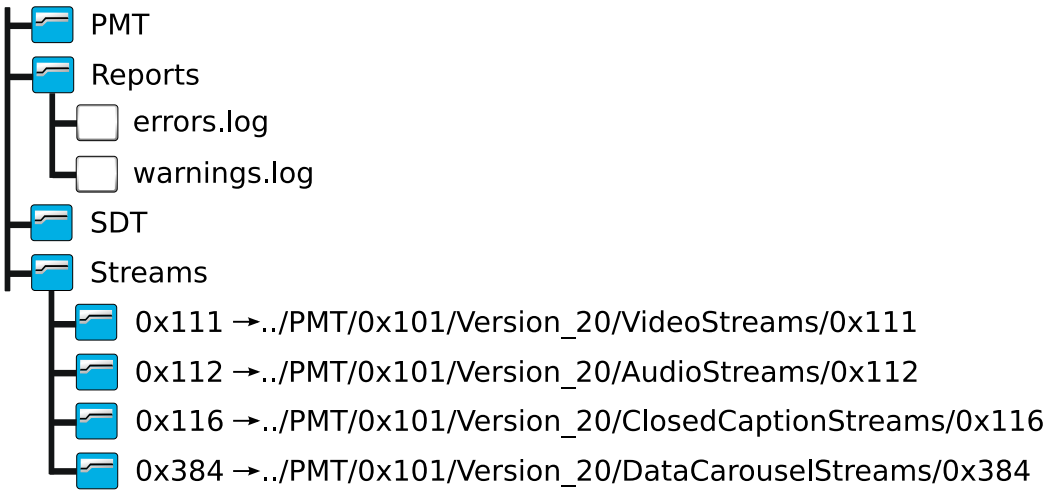

Figura 6.5: Os diretórios `/Reports` e `/Streams` expandidos

Salvo os diretórios supramencionados, os elementos expostos no segundo nível são definidos segundo três regras, aplicadas de acordo com as características da tabela representada e com o conteúdo do diretório. Essas características, listadas a seguir, podem ser acompanhadas na Figura 6.6:

[5] As ligações simbólicas estão listando origem e destino separados por uma seta (`->`)

- Tabelas que não incluem um campo de informação de versão: contém apenas um diretório denominado `Current`, dentro do qual os arquivos e diretórios associados à tabela se encontram. Esse é o caso da tabela de data e horário (TOT);

- Tabelas que incluem um campo de informação de versão: cada versão da tabela recebida é armazenada em um diretório separado. Uma ligação simbólica denominada `Current` é também criada, apontando para o diretório da última versão recebida. Esse é o caso de tabelas como PAT, NIT e SDT;

- Diretórios de agrupamento: esse tipo de diretório tem como finalidade agrupar todas as tabelas de um mesmo tipo recebidas no fluxo de transporte, como a PMT. O diretório principal recebe o nome da tabela em questão e abriga subdiretórios que levam como nome o PID de cada uma das tabelas que o compõem. Dentro dos diretórios de PID a organização baseada em versões e em uma ligação simbólica para a última versão recebida se aplica.

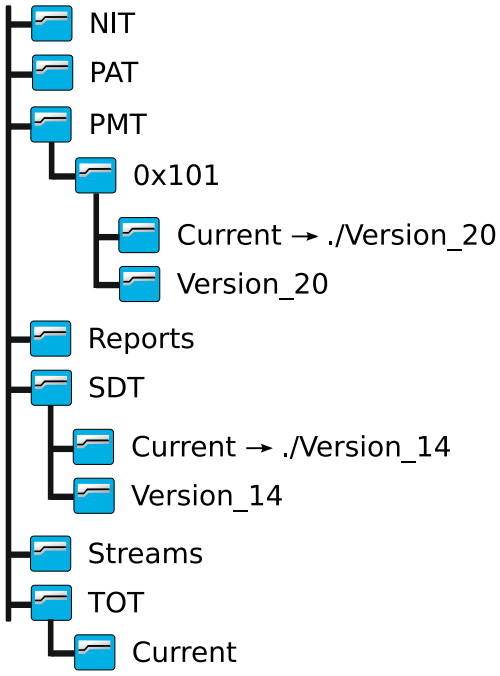

Figura 6.6: O segundo nível hierárquico do DemuxFS

A estruturação da árvore de diretórios dessa maneira traz uma coerência estrutural de grande valia para a busca de elementos específicos do fluxo de transporte, visto que seu estado atual pode ser sempre encontrado dentro dos objetos `Current`. Assim cada diretório contém os dados que dizem respeito àquela tabela, como seus atributos e descritores. A seguir, algumas das tabelas que apresentam

características peculiares quanto ao uso de ligações simbólicas, atributos estendidos e formatação de seus dados são apresentadas.

6.3.3.1 Organização do Diretório da Tabela PAT

Todas as tabelas de informação de serviço são precedidas de um cabeçalho em comum que apresenta informações quanto à sua identificação (expostos no DemuxFS dentro do diretório `tableHeader` pelos arquivos `table_id` e `identifier`), seu tamanho (`section_length`), versão (`version_number`) e número de seção (`section_number` e `last_section_number`). Encontram-se também nesse diretório os atributos `current_next_indicator`, que indica se a informação trazida na tabela deve ser imediatamente aplicada ou não, e o `section_syntax_indicator`, que deve ser um valor constante fixado em 1. Alguns campos de uso reservado são também encontrados nesse cabeçalho, porém seus valores não devem afetar a decodificação ou a correta interpretação de uma tabela de serviço. Por essa razão, eles não são expostos pelo sistema de arquivos.

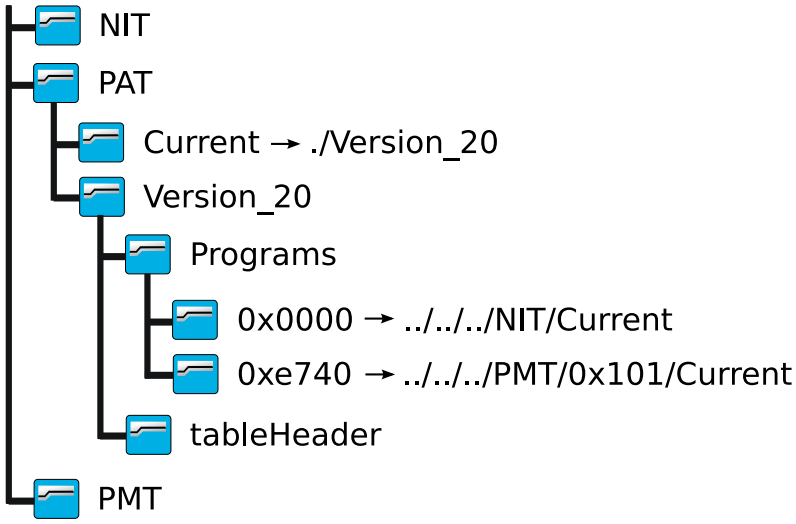

Figura 6.7: O subdiretório `Programs` da tabela PAT

A finalidade da tabela PAT é indicar o pacote que contém a definição atual da tabela de descrição de rede (NIT) e o mapeamento dos programas transmitidos pela emissora (PMT). Assim, além dos arquivos comuns entre as tabelas de informação de serviço, o diretório da tabela PAT contém um subdiretório exclusivo denominado `Programs`, dentro do qual encontram-se uma ligação simbólica para o diretório onde a tabela NIT atual está armazenada e uma ou mais ligações para cada um dos programas esperados nas tabelas PMT. A Figura 6.7 mostra como é feita a composição dos alvos desses objetos.

6.3.3.2 Organização do Diretório da Tabela PMT

A tabela PMT descreve os fluxos elementares que devem ser combinados para formar um programa. São também transferidos na PMT pacotes privados, que dizem respeito apenas a uma emissora em especial, e pacotes de carrosséis de dados e de objetos. A variedade de dados que podem ser transmitidos na PMT leva a criação de subdiretórios específicos, como mostra a Figura 6.8. Cada fluxo elementar é listado em um diretório dedicado que leva o nome do PID que o transporta.

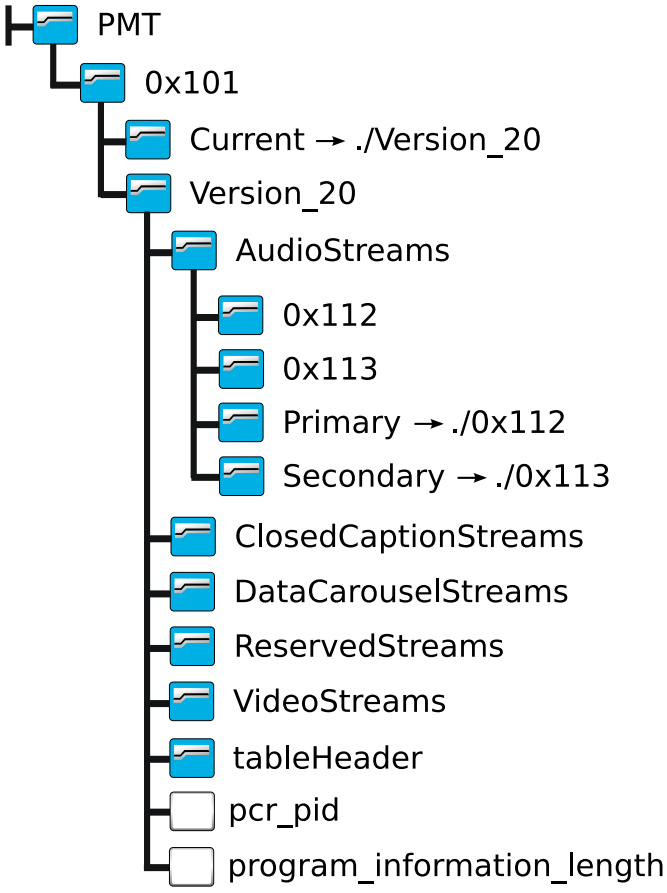

Figura 6.8: A composição da tabela PMT no DemuxFS

Para facilitar a depuração de fluxos de transporte, uma ligação simbólica denominada `Primary` indica os fluxos elementares de áudio, vídeo e legenda de prioridade máxima, que devem ser reproduzidos por padrão caso não haja intervenção do usuário na configuração do terminal de acesso. Caso haja mais de uma opção, uma ligação denominada `Secondary` é criada pelo DemuxFS, apontando para o fluxo de segunda prioridade, conforme as regras estabelecidas pelo SBTVD para sua determinação. Essas ligações simbólicas não são geradas para os fluxos de carrossel e de uso reservado da emissora pois seu tratamento não requer esse tipo de classificação.

Os fluxos de áudio e vídeo destinados à reprodução em dispositivos portáteis são diferenciados dos outros pela adição do sufixo `-OneSeg` [6] ao nome do diretório, conforme indica a Figura 6.9. Essa figura também introduz alguns importantes elementos do DemuxFS: a representação de descritores, os arquivos FIFO e os arquivos de prévia de vídeo (*snapshots*), explicados a seguir.

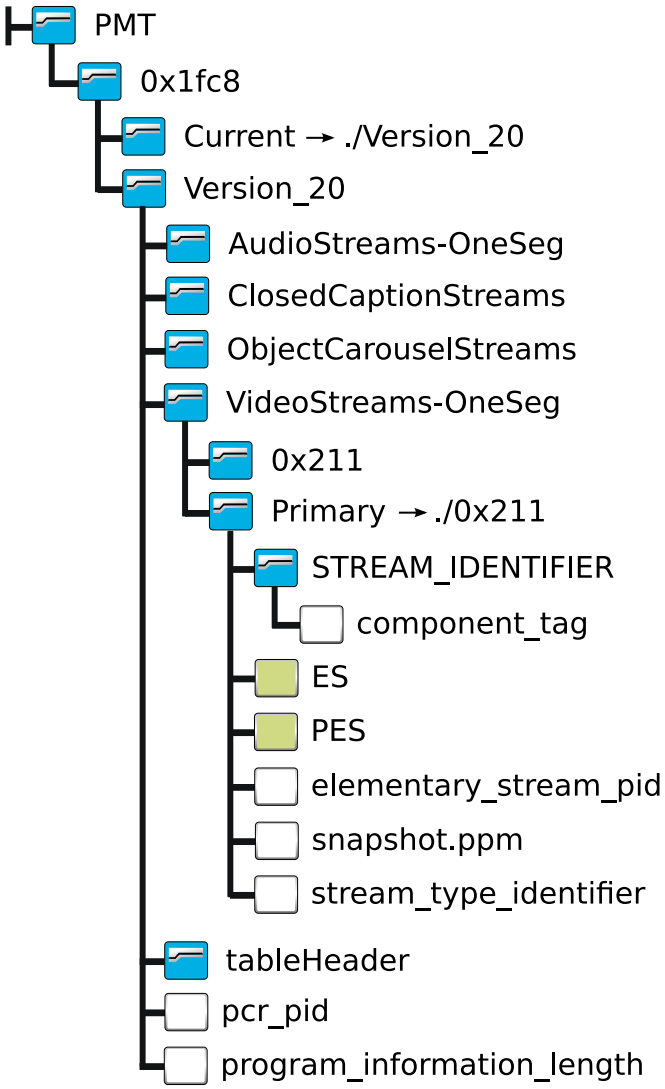

Figura 6.9: A descrição de fluxos elementares de áudio e vídeo na PMT

Cada descritor presente em uma tabela do fluxo de transporte é armazenado em um diretório especial, mesmo que ele consista em apenas um atributo. O nome do diretório é definido conforme o campo `descriptor_tag` da tabela e é sempre representado em caracteres maíusculos. Como regra geral, sempre que houver como interpretar os dados de um descritor, tanto a representação interpretada quanto a numérica constará nos dados retornados na operação de leitura do ar-

[6] Esse tipo de receptor é somente capaz de receber e decodificar sinais de perfil básico de TV digital, de um segmento. Dispositivos não-portáteis possibilitam a decodificação de mídias em 13 segmentos (ABNT, 2008c).

quivo. O exemplo da Figura 6.10 mostra o conteúdo do arquivo `component_tag` do fluxo elementar de áudio do PID 0x212 e o formato indicado pelo atributo estendido `system.format`. Enfatiza-se a estrutura do arquivo, formado pelo valor interpretado e seguido pelo valor numérico encontrado na tabela entre colchetes.

```
$ getfattr STREAM_IDENTIFIER/component_tag
system.format=‘‘string [number]’’

$ cat STREAM_IDENTIFIER/component_tag
Primary one-seg audio stream [0x83]
```

Figura 6.10: Formatação de um arquivo conforme indicado em seu atributo estendido

Os arquivos denominados `ES` e `PES` encontrados nos diretórios de áudio e vídeo representam os FIFOs do fluxo elementar, por meio do qual uma aplicação pode ter acesso para sua decodificação, inspeção ou cópia. Como o tamanho desses arquivos é indefinido, visto que a geração do conteúdo destes é potencialmente infinita, cabe à aplicação que os lê determinar quando a leitura deve terminar. A leitura dos arquivos `ES` e de prévia de vídeo (`snapshot.ppm`) está condicionada à disponibilidade do arquivo `PES`, por meio do qual o seu conteúdo é gerado. Caso o arquivo `PES` já esteja em uso por algum outro processo a abertura do arquivo `snapshot.ppm` é negada pelo DemuxFS.

Após a abertura dos arquivos PES e ES, a primeira operação de leitura garante que os primeiros valores lidos estejam sempre alinhados com o início de um pacote – como no exemplo da Figura 6.11, em que os primeiros 4 bytes lidos contêm o cabeçalho de um pacote PES de vídeo, iniciado por 00 00 01 0e. Isso facilita o reconhecimento e o tratamento do conteúdo extraído dos arquivos FIFO por outras aplicações.

6.3.3.3 Organização do Diretório da Tabela TOT

A tabela TOT é usada pelas emissoras para enviar a data e hora atual. É a partir desse valor de referência que outras tabelas de descrição de eventos são transmitidas, informando o horário de início e término de cada programa na grade de programação do canal. Além disso, o valor da tabela TOT enviado por diferentes emissoras pode diferir por variadas razões, o que torna a interpretação dessa tabela importante. Alguns dos problemas que podem ser resolvidos a partir da compreensão do valor da TOT incluem a exibição da data e hora nas tarjas de navegação da TV, o agendamento da gravação de um programa e o cálculo do

```
$ hexdump PMT/0x20/Current/VideoStreams/0x111/PES
00000000  00 00 01 e0 00 00 84 80  05 21 1a 25 5d 6d 00 00
00000010  00 01 09 50 00 00 00 01  28 fa 43 cb 00 00 01 06
00000020  01 07 00 00 0a 00 00 03  00 14 02 02 be a0 80 00
00000030  00 00 01 01 9e 14 29 1f  fc d6 ff 63 e7 3e 55 03
00000040  32 33 72 6d b3 96 a8 72  da 40 73 2b f3 90 0f 30
00000050  38 4d 3b e9 47 f2 dd 26  cd 00 bd 5c 3e a6 dd b5
00000060  9f 94 0d de 3e 72 d1 aa  85 8d 11 48 92 59 6c ba
00000070  d9 aa 85 41 e9 39 54 9b  60 8f c0 1f 77 0d 85 66
...

$ file PMT/0x20/Current/VideoStreams/0x111/PES
PES: MPEG sequence
```

Figura 6.11: Alinhamento dos arquivos FIFO com o cabeçalho de pacotes PES

tempo de duração de um evento ou programa específico.

```
$ getfattr TOT/Current/ut3c_time
system.format=‘‘string [number]’’

$ cat TOT/Current/utc3_time
Fri Dec 19 06:45:44 2008 [0xd623064544]
```

Figura 6.12: Representação de data e hora no DemuxFS

Para facilitar a análise da tabela TOT, o valor numérico de 40 bits `utc3_time` é convertido para um formato em texto compreensível. Esse formato, juntamente com o valor original, é obtido na operação de leitura do arquivo, como mostra a Figura 6.12.

6.3.3.4 Organização do Diretório das Tabelas DII e DDB

A integração do DemuxFS com a árvore de diretórios transmitida nos carrosséis de dados e objetos é feita a partir da análise da pilha de protocolos do DSM-CC. Duas tabelas são processadas para que os elementos do carrossel transmitidos no fluxo de transporte possam ser expostos conforme a hierarquia estipulada no cabeçalho das mensagens BIOP: a tabela DII e a tabela DDB.

O recebimento da tabela DII, que indica os blocos que compõem um determinado objeto, resulta na criação de um diretório `DII` na raiz do sistema de arquivos e de n subdiretórios denominados `module_`i para cada módulo $i, 0 \leq i<n$, recebido no carrossel. Cada subdiretório contém arquivos que identificam e descrevem o tamanho e a versão do módulo em questão. A recepção e a união dos pacotes

DDB transmitidos conforme a descrição desses módulos permite a reconstrução da hierarquia original, que fica exposta dentro de um diretório denominado DSM-CC, em um subdiretório que leva o nome do programa interativo, obtido a partir das informações de uma outra tabela denominada AIT (do inglês *Application Information Table*, ou Tabela de Informação da Aplicação).

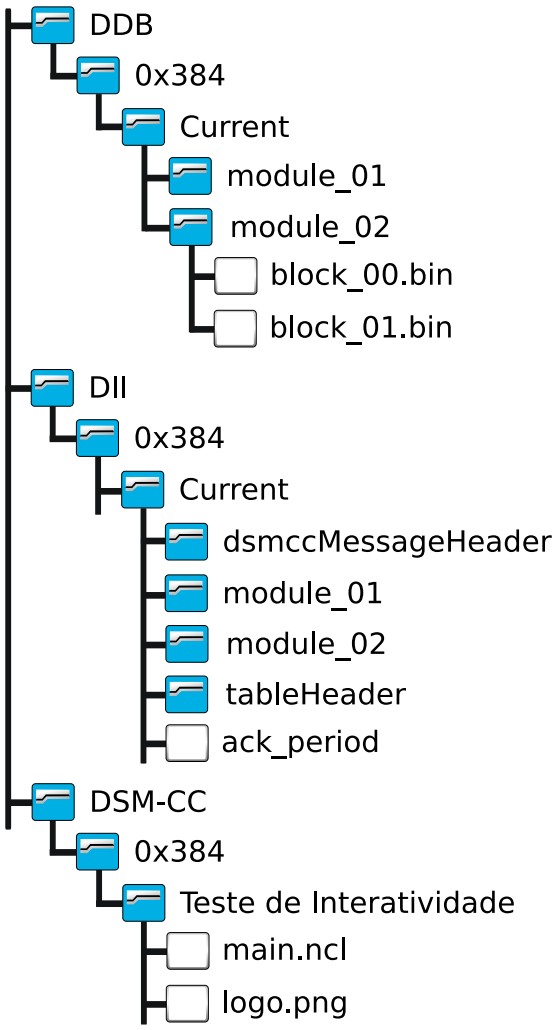

Figura 6.13: Tabelas DII e DDB, com expansão do sistema de arquivos do DSM-CC

A tabela DDB é também armazenada em um diretório `DDB` na raiz do sistema de arquivos. Duas formas de representação do conteúdo transportado na tabela DDB são oferecidas pelo DemuxFS, permitindo a depuração dos arquivos transmitidos nas mensagens BIOP ou a verificação de consistência de cada módulo e bloco dos carrosséis de dados e objetos. Na primeira opção, a árvore completa descrita no protocolo DSM-CC é extraída e armazenada em um diretório persistente estipulado pelo usuário, apontado no sistema de arquivos do DemuxFS pela ligação simbólica denominada `DSM-CC`. Esse cenário é mostrado na Figura 6.13. Na segunda opção, não há expansão da árvore de diretórios do DSM-CC. Nesse caso, cada módulo i recebido é armazenado em um diretório `module_`i e cada

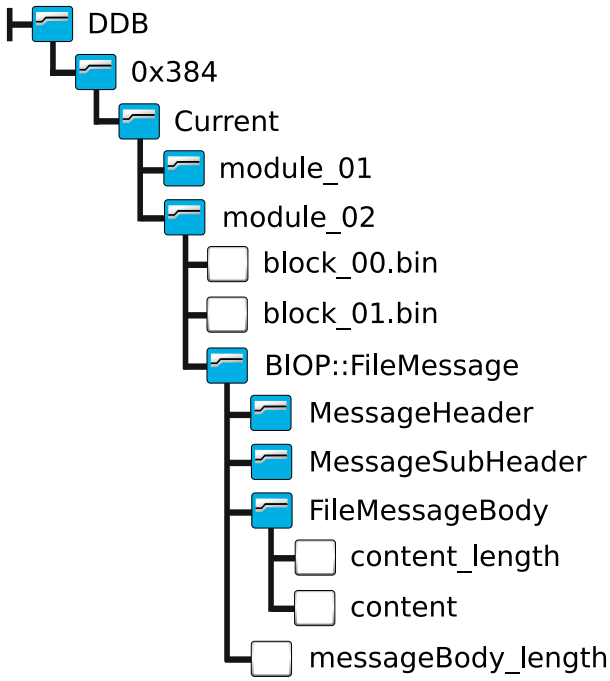

Figura 6.14: Tabela DDB, sem expansão do sistema de arquivos do DSM-CC

um dos blocos j que compõem esse módulo é exibido em `module_`i`/block_`j`.bin`. Uma ligação é também usada para a área persistente, visto que o tamanho dos dados transferidos pode ultrapassar os limites de memória RAM disponíveis na plataforma hospedeira do DemuxFS. A Figura 6.14 mostra a organização da hierarquia segundo esse critério.

Em ambos os casos, duas ligações simbólicas são criadas no DemuxFS. A primeira fica dentro do diretório da PMT que anuncia as tabelas de carrossel de dados ou objeto, apontando para o diretório `/DDB/<PID>/<Versão>`, enquanto a segunda é criada no diretório `/Streams`, apontando para o diretório da PMT. Com essas ligações simbólicas todas as informações referente a um programa específico ficam disponíveis em um ponto central (o diretório da PMT), ao mesmo tempo em que todos os fluxos elementares de áudio, vídeo e dados transmitidos ficam convenientemente centralizados em outro diretório (`/Streams`), facilitando a localização de dados frequentemente observados.

6.3.4 Medidas Métricas para Análise Quantitativas

Um diretório denominado `Reports` abriga na estrutura do DemuxFS relatórios indicativos de erros e avisos gerados durante o processamento dos pacotes do fluxo de transporte. Cada evento é acompanhado da data e hora, tornando possível a geração de gráficos desses eventos em função do tempo ou a obtenção da taxa

média de erros de integridade de pacotes, por exemplo.

Nota-se, no entanto, que não existe integração automática do DemuxFS com ferramentas de traço de gráficos. Essa funcionalidade requer que uma aplicação extraia a informação que for considerada relevante dos arquivos de relatório para que então os dados sejam encaminhados a uma biblioteca ou programa apropriado.

6.4 Síntese

Neste capítulo foi apresentado o DemuxFS, uma implementação de referência do sistema de arquivos para a análise do fluxo de transporte do SBTVD. Com o DemuxFS se mostrou ser possível expor os objetos de análise mais comuns ou desejáveis em uma aplicação especializada por meio de conceitos simples e padronizados, validando a arquitetura proposta.

Graças à escolha de uma plataforma de desenvolvimento em espaço de usuário (FUSE) foi possível integrar um recurso de decodificação de vídeo em *software*, permitindo a visualização imediata do quadro de vídeo transmitido naquele instante, sem que seja necessário um decodificador em *hardware* ou uma plataforma com alto poder de processamento para decodificar o fluxo de vídeo continuamente. Ainda assim o acesso direto aos fluxos elementares é oferecido, por meio dos arquivos FIFO, possibilitando sua aquisição ou reprodução contínua por *software* ou *hardware*, dependendo das capacidades da plataforma e seus decodificadores. Essas funcionalidades permitem que a arquitetura proposta seja utilizada para finalidades além da análise de dados, uma vez que sua infraestrutura acolhe os recursos necessários para a implementação de um sistema completo de um receptor de TV digital.

Com a inclusão do suporte a uma plataforma de *hardware* dedicada de desenvolvimento nacional, a validação e análise de fluxos de transporte pode ser feitas em tempo real pelo próprio dispositivo utilizado para assistir TV. Este é um grande diferencial da implementação realizada, considerando a inexistência desse tipo de recurso em produtos comerciais encontrados no mercado.

Espera-se com o DemuxFS que um novo conceito em serviço seja agregado aos terminais de acesso de TV digital, tornando o acesso aos elementos transmitidos no fluxo de transporte trivial e universal.

7 Conclusões

Fluxos de dados estruturados apresentam um fator especialmente importante para a sua análise: a informação transmitida é potencialmente infinita. Essa característica leva à necessidade da aplicação de técnicas que permitam extrair os elementos essenciais sem acarretar um impacto negativo no desempenho da análise ou nos padrões de uso de memória do receptor.

Este trabalho apresentou uma proposta de arquitetura para a análise de fluxo de dados estruturados, na qual cada elemento estruturado é mapeado a um conjunto de arquivos e diretórios. A leitura de metadados e a realização de operações especiais são possíveis por meio de operações padronizadas sobre esses tipos de objeto. A definição de arquivos e regras especiais para lidar com elementos de grande vazão mostrou que essa arquitetura pode se aplicar à análise de sistemas multimídia, como o adotado pelo Sistema Brasileiro de TV Digital (SBTVD).

7.1 Contribuições

A arquitetura proposta traz uma contribuição concreta à área pela sua originalidade. A literatura aponta ferramentas similares, embora não haja um compromisso destas com a representação dos elementos analisados em formatos portáveis e interoperáveis. A disponibilidade de uma implementação de uma arquitetura poderosa de análise de fluxo de dados estruturados é original por permitir o tratamento de fluxos de dados complexos, como o SBTVD, usando ferramentas padronizadas para a manipulação de sistemas de arquivos.

Para a validação desta proposta, um sistema de arquivos para a análise do fluxo de transporte do SBTVD foi implementado. A simplicidade na qual a informação hierárquica de cada elemento é acessada mostrou que essa arquitetura é viável e escalável. Além disso, a possibilidade de execução em uma plataforma de TV digital dedicada e de produção nacional permite incorporar essa funcionalidade a produtos de fácil acesso e de baixo custo, facilitando a análise de fluxos de transporte em diferentes pontos do país.

Outra contribuição relevante deste trabalho é o próprio sistema de arquivos implementado. O DemuxFS, distribuído sob uma licença de *software* livre [1], possibilita a análise de fluxos de transporte transmitidos por emissoras homologadas para transmissão no território brasileiro e representa uma importante adição para a comunidade acadêmica e tecnológica. Dessa forma, a relevância deste trabalho no contexto do SBTVD é um dos seus principais méritos.

7.2 Trabalhos futuros

Junto à sua implementação de referência, essa arquitetura abre diversas opções de trabalhos futuros, sejam na forma de extensões da ferramenta desenvolvida ou no acoplamento de módulos para a análise dos elementos disponibilizados nos objetos do tipo FIFO do sistema de arquivos – como analisadores de protocolos de áudio, vídeo e legendas.

A arquitetura permite ainda que outras fontes de entrada de dados sejam utilizadas, tornando possível a análise de fluxos de transporte oriundos da camada de rede, por exemplo. Dessa forma, um trabalho derivado de bastante relevância para o cenário de TV digital encontra-se na adição de suporte a tecnologias de Televisão sobre IP, também chamado de IPTV, na qual o fluxo de transporte é encapsulado em outros protocolos como o IGMP (do inglês *Internet Group Management Protocol*, ou Protocolo de Gerenciamento de Grupos na Internet) e o RTSP (*Real Time Streaming Protocol*, ou Protocolo de Fluxo de Dados em Tempo Real) (SILVERSTON et al., 2009).

A arquitetura proposta pode também ser estendida para permitir a escrita em determinados arquivos e diretórios para criar fluxos de transporte derivados daquele recebido pelo módulo de aquisição de dados. Esse processo, conhecido como remultiplexação, permite a simulação (ou correção) de erros e a inclusão de dados adicionais, como programas interativos, que não se encontravam no fluxo de transporte original. HIRAYAMA e Silveira (2006) apresenta alguns dos problemas relacionados a este tema que precisam ser considerados ao se produzir fluxos de transporte derivados de um outro.

Mostra-se factível também o desenvolvimento de aplicações interativas de reprodução de mídias digitais acerca da infraestrutura do DemuxFS, visto que toda a informação necessária por aplicações deste gênero se encontram para fácil acesso no sistema de arquivos exposto.

[1] O código fonte do projeto está disponível em http://demuxfs.sourceforge.net

Este tópicos estão diretamente alinhados com as linhas de pesquisa do Laboratório do Sistemas Integráveis da USP e devem se concretizar em projetos no decorrer dos próximos anos.

Anexo A – Atributos Estendidos em Sistemas de Arquivos Legados

A definição de interfaces de programação portáveis para a implementação de atributos estendidos em sistemas operacionais só ocorreu com a publicação, em 1998, do padrão POSIX 1003.1e, que trata de interfaces para listas de controle de acesso, separação de privilégios de usuários, controle de acesso mandatório e mecanismos para rotular informações (IEEE, 2001). Dessa forma, diversos sistemas de arquivos legados não contemplam um suporte efetivo a esse recurso. Esse é o caso do FAT32, que se tornou um *padrão de fato* devido à popularidade do sistema operacional que o originou.

A especificação do formato em disco do sistema de arquivos FAT32, descrita em (MICROSOFT, 2000), foi feita com base no FAT16 e FAT12, desenvolvidos originalmente pelo sistema de arquivos Microsoft MS-DOS em meados de 1980. A diferença básica entre esses arquivos e a razão para seus nomes é o tamanho, em bits, dos objetos presentes na estrutura FAT em disco. São 12 bits para um objeto na FAT12, 16 bits para um objeto na FAT16 e 32 na FAT32.

Duas estruturas de dados são contempladas e compartilhadas nos formato FAT12, FAT16 e FAT32: a estrutura FAT, que abriga informações quanto ao volume (seu formato, capacidade, identificação e outros pertinentes ao setor de boot), e a estrutura de diretórios, que contém dados como o nome do objeto, seus atributos de acesso (modo leitura, oculto, de sistema), o endereço físico onde se encontra o primeiro agrupamento de dados (*cluster*) desse objeto e estampas de tempo, que indicam a data e hora de sua criação, modificação e quando ocorreu a última leitura de seu conteúdo.

O formato da estrutura de diretórios do FAT é definido conforme mostra a Listagem A.1 [1], na qual enfatiza-se a o membro `clusterHigh`. Pelas limitações impostas ao tamanho máximo dos objetos presentes no disco, tanto o FAT12

[1] Os tipos de dados `uint8_t`, `uint16_t` e `uint32_t` definem variáveis de tamanho de 1, 2 e 4 bytes, respectivamente. O tamanho do tipo `char` é também de 8 bytes.

quanto o FAT16 não utilizam o espaço reservado para essa variável [2], o que possibilita o seu uso para finalidades mais nobres.

Listagem A.1: Estrutura de diretórios do FAT

```
struct fat_dentry {
{
  char      name[8];       /* Nome do arquivo */
  char      ext[3];        /* Extensão do nome */
  uint8_t   attribute;     /* Máscara de atributos */
  uint8_t   reserved;      /* Reservado */
  uint8_t   createTimeMs;  /* Hora de criação (ms) */
  uint16_t  createTime;    /* Hora de criação (seg) */
  uint16_t  createDate;    /* Data de criação */
  uint16_t  accessDate;    /* Data de último acesso */
  uint16_t  clusterHigh;   /* Endereço alto do 1o cluster */
  uint16_t  updateTime;    /* Hora de última atualização */
  uint16_t  updateDate;    /* Data de última atualização */
  uint16_t  clusterLow;    /* Endereço baixo do 1o cluster */
  uint32_t  size;          /* Tamanho em bytes */
};
```

O campo de uso livre passou a ser utilizado pelo sistema operacional IBM OS/2 para implementar o recurso de atributos estendidos, que já estava presente em seu sistema de arquivos HPFS (*High Performance File System*). Os 2 bytes livres foram usados para indexar o primeiro de uma lista de *clusters* no disco que armazenam os atributos estendidos do arquivo. No entanto, essa lista não pode ser simplesmente gravada em unidades lógicas livres pois, por não fazerem parte da lista encadeada iniciada no endereço indicado por `clusterLow`, o sistema de arquivos ainda os considera como blocos livres, possibilitando a perda de metadados caso eles sejam atribuídos para uso pelo sistema de arquivos. A solução encontrada para armazená-los de forma persistente e segura foi associar cada *cluster* de atributo estendido a um arquivo real [3], que os mantém em uma lista encadeada iniciada no endereço válido `clusterLow`.

No sistema de arquivos FAT32 os 2 bytes do `clusterHigh` são usados para compor o endereço dos objetos no disco, que podem ser indexados de 0 bytes a 4

[2] Sua existência, bem como a do campo reservado, se justifica pela projeção da estrutura de diretórios FAT com base no sistema de arquivos do CP/M, que a utilizava para descrever a localização no disco dos *clusters* que compunham um arquivo.

[3] Esse arquivo tradicionalmente recebe o nome de `EA DATA.SF` nos sistemas que o implementam.

GiB. Como a estrutura de diretórios do FAT é compartilhada pelo FAT12, FAT16 e FAT32, não existe, até a presente data, uma maneira segura para armazenar atributos estendidos nesse sistema de arquivos.

Anexo B – Disponibilidade do Código Fonte do DemuxFS

O código fonte da implementação de referência apresentada nesta dissertação encontra-se no CD-ROM em anexo. A sua versão mais recente pode ser sempre obtida no endereço de internet `http://demuxfs.sourceforge.net`.

Referências Bibliográficas

ABNT. *ABNT NBR 15603-2: Televisão digital terrestre - Multiplexação e serviços de informação (SI). Parte 2: Sintaxes e definições da informação básica de SI*. Rio de Janeiro, 2007.

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

ABNT. *ABNT NBR 15601: Televisão digital terrestre - Sistema de transmissão*. Rio de Janeiro, 2008.

ABNT. *ABNT NBR 15602-3: Televisão digital terrestre - Codificação de vídeo, áudio e multiplexação. Parte 3: Sistemas de multiplexação de sinais*. Rio de Janeiro, 2008.

ABNT. *ABNT NBR 15604: Televisão digital terrestre - Receptores*. Rio de Janeiro, 2008.

ABNT. *ABNT NBR 15608-2: Televisão digital terrestre - Guia de operação. Parte 2: Codificação de vídeo, áudio e multiplexação - Guia para implementação da NBR 15602:2007*. Rio de Janeiro, 2008.

ARIB. *Service Information for Digital Transmission System – ARIB STD-B10, version 4.6*. Nittochi Bldg. 11F, 1-4-1, Kasumigaseki, Chiyoda-ku, Tokyo 100-0013, Japan, 2008.

ATSC. *ATSC Digital Television Standard – Part 1: Digital Television System (A/53, Part 1:2007)*. 1750 K Street, N.W., Suite 1200, Washington, D.C. 20006, 2007.

BEDICKS JR, G. *Sintonizador-demodulador para o sistema brasileiro de TV Digital*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, 158, Trav. 3 - Butantã, São Paulo/SP, Brasil, 2008.

BELLARD, F. *FFmpeg*. 2000. Disponível em: <http://ffmpeg.org>. Acesso em: 01 de Março de 2009.

BENVENUTI, C. *Understanding Linux Network Internals*. [S.l.]: O'Reilly Media, Inc., 2005. ISBN 0596002556.

BRODER, A. Z. et al. Min-wise independent permutations. *Journal of Computer and System Sciences*, v. 60, n. 3, p. 630–659, 2000.

CARVALHO, E. R. de. *Uma Plataforma Modular para Testes com Interatividade na TV Digital Brasileira*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, Av. Prof. Luciano Gualberto, 158, Trav. 3 - Butantã, São Paulo/SP, Brasil, Jul 2008.

CHERNOCK, R. S. et al. *Data Broadcasting: Understanding the ATSC Data Broadcast Standard*. New York, NY, EUA: McGraw-Hill Professional, 2001. ISBN 0071375902, 9780071375900.

CORMEN, T. H. et al. *Introduction to Algorithms, Second Edition*. [S.l.]: The MIT Press, 2001. Paperback. ISBN 0-262-53196-8.

CORMODE, G. Fundamentals of analyzing and mining data streams. *European Workshop on Data Stream Analysis*, San Leucio, Italy, Mar 2007.

CORMODE, G.; MUTHUKRISHNAN, S. An improved data stream summary: The count-min sketch and its applications. *LATIN 2004: Theoretical Informatics*, p. 29–38, 2004.

CRANOR, C. et al. Gigascope: A stream database for network applications. In: *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. New York, NY, USA: ACM, 2003. p. 1–1.

DEKTEC. *DTC-320 StreamXpert*. Van Riebeeckweg 43A - 1212 EH Hilversum, Holanda: DekTec Digital Video BV, Mai 2006. Disponível em: <http://www.dektec.com>. Acesso em: 01 de Março de 2009.

DIMITROPOULOS, X. et al. The eternal sunshine of the sketch data structure. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 52, n. 17, p. 3248–3257, 2008. ISSN 1389-1286.

DRISCOLL, E.; BEAVERS, J.; TOKUDA, H. *FUSE-NT: Userspace File Systems for Windows NT*. 2008. Disponível em: <http://pages.cs.wisc.edu/ driscoll/fuse-nt.pdf>. Acesso em: 01 de Março de 2009.

ELECARD. *Elecard Stream Analyzer*. 1700 South Amphlett Blvd, Suite 200. San Mateo, CA 94402, EUA: [s.n.], Jul 2006. Disponível em: <http://www.elecard.com>. Acesso em: 01 de Março de 2009.

ETSI. *Digital Video Broadcasting (DVB): Framing structure, channel coding and modulation for 11/12 GHz satellite services*. European Telecommunications Standards Institute. 650 Route des Lucioles - Sophia Antipolis. Valbonne, FRANCE, 1997.

GIRÃO, G. et al. Implementation of a HDTV transport stream multiplexer based on ITU-T H.222.0 recommendation. In: *WebMedia '05: Proceedings of the 11th Brazilian Symposium on Multimedia and the web*. New York, NY, EUA: ACM, 2005. p. 1–9.

HANKERSON, D. R.; JOHNSON, P. D.; HARRIS, G. A. *Introduction to Information Theory and Data Compression*. London, UK, UK: Chapman & Hall, Ltd., 2003. ISBN 1584883138.

HIRAYAMA, R. M.; SILVEIRA, R. M. *Framework para testes e avaliação de sincronismo para aplicações de TV digital móvel*. Dissertação (M.Sc.) — University of São Paulo, São Paulo, 2006. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3141/tde-19092006-172310/>.

IEEE. *IEEE Standard for Information Technology: Portable Operating Sytem Interface (POSIX). Part 1: System Interface*. 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA: IEEE Standards Association, 2001.

INFANTE, S.; NASIOPOULOS, P. Real-time DVB-MHP interactive data transcoding to blu-ray. *International Journal of Digital Multimedia Broadcasting*, v. 2008, p. 18, 2008.

ISO. *ISO 8879:1986, Information processing – Text and office systems – Standard Generalized Markup Language (SGML)*. Geneva, Switzerland: International Organization for Standardization, 1996.

ISO. *ISO/IEC 13818-1, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Systems*. Geneva, Switzerland: International Organization for Standardization, 1996.

ISO. *ISO/IEC 13818-2, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Video*. Geneva, Switzerland: International Organization for Standardization, 1996.

ISO. *ISO/IEC 13818-6, Information Technology – Generic Coding of Moving Pictures and Associated Audio Information: Extensions for DSM-CC*. Geneva, Switzerland: International Organization for Standardization, 1996.

ISO/IEC. *ISO/IEC 26300:2006, Information technology – Open Document Format for Office Applications (OpenDocument) v1.0*. Geneva, Switzerland: International Organization for Standardization, 2006.

ITU-T. *H.264 : Advanced video coding for generic audiovisual services*. Place des Nations, 1211 Geneva 20, Switzerland, 2007.

KAMINA, T.; TAMAI, T. Embedding XML processing toolkit on general purpose programming language. *Asia-Pacific Software Engineering Conference*, IEEE Computer Society, Los Alamitos, CA, USA, v. 0, p. 159, 2002. ISSN 1530-1362.

KAY, D. C.; LEVINE, J. R. *Graphics Files Formats*. 2. ed. New York: Windcrest/McGraw-Hill, 1995. ISBN 0-07-034025-0.

KILLIAN, T. J. Processes as files. In: *Proceedings of the USENIX Summer Conference*. Salt Lake City, Utah: USENIX Association, 1984. p. 203–207.

LEWKOWICZ, J. M. *The complete MUMPS: an introduction and reference manual for the MUMPS programming language*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989. ISBN 0-13-162125-4.

MATTHEW, T. Gb2393069: Monitoring program map tables of a compressed transport stream. Mar 2004. Disponível em: <http://www.wikipatents.com/gb/2393069.html>. Acesso em: 01 de Março de 2009.

MATTHEW, T. Gb2393070: Monitoring service description tables of a compressed transport stream. Mar 2004. Disponível em: <http://www.wikipatents.com/gb/2393070.html>. Acesso em: 01 de Março de 2009.

MCDOUGALL, R.; MAURO, J. *Solaris Internals: Solaris 10 and OpenSolaris Kernel Architecture*. Second. Upper Saddle River, NJ, USA: Sun Microsystems Press/Prentice Hall, 2007. 1072 p. ISBN 0-13-148209-2 (hardback).

MICROSOFT. *FAT32 File System Specification – FAT: General overview of on-disk format, version 1.03*. Dez 2000. Disponível em: <www.microsoft.com/whdc/system/platform/firmware/fatgen.mspx>. Acesso em: 01 de Março de 2009.

MUHAMMAD, H. H.; DETSCH, A. Uma nova proposta para a árvore de diretórios unix. In: UNIVERSIDADE DO VALE DO RIO DO SINOS. *III Workshop sobre Software Livre (WSL2002)*. [S.l.], 2002.

MUTHUKRISHNAN, S. Data streams: algorithms and applications. In: *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2003. p. 413–413. ISBN 0-89871-538-5.

MUTHUKRISHNAN, S. *Data Streams: Algorithms and Applications*. [S.l.]: Now Publishers Inc, 2005. ISBN 193301914X.

OFFICE, A. P. *The Patents and Designs Journal*. Newport, South Wales NP10 8QQ, 2004. Edição número 5991.

PEREIRA, M. M. et al. Projeto e implementação de um multiplexador de transport streams com suporte a HDTV. In: *IV Workshop Técnico Científico do DIMAp*. [S.l.]: Natal: EDUFRN, 2005. p. 48–59.

PIKE, R. et al. Plan 9 from bell labs. In: *Proceedings of the Summer 1990 UKUUG Conference*. Buntingford, UK: UKUUG, 1990. p. 1–9. ISBN 0-9513181-7-9.

PIKE, R. et al. The use of name spaces in plan 9. In: *EW 5: Proceedings of the 5th Workshop on ACM SIGOPS European Workshop: Models and paradigms for distributed systems structuring*. New York, NY, EUA: ACM, 1992. p. 1–5.

SCHERG, R. *DVB Snoop*. 2001. Disponível em: <http://dvbsnoop.sourceforge.net>. Acesso em: 01 de Março de 2009.

SGI. *IRIX Admin: Disks and Filesystems*. [S.l.]: Silicon Graphics Inc., 1996. IRIX 6.2 Reference Manual, Document Number: 007- 2825-001.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. *Operating System Concepts*. 7th. ed. Hoboken, NJ, EUA: John Wiley & Sons, Inc., 2004. ISBN 0-471-69466-5.

SILVERSTON, T. et al. Traffic analysis of peer-to-peer IPTV communities. *Computer Networks*, v. 53, n. 4, p. 470–484, 2009.

SINGH, A. *Mac OS X Internals: A Systems Approach*. 3rd. ed. Upper Saddle River, NJ, EUA: Addison-Wesley Professional, 2006. ISBN 0-321-27854-2.

SZEREDI, M. *File system in user space (FUSE)*. 2009. Disponível em: <http://fuse.sourgeforge.net>. Acesso em: 01 de Março de 2009.

VITTER, J. S. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, ACM, New York, NY, USA, v. 11, n. 1, p. 37–57, 1985. ISSN 0098-3500.

W3C. *World Wide Web Consortium*. 1994. Disponível em: <http://www.w3.org>. Acesso em: 01 de Março de 2009.

WEIDERHOLD, G. (Ed.). *File organization for database design*. New York, NY, USA: McGraw-Hill, Inc., 1987. ISBN 0-070-70133-4.

WEINBERGER, P. J. The version 8 network file system. In: *Proceedings of the USENIX Summer Conference*. Salt Lake City, Utah: USENIX Association, 1984.

YAMADA, F. et al. *Revista de Engenharia e Computação*. [S.l.]: Editora Mackenzie, 2004. Ano 5.

ZANALYZER, I. *Z1 DTSA*. Smithfield, Rhode Island, EUA: [s.n.], 2008. Disponível em: <http://zanalyzer.com>. Acesso em: 01 de Março de 2009.